Politecnico di Torino

Master of Science Degree in MECHATRONIC ENGINEERING

Master Thesis

Person tracking methodologies and algorithms in service robotic applications

Supervisor:
prof. Marcello Chiaberge

Candidate:
Anna Boschi
S253105

Academic year 2018 - 2019

Abstract

The vital statistics of the last century highlight a sharply increasement of the average life of the world population with a consequent growth of the number of elderly people. This scenario has caused new social needs that the research in the service robotics field is trying to fulfill. Particularly, the idea of this thesis is born at the PIC4SeR (PoliTo interdepartmental centre for service robotics) with the purpose of creating complex service robotics applications to support the autonomous and self-sufficient old people into their house in everyday life, avoiding the task of monitoring them by third parties. This work represents the first steps of a broad project in which many other service tasks will be integrated.

The main argument of this thesis is to develop algorithms and methodologies to detect, track and follow a person in an indoor environment using a small wheeled rover and low cost and available sensors to monitor the target person. Several techniques are explored showing the evolution of these methods along the years: from the classical Machine Learning algorithms to the Deep Neural Network ones. Since the main requirement to be respected is the necessity of real-time results, only few of the analysed algorithms are developed for this project scope and at the end are compared in order to find the best solution with optimal outcomes. The detection and localization are the basis of the person tracking application, done by the robot on which it has been implemented a movement control algorithm and at last it has been introduced an obstacle avoidance algorithm to prevent collisions.

Contents

List of Figures

Chapter 1

Introduction

1.1 Objective of the thesis

The vital statistics of the last century highlight a sharply increasement of the average life of the world population with a consequent growth of the number of elderly people. This scenario has caused new social needs that the research in the service robotics field is trying to fulfill. Particularly the attention has been focused on the assistive system in order to promote the ageing-in-place and to make the independent indoor life easy [1]. The idea of this thesis is born at the PIC4SeR (PoliTo interdepartmental centre for service robotics) with the purpose of creating complex service robotics applications to support the autonomous and self-sufficient old people into their houses in everyday life, avoiding the task of monitoring them by third parties. This work represents the first steps of a broad project in which many other service tasks will be integrated.

The goal of the thesis is to develop algorithms and methodologies to detect, track and follow a person in an indoor environment using a small wheeled rover and available low cost sensors to monitor the target person.
Several techniques have been analysed showing the evolution of the object detection methods along the years: from the Machine Learning algorithms to the Deep Neural Network ones. Since the main requirement to be respected is the necessity of real-time results, only few of them are developed for this scope and at the end are compared in order to find the best solution with optimal outcomes.
Many hardware configurations are analysed in order to find a good compromise between the external payload of the robot and the performances required by the algorithms. The detection and localization are done using different kind of data provided by a single stereo-camera and the robot control movement is developed in order to follow correctly the person, remaining at a certain safety distance from it. To conclude, a basic obstacle avoidance algorithm is integrated in order to prevent collisions.

1.2 Organization of the thesis

The thesis is composed of eight chapters and the work is organized as follows:

Chapter 1 is introductory and gives the motivations that stimulate researchers in the service robotics for elderly people field. Next, the goal of the thesis is explained and its organization structure is provided.

In chapter 2 there is an overview of all the object detection methods developed during the years starting from the classical Machine Learning algorithms to the Deep Neural Networks ones.

Chapter 3 provides a detailed study of the Y.O.L.O. networks starting from the first version and showing the improvements obtained in the following ones.

Chapter 4 is functional to explain the parameters used to evaluate the performance of the neural networks and the re-training operation on them.

At the beginning of chapter 5 a short overview of the types of robots and sensors, used for different applications, is presented. After that, the Hardware configuration tried to best satisfy these application requirements and the Software used in the thesis are described.

In chapter 6 are explicated all the implementations realized to do person tracking with the different algorithms chosen including the control of the robot's movement.

In chapter 7 there is an overview of the obstacle avoidance algorithms and it is explained the implementation of a specific one of them into this project.

In chapter 8 the results obtained from the implementations are presented, finding also the best solution between the developed ones. At the end the conclusions and some future works ideas are given.

Chapter 2

Object Detection

2.1 Introduction

Object detection is an important area of research interested into the processing of images and videos to detect and recognise object. This sector is called *Computer Vision* and it is possible to see its evolution in the literature: from the classical algorithms to the ones that used the Deep Neural Networks technologies.
The Computer vision discipline was born in the late 1960s in universities pioneering artificial intelligence, but only the studies of the 1970s produced algorithms used also nowadays as labelling of lines, extraction of edges, segmentations and others.
The evolution of these techniques of machine learning permits to do two different actions: to localize and identify multiple objects in a single image or video frame. These actions are very similar, but not the same, as the figure 2.1 shows.

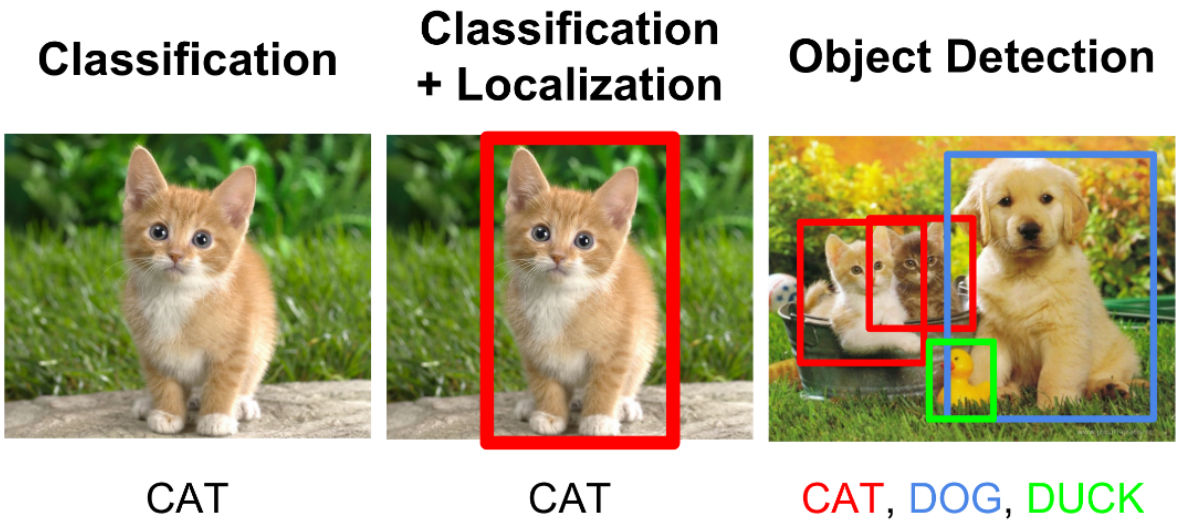


FIGURE 2.1: Object Classification, Localization and Detection [2].

For the localization it is necessary to identify the image under a specific category. Differently, for the identification, it is necessary the localization of the objects, that

are recognised into the image, and then this information must be processed. These two methods are very different in complexity and results, but the best thing to do is to realize an algorithm able to do both the actions with great performances. All the objects belong to a class, identified by some features. If the object detected in the figure has these specific features, it belongs to this class, so it is possible to identify it.

Region of Interests (RoIs)

The object detection framework can be divided in three steps [3]:

1. Generation of Regions of Interests (RoIs): an algorithm is used to create a large set of bounding boxes crossing the entire image;
2. From all the bounding boxes created, visual features are extracted and are evaluated in order to classify the object detected, if present, on each RoI;
3. The overlapping boxes, which contain the same object, are merged into a unique box containing the overall detected element.

In the figure 2.2 it is possible to see an example of this concept.

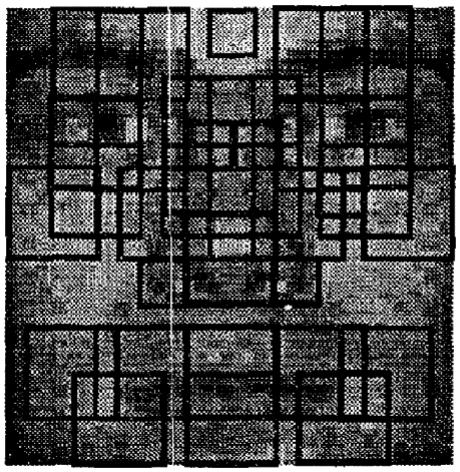


FIGURE 2.2: RoIs example [4].

Image Gradient Vector

During the evolution of the object detection, the creation of RoIs can be implemented in different ways and all these methods are based on the *Image Gradient Vector*, used to determine the intensity and the edge direction in a (x, y) position. The gradient contains partial derivatives, computed as the colour difference among the adjacent

pixels of the image along the principal directions. It is important to notice that, in this case, the gradient is discrete because each pixel is independent from the others and cannot be split anymore:

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

This formula can be easily understood using the figure 2.3.

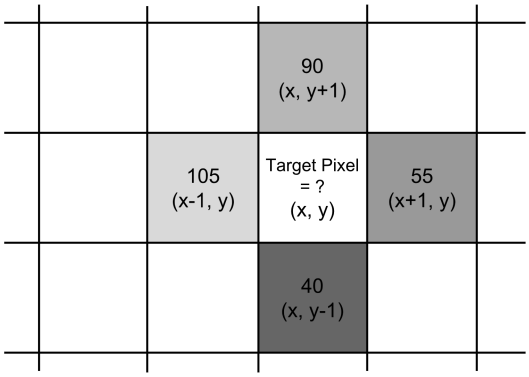


FIGURE 2.3: Gradient vector computation [5].

The gradient gives information about two values used in the algorithms:

- **Direction** it is given by the arctangent of the ratio between two partial derivatives on two directions: $\theta = \arctan(g_y/g_x)$
- **Magnitude** it is calculated as the L2-norm of the vector: $g = \sqrt{g_x^2 + g_y^2}$

The application of this process to every pixels of the image is a big waste of time. The solution is to include this operation inside a convolutional operator on the whole image matrix. In this case, the operator depends on the algorithm and have a specific purpose.

2.2 Evolution of Object Detection

Here the main solutions and methods used for object detection in its evolution are reported.

2.2.1 Viola-Jones algorithm

The *Viola-Jones image-base algorithm* was developed in 2001 and for that year was very powerful even if simple [6].
It uses an image-based method able to relate an input image to the class selected or not. The algorithm consists of two stages: *training* of the *classifier* and detection of the object.

During the detection, if the trained classifier is not able to find the object (*false negative*) or differently detected a wrong object (*false positive*) it is necessary to retrained the classifier, including these examples in the training set.
To do the detection, it is necessary to turn on the image in greyscale and divide it in sub-windows. Inside these sub-windows the classifier tries to find the features of the class that should be detected. Only when all the necessary features of the class are detected, it traces a box around the sub-windows identified and the box remains fixed also when the image is turning on the RGB scale.
There are three main contributions to this approach:

- Haar Feature-based Cascade Classifier;
- AdaBoost;
- Haar Cascade Classifier.

Haar Feature-based Cascade Classifiers

This method has its bases in the machine learning. It uses a classifier trained with multiple positive and negative images, which later is applied on other test images. During the training, many features are extracted from the examples and the more discriminant ones are selected and set inside the statistic model. For this operation many Haar features (Figure 2.4) are used like a kernel.

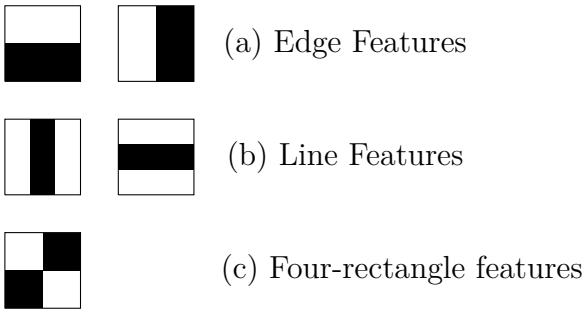


FIGURE 2.4: Haar features representation [6].

Since an image is composed by pixels partially black and white, dividing the image in part, depending on the luminosity, it is possible to identify the different features in the image and so to recognise the object to be detected. In fact, each feature has a value obtained by the subtraction between the pixels sum under the black rectangle and the pixels sum under the white rectangle.
Even if the computation for this calculus is huge, the solution can be found in the Integral Image. The integral image contains at the (x, y) location of the image the sum of the pixels above and to the left of the position inclusive:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y')$$

$$s(x, y) = s(x, y - 1) + i(x, y)$$

$$ii(x, y) = ii(x - 1, y) + s(x, y)$$

In this way the calculation is accelerated and makes possible the integral image computation just passing one time only over the original image.
Anyway, it is evident that many of the calculated features are useless or relevant only in specific zones of the image, as it is showed in the figure 2.5. In fact, if they are shifted in another point of the image, they make no sense. This means that only few features can be combined into an effective classifier, but it is necessary to find them.

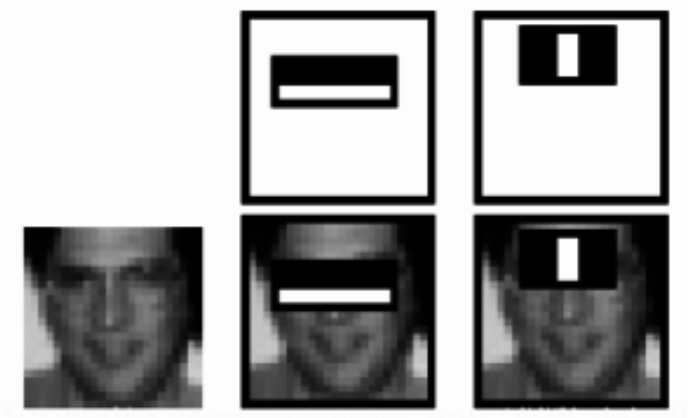


FIGURE 2.5: Haar features used for face recognition [7].

AdaBoost classifier

The AdaBoost algorithm is used to improve the performance of a simple classification algorithm. It is able to create a classifier by selecting a small set of important features. It consists in a modification of a Viola-Jones algorithm: every weak learner classifier is able to return the response using a single feature. There are many versions of this algorithm, but the most performant is the latest one, here, explained by steps:

- All the features are used for all the training image dataset;
- The algorithm connects the best threshold to each feature and gives a classification of the prediction as positive or negative;
- The previous point generates errors or wrong classifications that are useful to realize a rank of features saving only the one that provides accurate classification;
- This process is repeated until the right error rate satisfies the required parameters.

This improvement guarantees better performances and strong bounds in the generalization of the algorithm. However, this process is not so fast, even if the features used have been reduced to the minimum necessary. This problem is due to the application

of all the selected features to the entire image area, even if the object to be analysed is in a small zone, so there are some parts of the image that can be avoided in the process to waste less time.

Haarcascade Classifier

This technique is an improvement of the previous ones [8]. The Haarcascade Classifier is the best expression of the evolution of the machine learning classical method used for object detection.

Structure:
The algorithm consists in the construction of a cascade of classifiers in series to increase the detection results and reduce the computation time. This is a great intuition because the use of smaller boosted classifiers reduces the complexity of the algorithm, which is considered to run real-time.
The detection process can be explained with a degenerate decisional tree structure (figure 2.6) of classifiers commonly called "cascade". The process of detection consists in dividing the image in sub-windows and then analysing all of them with the first stage of classifier. The sub-windows that receive negative response are rejected, differently the ones that result positive are analysed using the second stage of classifier. The process continues until either all the areas of the image are rejected or there is a final positive response, so the object requested is detected in the image. In this way the time cost is reduced and the performance of the algorithm is high.
The first classifier is more general to do an initial considerable skimming, then the others are more specific so increase the accuracy of the results. The stage in cascade are constructed using AdaBoost for the training process and then thresholds regulation is done in order to reduce the false negatives.

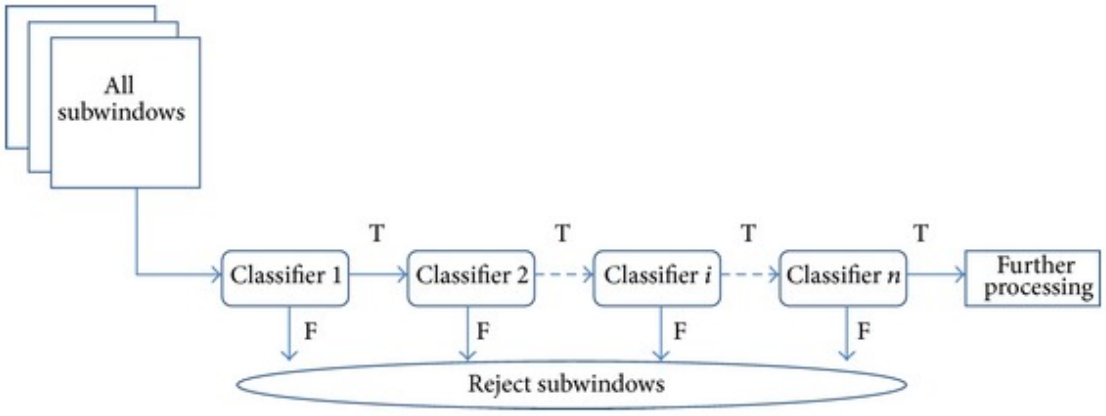


FIGURE 2.6: Haarcascade classifier structure [9].

Training
The training process of a cascade of classifier wants to balance two necessities: the accuracy and the cost time. In fact, the classifiers with more features reach high detection rates, but obtain lower false positives results and need a long computation time.
To obtain a good result is necessary to find a good compromise between the following points:

1. Number of stages of cascade classifier;
2. Number of features for each stage;
3. Thresholds applied to each stage.

in order to minimize the value of the evaluated features, but is not a simple choice.
Each stage of the classifier is trained by adding features until the false positive rates and the detection target rates are reached.

Results
The progress with this method is evident, but there could be some problems due to:

- False positives and negatives;
- Not detected object (mainly due to the excessive backlighting);
- C-error: the localization error (err<0.25 to have a correct localization).

2.2.2 Histograms of Oriented Gradients for Human Detection

Developed in 2005, the HOG algorithm uses the image gradient vector for extracting features out of the colours of pixels with great performance [10]. The aim of this method is to search the major gradient direction related to each sub-group of pixels, which indicates the flow of the image from light to dark. In the figure 2.7 it is possible to see an example of the image derived from the algorithm

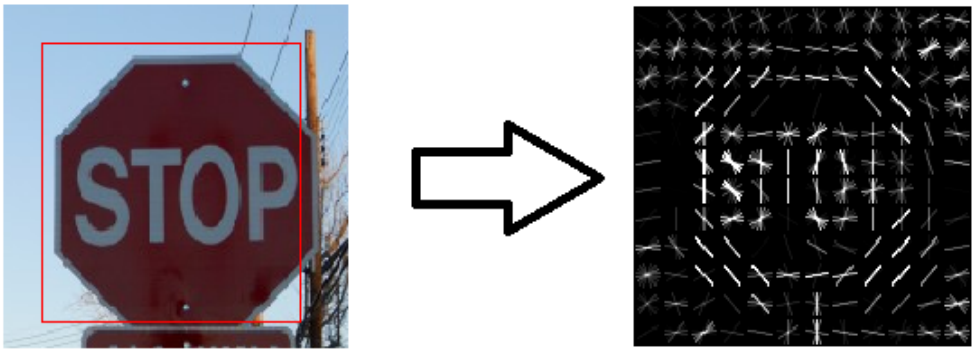


FIGURE 2.7: Results of an HOG application [11].

This method can be analysed in six steps (figure 2.8):

1 **Gamma/Colour Normalization:** all images have to have the same dimension of pixels in order to obtain correct and meaningful results;

2 **Gradient Computation:** the gradient is calculated in two main directions with a kernel and this value is used to obtain both the magnitude and the direction. In fact, the gradient eliminates the useless information of the image, giving importance only to the outlines;

3 **Spatial/Orientation Binning:** the image is divided in 8x8 pixels cells and there each magnitude is changed into 9 bins of unsigned direction, but, if a magnitude is set in the board of two bins, it is proportionally separated into the two;

4 **Normalization and Descriptor Blocks:** a 2x2 cells block slip on the image and in each region is created a one-dimensional vector of 36 values coming from four histograms of four cells and then is normalized in order to have a unit weight. At the end, the feature of the HOG is a vector generated from the concatenation of all the block vectors;

5 **Detector Window and Context:** the reduction of the 16 pixels' margin around the 64x128 detection window decreases the performance of about 3%;

6 **Classifier:** using a Gaussian kernel SVM improves the performance by about 3%, but there is a cost in run time.

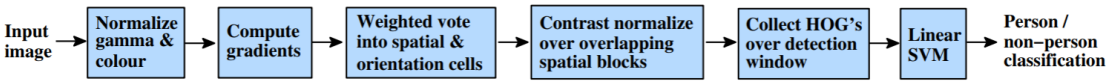


FIGURE 2.8: HOG steps structure [10].

2.2.3 CNN

At the beginning of Deep Learning the *Convolutional Neural Network* (CNN) was the most used architecture for classification [1].

Sliding-window detectors

One of the approaches for object detection is to divide the image in sub-windows of varied sizes and proportions and to identify the object using classification [13]. The sub-windows identified are cut out from the image and these patches are warped because the classifiers need fixed size images only, but this should not impact in the accuracy of the classification because the classifiers are trained to manage warped images.
The warped image patch is given to a CNN classifier to pull out 4096 features. After that, a SVM classifier is applied to it in order to find the class and a linear regression for the boundary box (figure 2.9).

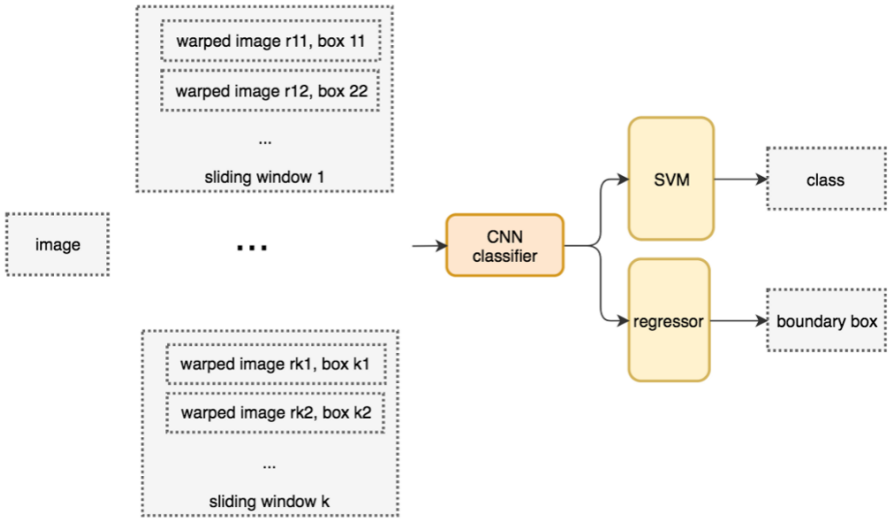


FIGURE 2.9: Procedure of the sliding-window detector [13].

In order to increase the performance of the algorithm, it is necessary to reduce the number of sub-windows.

Selective Search (SS)

Differently from the previous method, this one uses a regional proposal approach [14] [13] to create RoIs necessary for doing object detection (figure 2.10). Each pixel is considered as a group and for each group is calculated the texture matching the two ones that are strictly nearest. However, in order to avoid that every single region becomes part of the others, it is better to generate smaller group first. Then, the process of integrating region continues until everything is combined together.

[1] To read this section it is necessary to have knowledge about Machine Learning, Deep Learning and Convolutional Neural Networks [12].

FIGURE 2.10: Example of image processing using Selective Search [13].

Regional CNN (R-CNN)

R-CNN [15] is a combination between the SS and the CNN and is able to generate about 2000 RoIs. Each region is warped and passed to the CNN network individually, in this way it is followed by fully connected layers to define the bounding boxes and correctly classify the object (figure 2.11).

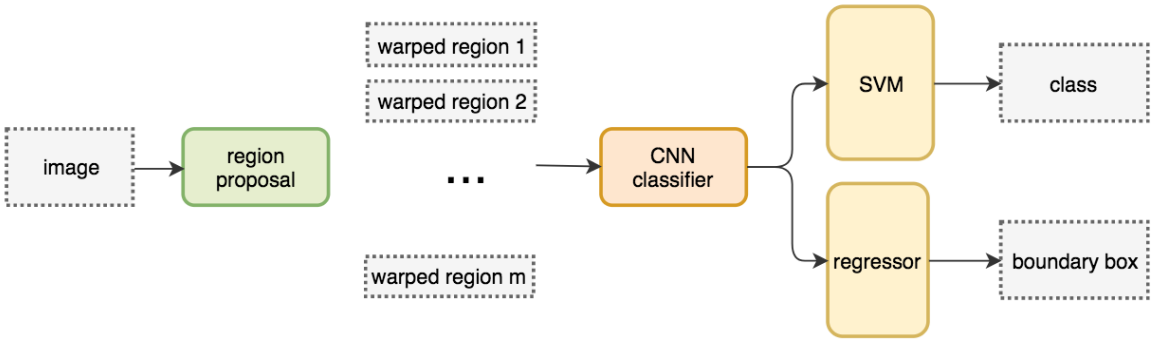


FIGURE 2.11: Procedure of the R-CNN [13].

This method needs a huge amount of time during the training of the network because it depends on to the large quantity of regional proposals per image to classify. This implies not real-time application: the process needs 50s for a single image.

Fast R-CNN

It is a great evolution of the R-CNN [16]: it does not extract the features for all images from scratch, but it uses a unique features extractor (a CNN) applied to the whole image first. Then it uses regional proposal method as SS in order to generate RoIs that will be combined with the corresponding features maps to create patches for object detection. These patches are wrapped to a fixed size with a pooling layer and fed to a fully convoluted network, able to classify and localize the object (figure 2.12).

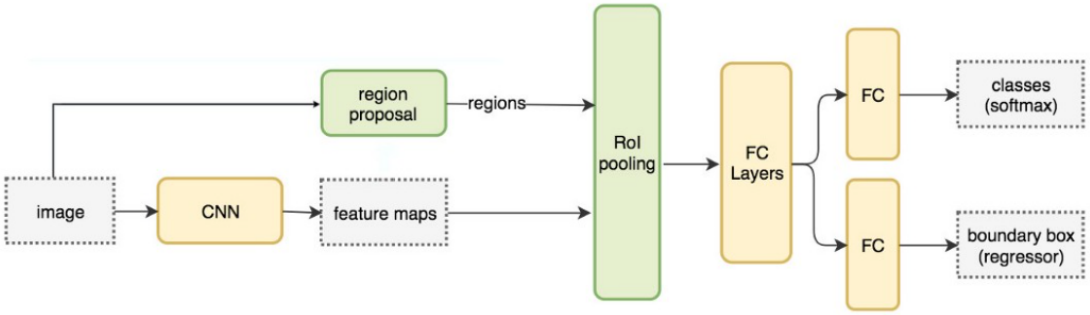


Figure 2.12: Procedure of the Fast R-CNN [13].

This network is faster than the previous one because it does not repeat the extraction of the feature one-by-one, but only once, so it is 10x faster during the training and 150x faster during the inferencing.

Faster R-CNN

R-CNN and Fast R-CNN algorithms use external regional proposal method like selective search. This usage affects the performance of the algorithm because it needs CPU to run and consequently waste much time. In order to reduce this problem, in the new version of the algorithm (the faster R-CNN [17]), this step is replaced with an internal deep network which directly derives the RoIs from the feature maps. The new network is called *Regional Proposal Network* (RPN) and is really efficient: it is able to process an image in about 10 ms (figure 2.13).
RPN makes prediction in order to recognise an anchor, a different size box which can contain the requested object on the foreground or on the background and refines it.

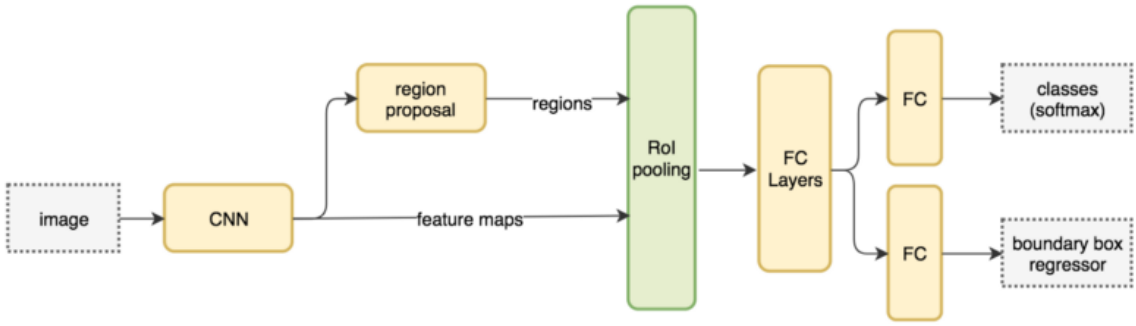


FIGURE 2.13: Procedure of the Faster R-CNN [13].

The remaining part of the network is the same of the fast R-CNN, replacing only the region proposal with a convolutional network.

2.2.4 Single Shot MultiBox Detector (SSD)

The acronym of the method stands for [18] [19]:

- **Single shot:** object localization and classification are both done in a single step forward over the network;
- **MultiBox:** it is the name of the bounding box regression technique developed by *Szegeday* et al;
- **Detector:** it means that the neural network detects and classifies the object found.

The SSD method is based on a feed-forward convolutional neural network able to generate a fixed-size set of bounding boxes and scores related to the percentage of correlation between the object detected and the one requested, followed by an evaluation phase used to obtain the final results. This architecture makes possible to reach a high value of mAP (mean Average Precision §4.1.2) with a higher frame rate than R-CNN.

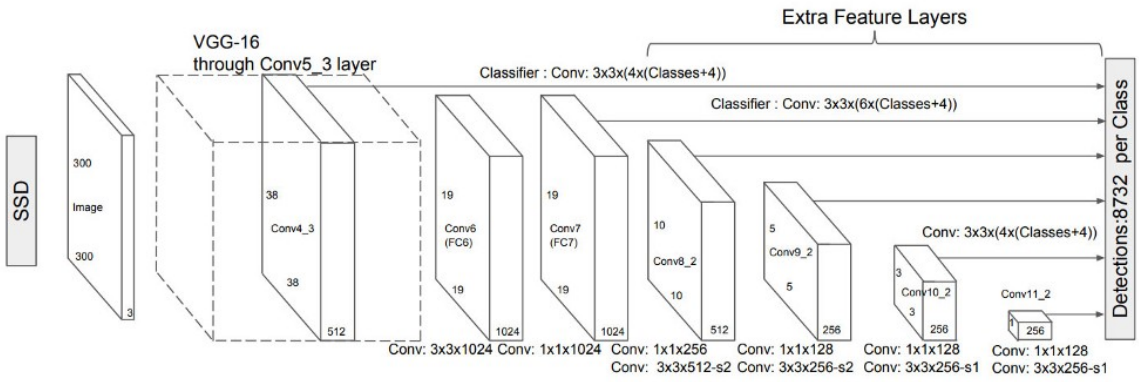


FIGURE 2.14: SSD structure [19].

The SSD architecture (figure 2.14) has as base network a VGG-16, which is chosen for its high performance on the classification of high quality images and the extraction of the feature maps. Differently from the original VGG, this one is not a fully connected layers, but it is a set of auxiliary convolutional layers used to extract features at multiple scales and step by step to curtail the size of the input image to the following layer.
Instead of using anchors like CNN, SSD uses *priors*, a pre-computed, fixed size bounding boxes which try to match the allocations of the original ground truth boxes. The *priors* are selected depending on the minimum Intersection over Union ratio (IoU §4.1.1) of 0.5. This is a surely better starting point instead of starting the prediction in random coordinates. Therefore, this technique starts with the *priors* as predictions and reaches a position close to the ground truth bounding boxes.

Chapter 3

Y.O.L.O.

You Only Look Once [20] is an object detection method used for real-time image processing applications. As it is an evolution of the SSD concept, it is able to predict bounding boxes and the class detection probability analysing the image just one time. Its architecture is based on a single neural network only that can be trained end-to-end to increase the accuracy, moreover Y.O.L.O. reduces the predictions of false positives on background.

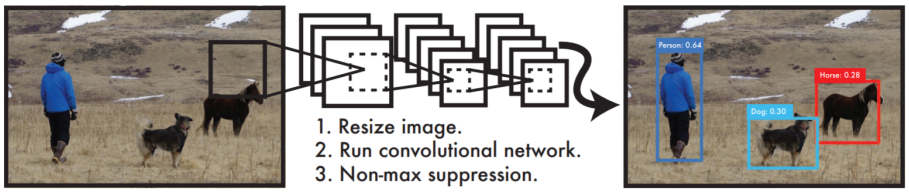


FIGURE 3.1: Y.O.L.O. detection passages [20].

3.1 How it works

The algorithm could be divided in four steps:

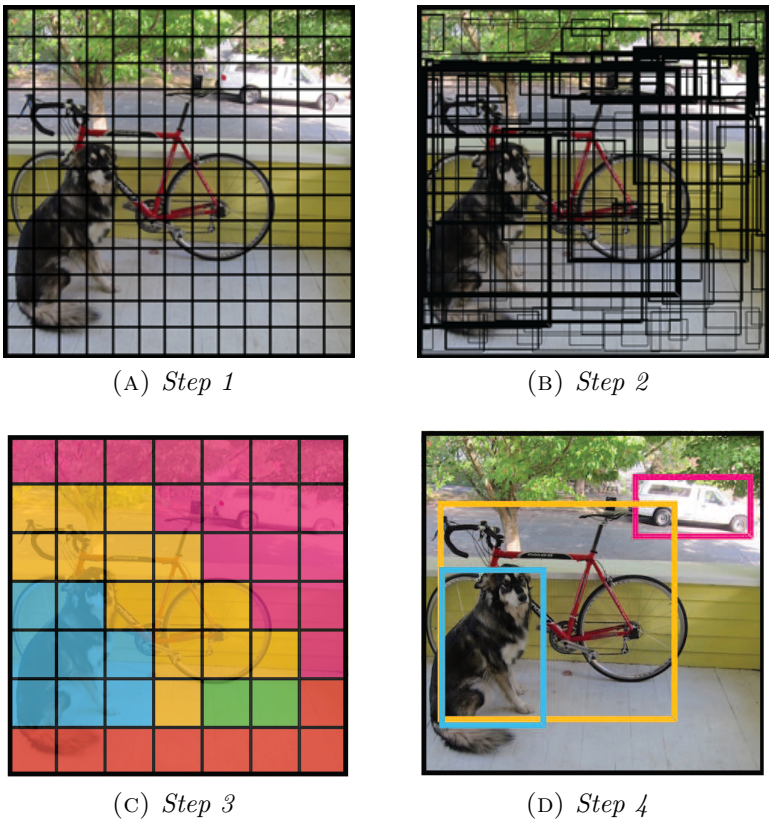
1. The input image is divided into a grid of S x S cells (figure 3.2a).

2. Each grid cell generates B bounding boxes and predicts their confidence rate. The confidence value depends on how the network is sure that inside a precise bounding box there would be an object. In the figure (3.2b) it is reported the step already explained underlying that the higher confidence of the model is indicated using a fatter drawn boxes. Each bounding boxes have 5 predictions: x, y, w, h and the confidence. The (x, y) position identify the centre of the box, w and h are respectively the weight and the height of the box.

3. Each grid cell has a C value representing the class probabilities. This value depends on the dataset used during the network training process (PASCAL VOC,

COCO, etc...). In the figure (3.2c) the class probability map for each cells is represented.

4. The total number of the bounding boxes, calculated as S x S x B must be minimized because many of them have a lower confidence rate. By setting a minimum value of this rate it is possible to reduce the bounding boxes to the only sure one and the final result obtained is shown in the figure (3.2d).



(A) *Step 1* (B) *Step 2*

(C) *Step 3* (D) *Step 4*

FIGURE 3.2: Y.O.L.O. procedure sequence [20].

3.2 Network Design details

The network architecture (figure 3.3) is composed by 24 convolutional layers followed by 2 fully connected layers. The alternation between 1 x 1 and 3 x 3 convolutional layers is useful to limit the feature space from the preceding layers. Differently, the fully connected layers are used to generate an output tensor with a desired shape, which is flattened and used to obtain (S , S, B x 5 + C) parameters.

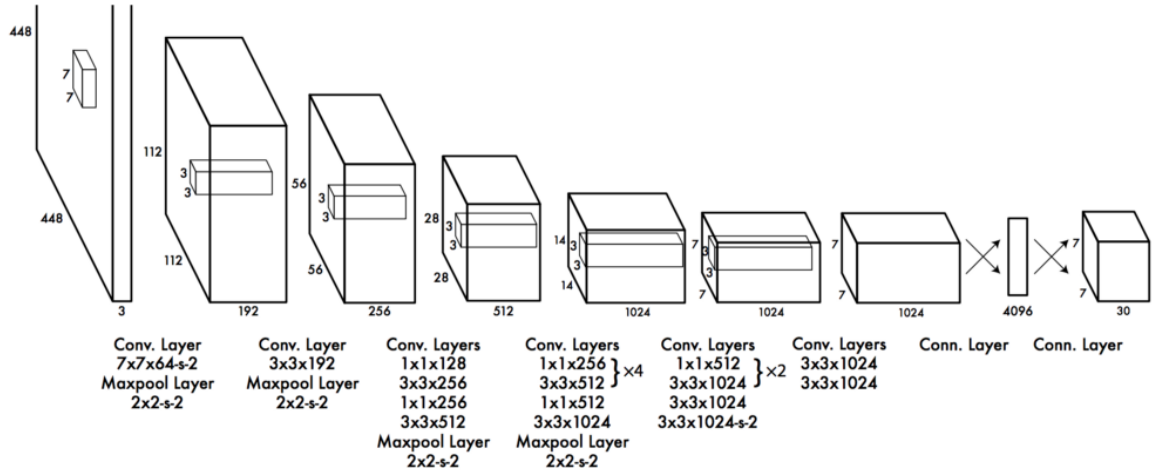


FIGURE 3.3: Structure of Y.O.L.O. [20].

Layer	Kernel	Stride	Output
Input			(416, 416, 3)
Convolution	3x3	1	(416, 416, 16)
MaxPooling	2x2	2	(208, 208, 16)
Convolution	3x3	1	(208, 208, 32)
MaxPooling	2x2	2	(104, 104, 32)
Convolution	3x3	1	(104, 104, 64)
MaxPooling	2x2	2	(52, 52, 64)
Convolution	3x3	1	(52, 52, 128)
MaxPooling	2x2	2	(26, 26, 128)
Convolution	3x3	1	(26, 26, 256)
MaxPooling	2x2	2	(13, 13, 256)
Convolution	3x3	1	(13, 13, 512)
MaxPooling	2x2	1	(13, 13, 512)
Convolution	3x3	1	(13, 13, 1024)
Convolution	3x3	1	(13, 13, 1024)
Convolution	1x1	1	(13, 13, 125)

TABLE 3.1: Structure of Y.O.L.O.

3.2.1 Loss Function

Y.O.L.O. can predict multiple bounding boxes for each grid cell, but, during the training, it is requested only one bounding box predictor for each object. This selection of the predictor depends on the highest IoU with the ground truth. In this way, the size and shape of the bounding boxes are easier to recognise and also it is possible to compute the *loss function* as the correlation of three factors:

- Localization loss;
- Confidence loss;
- Classification loss.

Localization Loss

These parameters calculate the errors of the predicted bounding box locations and size, but related only with the box that detected an object.

$$\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2]+\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}[(\sqrt{w_i}-\sqrt{\hat{w}_i})^2+(\sqrt{h_i}-\sqrt{\hat{h}_i})^2]$$

Where:

- $\mathbb{1}_{ij}^{obj}$ is equal to 1 if the j-th edges of the cell i is selected only once to recognise the object, differently is set equal to 0.
- λ_{coord} is used to improve the weight of the loss in the bounding box coordinates, so it normalizes the error among small and large boxes.

Confidence Loss

It is formed by two formulas and the use of one instead of another depends on the object being detected or not:

$$\begin{cases}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}(C_i-\hat{C}_i)^2 & \text{if detected}\\ \lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{noobj}(C_i-\hat{C}_i)^2 & \text{if not detected}\end{cases}$$

Where:

- If $\mathbb{1}_{ij}^{obj}$ is equal to 1 the j-th bounding box in i-cell is selected once to recognise the object, 0 ($\mathbb{1}_{ij}^{obj}$ is its complement) in the opposite case.
- $\hat{C}_i$ is the box confidence score of box j in the cell i and λ_{noobj} reduces the loss during the background detection.

Classification Loss

In case of detected object, this parameter is given for each cell as the squared error of the class conditional probabilities for each class.

$$\sum_{i=0}^{S^2} \mathbb{1}_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

Where:

- $\mathbb{1}_i^{obj}$ is equal to 1 if there is an object in the i cell, otherwise is equal to 0.
- $\hat{p}_i(c)$ represents the class conditional probability of the class c in cell i.

3.2.2 Inference

Y.O.L.O. is very fast during the test process because it needs only a single network evaluation. In the most of the cases the detection is accurate and each object is identified inside a bounding box. However, in some cases, when the objects in the image are close one to another or when the object is really large, the algorithm makes a doubles detection of the same object. This problem can be solved by applying the *non-maximal suppression*. This method removes inference deleting the bounding with the lower confidence and increases the mAP of the 2-3%.

3.2.3 Limits

Y.O.L.O. is a really strong object detection method. It is really simple to realize and can be trained directly on the entire images. However, it has some limitations:

- Spatial constraints limit the nearby objects prediction;
- Problem in the generalization of new objects or atypical aspect ratios;
- Same treatment of the error in small or big bounding boxes.

So the main problems of this architecture are the wrong localizations.

3.3 Y.O.L.O.v2

3.3.1 Accuracy Improvements

Y.O.L.O.v2 is a great improvement of the previous model version [21]. The object detection is more accurate and extremely fast, so the performances are higher. As it is shown in the following table many parameters are modified in order to obtain this result.

	YOLO							YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓
hi-re classifier?			✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓
anchor boxes?				✓	✓			
new network?					✓	✓	✓	✓
dimension priors?						✓	✓	✓
location prediction?						✓	✓	✓
pass-through?							✓	✓
multi-scale?								✓
hi-res detector?								✓
Pascal VOC (2007) - mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	78.6

TABLE 3.2: Improvements in Y.O.L.O.v2.

These parameters affect the increases in mAP.

Batch normalization

The batch normalization increases the regularization of a neural network. It "normalizes the output of the previous layer by subtracting the batch mean and dividing by the batch standard deviation." [22]. Generally, it is set between the convolutional layer and the activation function and it is able to remove the drop-out from the model and to guarantee an improvement of 2% in the mAP (figure 3.4).

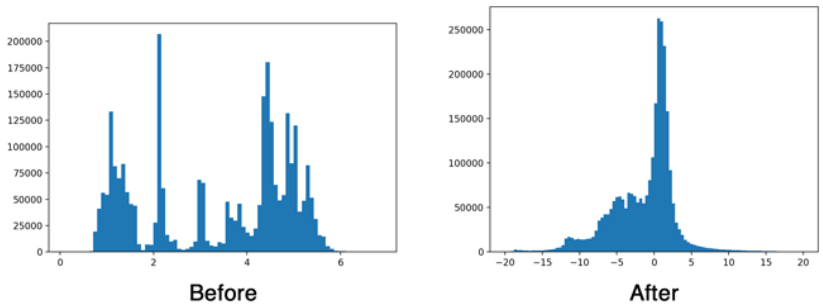


FIGURE 3.4: Example of batch normalization [23].

High-resolution classifier

The training of the original Y.O.L.O. starts with the classifier and then to the fully connected layers for object detection. In Y.O.L.O.v1 the classifier is trained with 224x224 images and the object detection part with 448x448 images resolution. In Y.O.L.O.v2 this process changes: there are two trainings on the classifier. The first one is done with 224x224 images and the second retrain is done with 448x448 images resolution and 10 epochs. This retrain makes the network accurate and improves the value of the mAP of about 4%.

Convolutional Layer with Anchor Boxes

Y.O.L.O. uses priors to predict the bounding boxes. The boxes are selected randomly and not always fit all the objects well, so the gradient is unstable during the step changes. By analysing other networks for object detection, it is noticed that replacing the offset prediction with the coordinate prediction makes the problem easier. The concept of offset prediction is the following: the priors that include a specific class are constrained between an offset range. In this way the diversity of the prediction does not change, but at the end of each epoch the collected shape information makes it accurate. According to this, in Y.O.L.O.v2 there are some changes:

- The fully connected layers are removed and substitute with anchor boxes;
- The predictions are done not at cells level, but at boundary box level;
- The image input size from 448x448 becomes of 416x416. This is done in order to:
  - Have odd number of locations in the feature map and to easily determine at what centre cell the object belongs;
  - Get an output feature map of 13x13 through a convolutional layers down-sampling of 32 ratios.

The anchor boxes method affects in the accuracy in fact, Y.O.L.O. predicts at least 98 boxes per image and Y.O.L.O.v2 can predict until a thousand ones. Without changing in the network, the model gets 69.5 mAP with a recall of 81%. In Y.O.L.O.v2 the use of anchor boxes reduces the mAP of a 0.3 factor, but the recall reaches 88%.

Dimension Clusters

The priors have similarities that are not too many in each dataset, so it is necessary to choose the most important one to be used. In the classical anchor boxes approach, the priors are find by hand and that increases the complexity of the network. In order to simplify the process, it is used a *K-means* clustering on the training dataset. With

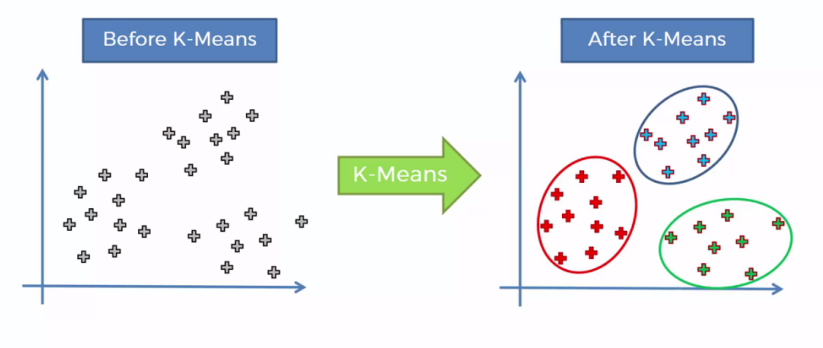


FIGURE 3.5: Priors found with K-means clustering [24].

this method it is possible to find the principal boundary boxes through the centroids (figure 3.5).
K is the number of clusters and has to be tuned depending on the used dataset and its complexity. The choice has to guarantee a good IoU scores independently from the size of the box. As it is possible to see in the figure 3.6 K=5 is a great trade-off between the complexity of the model and the high recall.

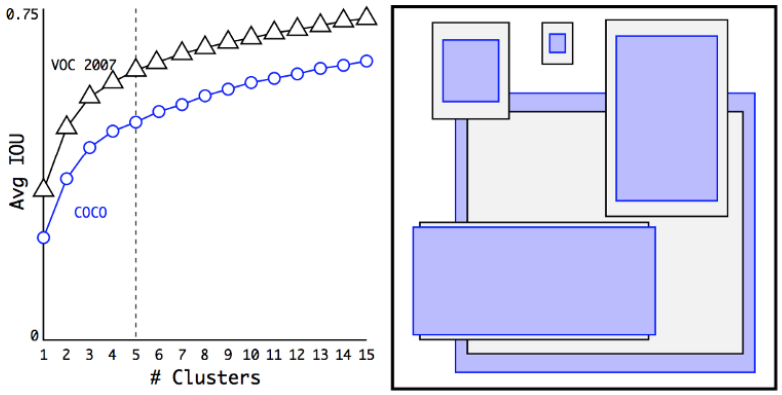


FIGURE 3.6: Clustering box dimensions on VOC and COCO. The right image shows the relative centroids found for VOC and COCO [21].

By analysing the difference between the handmade priors and the K-means method it is possible to notice an increasing in the IoU average of 0.2 and consequently a simplification of the network.

Direct Location Prediction

The anchor boxes used is responsible of the model instability during the initial iterations. The instability problem derives from the prediction of the (x, y) coordinate of the box. The classical offset prediction approach is extremely difficult to stabilize, so it is preferable to predict the coordinate related to the position of the grid cell. In this way the network's prediction must be in the 0 to 1 range from the ground truth. It

is possible to predict 5 bounding boxes for each cell and find their respectively values through the following formulas:

$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x \\
b_y &= \sigma(t_y) + c_y \\
b_w &= p_w e^{t_w} \\
b_h &= p_h e^{t_h}
\end{aligned}
$$

$$
P_r(object) * IoU(b, object) = \sigma(t_o)
$$

From these equations are obtained (figure 3.7):

- t_x, t_y, t_w, t_h: the coordinate of each bounding boxes;
- (c_x, c_y): normalized by the size of the image, is the origin point set in the top left of the prior;
- b_x, b_y, b_w, b_h: the predicted boundary box;
- $\sigma(t_o)$: the confidence score of the box;
- c_w, c_h: , normalized by the size of the image, are the width and the height of the prior.

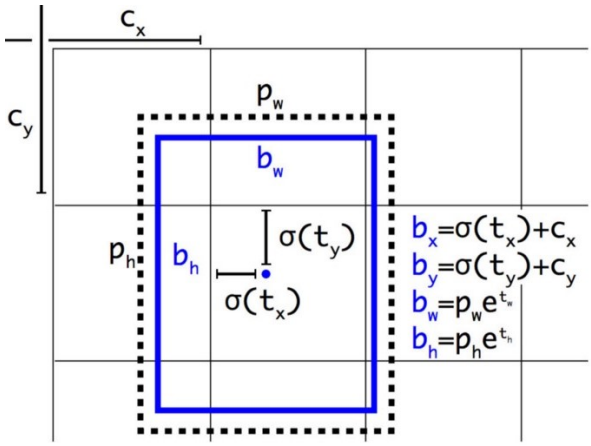


FIGURE 3.7: Visualization of Y.O.L.O.v2 boxes algorithm. The dotted box is the prior and the blue box is the predicted boundary [21].

This prediction method further simplifies the network and increases the performance of 5%.

Fine-Grained Features

Y.O.L.O. does a prediction with a 13x13 feature map. It is sufficient and works correctly with the medium size objects, but it can have some problems in case of small object

recognition. Both SSD and Faster R-CNN run the proposal networks over many feature maps obtaining object of different dimension, using a range of resolutions. In order to do that in the Y.O.L.O. network, as it passes through the image only once, it is necessary to add a pass-through layer to modify the resolution of the feature map from 13x13 to 26x26. The detector is set on the top of these features expansion and the result is an increase of 1% of the performance.

Multi-Scale Training

Initially Y.O.L.O. receives input size images of resolution 448x448, but the use of anchor boxes modifies this value into 416x416. However, considering that the new architecture is composed by convolutional and pooling layers, the size can be changed simply on the fly. This reduce the complexity of the network and Y.O.L.O.v2 results more robust to run on images of various dimensions. In this way the input image size is not fixed, but every 10 batches the network chooses a new image size randomly. The model down-samples by a 32 factor, so it can be trained with images of different sizes from 320x320 to 608x608 (320, 352, .., 576, 608). The use of different resolution images during the training of the network increases the quality of the predictions. Now the system can be used for lower resolution applications with an increase in the mAP, which becomes next to the Fast R-CNN with more than 90FPS. At high resolutions Y.O.L.O.v2 reaches 78.6mAP with PascalVoc.

Accuracy and Speed Comparison

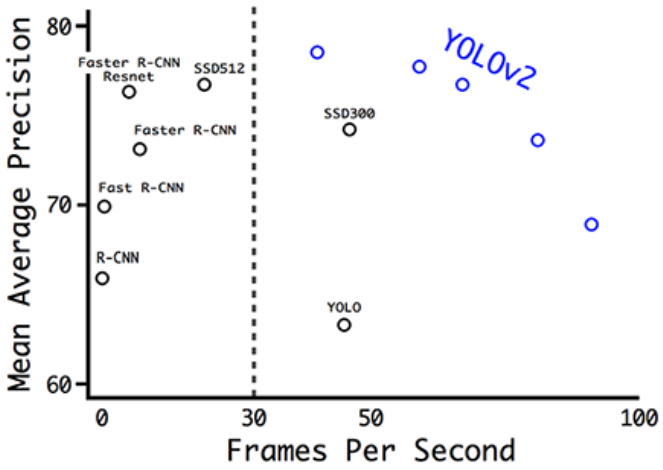
Y.O.L.O.v2 is faster and more accurate than the previous object detection algorithms (figure 3.8) because it is trained to run at different resolutions.

3.3.2 Speed

A speed improvement is requested in Y.O.L.O.v2 to maximize the performance of the algorithm. Most of the networks use as feature extractor VGG-16, which is powerful, but extremely complex: it requires 30.69 billion of floating point operations to analyse an image with 224x224 resolution.

A solution can be replacing this architecture with *GoogLeNet*, which guarantees an important improvement in the processing time although the accuracy decreases by 2%.

Another improvement is obtained changing the classification model and using Darknet-19. This new classifier uses 3x3 filters and doubles the number of channels after every pooling step. Moreover, it uses 1x1 filters to compress the feature map representation between 3x3 convolutions and also global average pooling to make predictions. Darknet-19 processes an image using only 5.58 billion operations and reaches 72.9% of



Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	**78.6**	40

FIGURE 3.8: Accuracy comparison for different detectors [21].

top-1 accuracy and 91.2% of top-5 accuracy on *ImageNet*.

The network is trained on ImageNet with a dataset composed by 1000 classes for 160 epochs and considering momentum of 0.9, weight decay of 0.0005, stochastic gradient descendent with starting learning rate of 0.1 and a polynomial rate decay with a power of 4.

Y.O.L.O.v2 architecture is showed in the following table:

Layer	Filters	Stride	Output
Convolution	32	3x3	224x224
MaxPooling		2x2/2	112x112
Convolution	64	3x3	112x112
MaxPooling		2x2/2	56x56
Convolution	128	3x3	56x56
Convolution	64	1x1	56x56
Convolution	128	3x3	56x56
MaxPooling		2x2/2	28x28
Convolution	256	3x3	28x28
Convolution	128	1x1	28x28
Convolution	256	3x3	28x28
MaxPooling		2x2/2	14x14
Convolution	512	3x3	14x14
Convolution	256	1x1	14x14
Convolution	512	3x3	14x14
Convolution	256	1x1	14x14
Convolution	512	3x3	14x14
MaxPooling		2x2/2	7x7
Convolution	1024	3x3	7x7
Convolution	512	1x1	7x7
Convolution	1024	3x3	7x7
Convolution	512	1x1	7x7
Convolution	1024	3x3	7x7

TABLE 3.3: Structure of Y.O.L.O.v2.

3.3.3 Hierarchical Classification

The datasets used in object detection have less class categories than those used for classification. In Y.O.L.O.v2 the images are mixed from both detection and classification datasets during the training. The whole network is trained once using object detection samples, instead, the classification loss is back-propagated from the classification path. However, the main problem is to find a way to merge different datasets in order to avoid mutual exclusion classes. The solution is the use of a hierarchical classification (figure 3.9).

This type of structure links an image detected as “cat” from COCO to “Siamese cat” recognised from ImageNet. According to that it is generated a WordTree structure with parent/children used also for the final classification result. Generally, the children

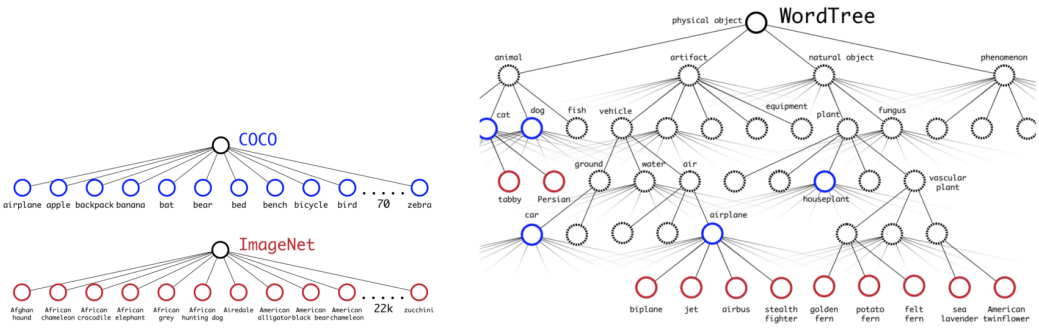


FIGURE 3.9: Structure of COCO, ImageNet and WordTree dataset [21].

confidence is law, so the output confidence of the all system is related to their parent (in this case of the "cat").

Joint classification detection

Using the Y.O.L.O.v2 structure with only 3 priors it comes up Y.O.L.O.9000.
Y.O.L.O.9000 leads the WordTree to the limits linking the COCO detection dataset to the top 9000 classes from the *ImageNet* release. In conclusion joint classification detection training makes the network able to detect the object in the images using the detection data of the COCO dataset and to classify a large quantity of these objects with *ImageNet*.

3.4 Y.O.L.O.v3

Y.O.L.O.v3 [25] is the last evolution of this network. It introduces a notable modification in the class prediction. Until now the classifiers have a mutually exclusive output labels, so the final softmax function has maximum result equal to 1. Y.O.L.O.v3 modifies this structure and becomes a multi-label classifier, so an output result could be labelled as both "car" and "vehicle". The softmax function is not limited anymore to 1, so it is replaced with an independent logistic regression classifier which calculates the correct belonging of the input to a specific label. In this way, the mean square error of the classification loss is eliminated and replaced with the cross-entropy loss for each label and the cost function is modified. An objectness score equal to 1 is linked to the prior bounding box that overlaps a ground truth object. For the priors that satisfy the predefined threshold of IoU, the cost function is equal to 0. In this way, only a ground truth object is linked to a prior boundary box and, if more bounding boxes satisfy the requirements, there is no classification.

3.4.1 Prediction

Y.O.L.O.v3 makes 3 predictions for locations including: bounding boxes, 80 class scores and objectness. A concept similar to Feature Pyramid Networks (FPN) is used to extract the features and it is explained in the following steps:

1. Realizing a prediction in the last feature map layer;
2. Taking into account the 2 layers back and up-samples this layer by 2;
3. Taking a feature map from the earlier and merging it with the up-sampled feature map in order to have more detailed information of both the resampling features and finer-grained;
4. Adding a convolutional filter on the merged map to obtain the second prediction;
5. Repeating the process from the second step in order to obtain the final prediction. In this way the last prediction will take into account of the previous calculations and of the fine-grained features of all the network.

The priors of Y.O.L.O.v3 are found using K-Means clustering on the COCO dataset. The preselected clusters are 9: (10x13); (16x30); (33x23); (30x61); (62x45); (59x119); (116x90); (156x198); (373x326). These ones are divided into groups and then assigned to a specific map, so the object can be detected better.

3.4.2 Feature Extraction

The network used for the feature extraction is changed from the Darknet-19 of Y.O.L.O.v2 to Darknet-53 of Y.O.L.O.v3, so there is an important increase of layers (figure 3.10): now it has 53 convolutional layers.

It better uses the GPU because it reaches the best, highest value of floating point operations per second, so it is more powerful than the Darknet-19 and still more efficient than ResNet-101 and ResNet-152.

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

FIGURE 3.10: Darknet-53 network structure [25].

3.4.3 Results

As it is possible to see in the figure 3.11 Y.O.L.O.v3 is the fastest and performant network tested on COCO dataset. The multi-scale predictions increase the APs performance of the network and the small object detection is not a problem anymore. Differently there is a little worsening in the detection of medium and large dimension objects.

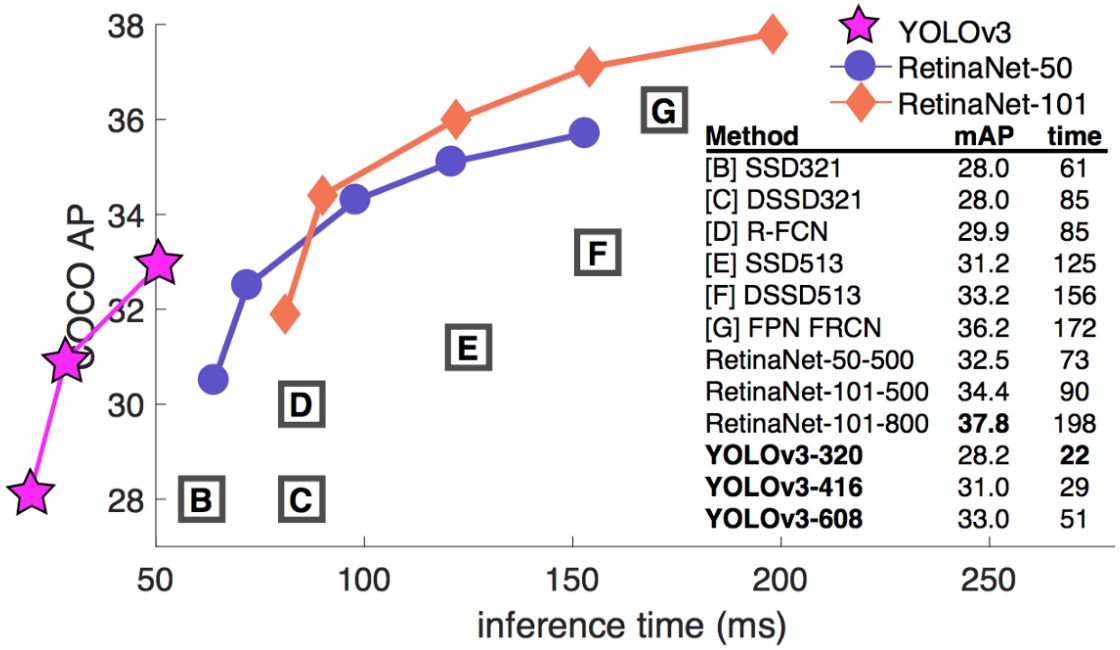


FIGURE 3.11: Performance comparison [25].

It is strongly used for the real-time operation with great results. There are many types of this network depending on the complexity requested by the algorithm (as for example Y.O.L.O.v3, Y.O.L.O.v3-tiny, Y.O.L.O.v3-spp) and on the resolution of the images adopted (320, 416, 608).

Chapter 4

Metrics

The machine learning researchers can be divided in two factions, supporting two different ideas. The first one believes in the uselessness of knowing how exactly a neural network works, differently the second one says that it is important to know everything about it: parameters, steps, all the network structure to have the control of each passage.
However, it is extremely complex to understand how a neural network works, moreover the one used for deep learning. Generally, the global functionality and the general passages are clear, but it is difficult to go into details of the particular values used.
The thing that the researchers have in common is the necessity to evaluate the results and to understand if and how a network is performing. To do that many methods are used to evaluate the networks. In the following part of the chapter the ones used in this thesis work are listed in order to evaluate the performance of the used algorithms.

4.1 Metrics used for Object Detection

The methods used to evaluate the object detection analysis are different from the general ones because in this case the evaluation is based on pixels and not on boolean data. There are two main methods: IoU and Mean Average Precision.

4.1.1 IoU

Intersection over Union [26] is the evaluation metric used to assess the accuracy of an object detection neural network on a particular test dataset. It can be used easily with all the algorithms that give bounding boxes as output and it needs only two elements per image to evaluate:

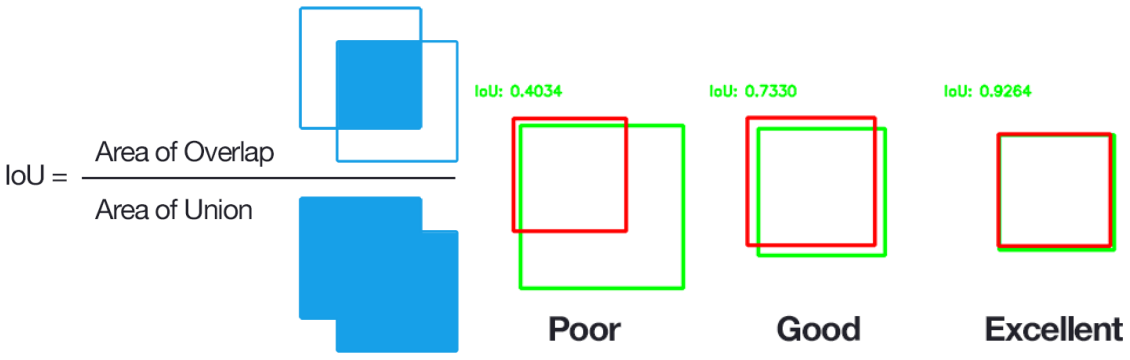
- *Ground-truth bounding box*: the minimum box size that can contain the detected object. It is hand labelled from the programmer;
- *Predicted bounding box*: the output of the model.

IoU measures the overlapping between these two regions (figure 4.1) with a simple mathematical operation:

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union}$$

and its value is a number between 0 and 1. In case of perfect overlap it is equal to 1, differently if the two boxes do not overlap at all the IoU is 0 (figure 4.2). Generally an acceptable IoU is considered over 0.5.

FIGURE 4.1: IoU application on a person. The red box is the predicted, differently the blue one is the ground-truth [27].



(A) *IoU calculus.* (B) *Examples of IoU results.*

FIGURE 4.2: IoU computation and evaluation [26].

Ground-truth

The real problem of the IoU computation is related to the ground-truth bounding boxes because each algorithm requires a specific ground-truth format file. There are some programs that are able to give also different outputs for the selected dataset, so

it is possible to choose the one compatible with the ground-truth request. However, not all the networks can give a specific output format and it is necessary to create autonomously the ground truth files starting from the predicted output files of the network. This is why, before, it has been said that the ground truths are hand labelled.

4.1.2 Mean Average Precision

The *mean Average Precision* evaluates the general performance of the network, so it is not referred to a single prediction, as for IoU computation. In order to understand how it is computed it is necessary to firstly recap the definition of Precision and Recall factors.

Precision & Recall

The *Confusion Matrix* is a table used to evaluate the correctness of an object detection network.

		Predicted	
		P	N
Real	P	True Positive (TP)	False Negatives (FN)
	N	False Positive (FP)	True Negatives (TN)

Each *Predicted* result of a network enters into the confusion matrix and is compared to the *Real* one, obtaining as output one of the following four combinations:

- TP = True positive (positive predicted and positive real);
- TN = True negative (negative predicted and negative real);
- FP = False positive (positive predicted and negative real);
- FN = False negative (negative predicted and positive real).

By doing this operation for all the predicted values of the network we obtain the global values of the TP, TN, FP and FN, so they can be used to compute the evaluation criteria as for example the precision factor and the recall factor.

- *Precision* (True Positive Rate): it evaluates the correctness or the prediction

$$\frac{TP}{TP + FP}$$

- *Recall* (Sensitivity): is the ratio between the predicted positives and the possible positives

$$\frac{TP}{TP + FN}$$

These factors depend on the pre imposed threshold as the IoU detection (generally IoU>0.5) because it determines the results of the detections. In fact, if a box has a IoU under the threshold it is TN, differently if its IoU is over the threshold it is TP. This affects the performance of a classifier, which can be evaluate considering how the threshold changing affects the precision and recall factors.
In case of a good classifier the precision will remain high when the recall increases. If the classifier is weak a loss of precision is required to have a high recall.
This behaviour can be shown using the Precision-Recall curve (figure 4.3).

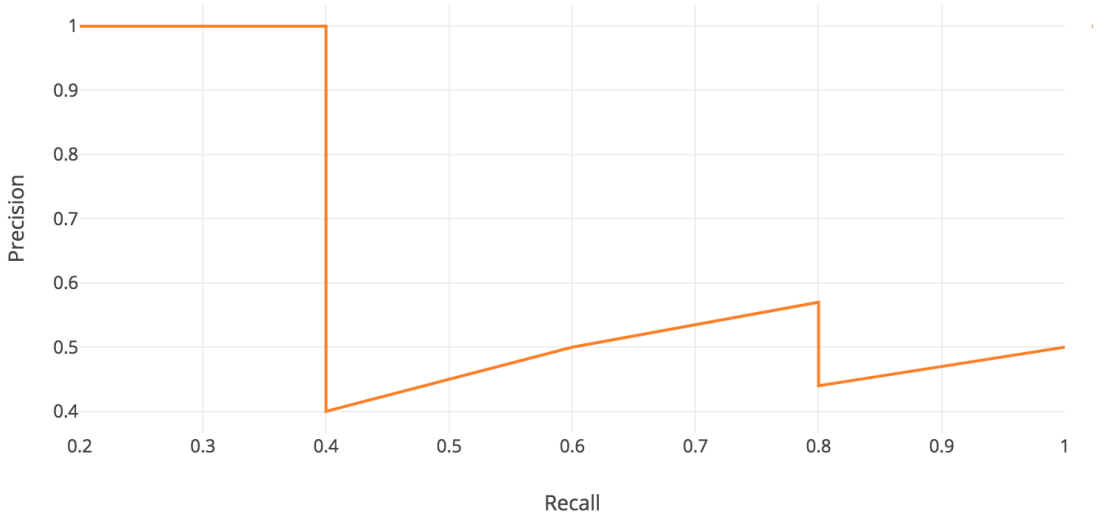


FIGURE 4.3: Example of a Precision-Recall graph [28].

AP

The Average Precision is an important value used to make the comparison of the results graphs of different algorithms possible.
The rigorous definition of the AP said that it is the area under the Precision-Recall curve. However, this can give strange results with few variations, so the solution is to use the interpolated precision technique. This method is very common and usually it is used to indicate the AP. In this case it is not considered exactly the area under the curve, but an approximation of it. In the figure 4.4 it is shown the area considered (the one under the green line) using the interpolation method.

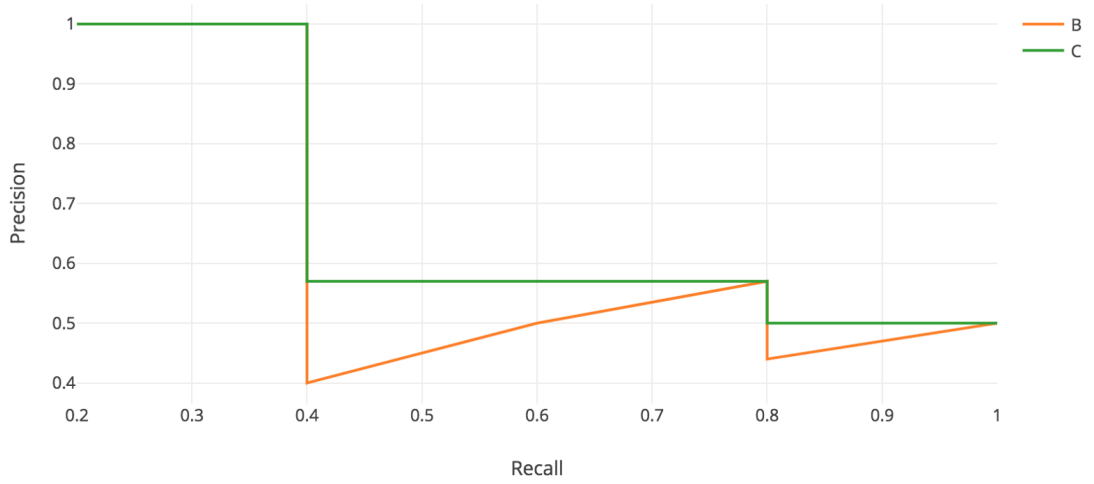
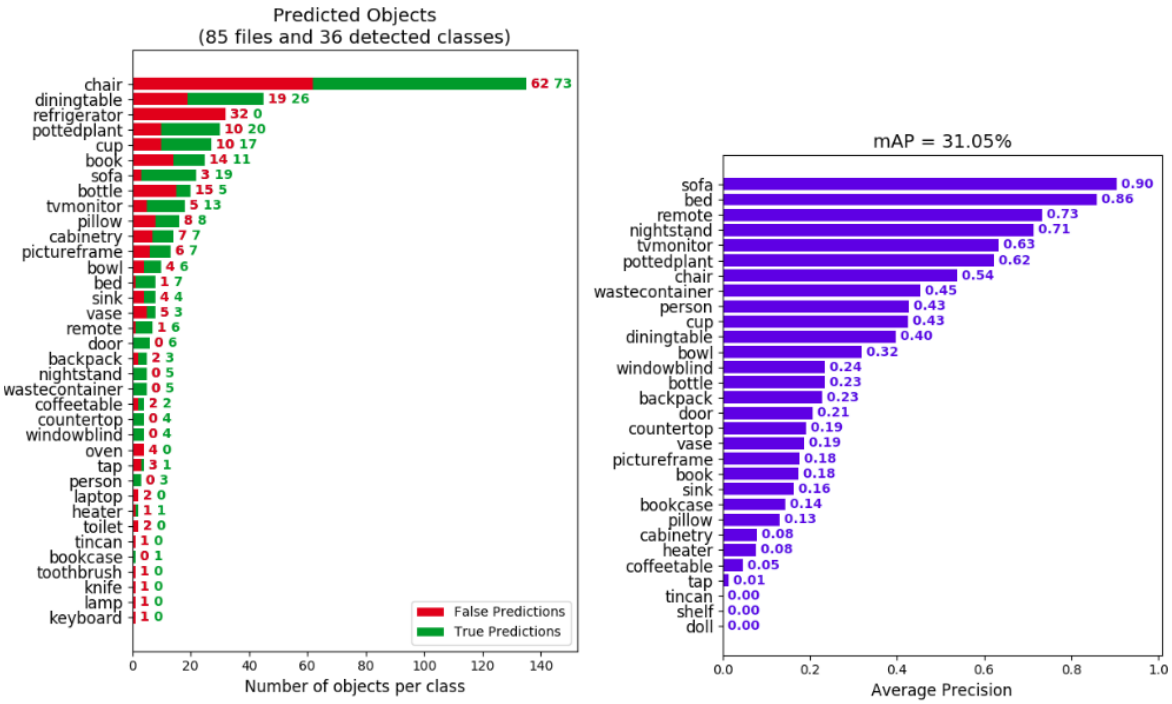


FIGURE 4.4: Example of a Precision-Recall approximated graph with interpolation technique [28].

mAP

It is possible to evaluate all the classes of the network using the AP of each one. The mean of all the classes considered is called mAP (figure 4.5). This value depends on the IoU predefined value.



(A) False and True predictions of the whole detected classes.

(B) mAP and the AP related to each class.

FIGURE 4.5: AP and mAP example results [29].

In this work thesis the mAP is used to evaluate the results of the networks chosen. This

value is calculated both with IoU=0.5 (that is the same of AP) and with IoU=0.75 for the networks pre-trained with COCO dataset and these values are compared with the same ones obtained after the retraining of these networks just for the class person.

Chapter 5

Robot

5.1 Introduction

A Robot is a programmable machine, autonomous or semi-autonomous, capable of performing a series of tasks alongside or in the place of a man. It can perform exclusively mechanical and repetitive tasks or adapt its behaviour to the surrounding environment, learning from experience. A robot is composed by a hardware part and a software part. Everything related to motors, circuits, mechanism and sensors is part of the hardware. Differently, the components of the software are: microcontroller firmware and map building software. Dependently on which part we have to operate in, there are different platforms. The Robot software platform contains many tools used in robot application programs. The hardware platforms are not only study, but also commercial platforms. The most important software platform used to control the robot is ROS, that is the acronym of Robot Operating System. It is an open source framework used to control robots' tasks, motions and other operations. It is very simple to use, so it is perfect not only for people that daily work with robots, but also for the beginners.
The most important robot packages are two:

- PR2 for mobile-based humanoid robot. It is really performant, but quite expensive;

- TurtleBot is the most used such that increases the market of ROS.

The TurtleBot package exists in three versions. The first one is followed by TurtleBot2, where the mobile platform adopted is KOBUKI, and then there is TurtleBot3 developed in collaboration between Open Robotics and ROBOTIS.
There are other many robots used for different applications, the most important are set in the following list:

- Industrial robots: similar to a human arm with wrists and an arm;

- Humanoid robots: similar to a human body with a torso, two legs, two arms and 2-5 fingered hands;
- Biomimetic robots: similar to animals;
- Service robots: they have different structures, but the most popular is wheeled;
- UAVs: *Unmanned Aerial* (or Autonomous) Vehicles, the most common are the quadcopters;
- Exploration robots: used during planetary or deep space exploration.

5.2 Sensors

Sensors are devices able to give a miserable response to a change in physical quantity related to the robot itself or to the external environment. Generally, sensors convert the physical quantity into a signal that can be measured electrically. The sensors are divided in classes using some criteria:

- Applications;
- Primary Input quantity;
- Transduction principles;
- Used technology and material;
- Various measured property.

All the information collected from the sensors is used by the robot in order to perform actions or other operations as for example generating a map or interacting with objects or humans.
The most used sensors are the ones related to the mobile robots that collect information from the environment, so they could have some problem related to the dynamic changing of the environment or they could be affected by the noise.

The most common sensors are: encoders, resolvers, inertial sensors, inclinometers, gyroscopes, IMUs (Inertial Measurement Units), accelerometers, MEMs, beacons, GPS, distance sensors such as LDS (Laser Distance Sensor), LiDAR (Light Detection and Ranging) or LRF (Laser Range Finders), ultrasonic sensors, laser sensors, IR sensors, 3D sensors, vision sensors, inertial sensors, microphones and torque sensors.
Depending on the type of sensor used for a specific topic, the robot receives different data information with a specified frequency that has to be inside the limit of the microprocessor. For example, 1D or 2D sensors do not send heavy data, differently, from the camera that needs a high processing power microprocessor able to collect and analyse a lot of data.

There are many ROS packages related to the sensors, which are classified in 1D range finders (as Infrared distance sensor), 2D range finders (as LDS generally used in navigation), 3D sensors (as Intel RealSense), pose estimation (GPS+IMU), cameras, audio/speech etc...

5.2.1 Cameras

The vision is the most important human sense and it is also very important in robotics because it is not expensive and gives many data information of different type. The images taken from the camera sensors are used to recognise the objects and all the environment around the robot.
There are many applications [30] both in 2D (as for example object detection, colour recognition) and 3D (as for example distance evaluation) even if the 3D collected data generate a more robust system. In fact, as many 2D information of the environment could rapidly change, the 3D data are invariant, so many algorithms are interested in 3D.

5.2.2 Depth Camera

As it is anticipated many algorithms work better with 3D data and that increases the big progress in low-cost depth cameras. These are active devices.
The ToF (Time of Flight) cameras is one application of the depth camera in which a transmitter emits a modulated light. This light is reflected by the objects and it is received by a sensor, in this way it is possible to measure the distance from each pixel. This is an efficient algorithm; however, it is too expensive because it needs the use of a complex hardware.
Other depth cameras are based on the method of the structured light. This consists in illuminating the object with structured light and using 2D image sensor in order to detect the reflected pattern. In this way the depth camera is composed by an infrared projector and an infrared camera, which uses a coherent radiation pattern, differently from the ToF. This method is cheaper than the previous analysed, so it is generally used for low-cost robot.
The last depth camera method is the stereo-camera. It is based on the work of left and right eyes of the people. The stereo-camera is composed by two image sensors for capturing the image. The distance between the two lens is important to calculate the grid value using the difference between the two images. To compute the distance value between the camera and the object the stereo-camera uses the triangulation method. It consists in an infrared projector which emits IR with a coherent method and two infrared image sensors have to collect the receiving infrared rays to recreate an image.

5.2.3 Laser Distance Sensor

Laser Distance Sensor (LDS) is a series of different sensors as for example: Light Detection and Ranging (LiDAR), Laser Scanner and Laser Range Finder (LRF).
LDS measures the distance from the reflecting obstacles on a plane and gives high performance with real-time data acquisition that is why it is commonly used in robotic applications to find the distance between the robot and a person or an object in the environment.
LDS is constructed by a laser source, a motor and a reflective mirror. The motor rotates the mirror around an axis while it is scanning by using the laser. The typical range of a LDS goes from 180° to 360°. The object is scanned in the horizontal plane, so the objects are better detected and the accuracy is inversely proportional to the increasing distance.
There are some problems in the use of the LDS. The first is that not all the objects of the environment are able to reflect the laser correctly. In fact, there are some objects made in plastic or transparent glass that spread the laser in other directions generating a wrong or not accurate distance measurement. The second problem is that the acquisition of the data is done only in 2D, so it is a sort of limit of the sensor. The last is about the possible damage on eye risk. The lasers used from the LDS are classified with a number between 1 and 4, small numbers correspond to minor damage. The data collected with these sensors can be used in order to create a map of obstacles around the robot and to understand the pose of the robot inside it. One of the main program used to elaborate the LDS data is SLAM (Simultaneous Localization and Mapping).

5.2.4 ROS Motor Controller Drivers

Dynamixel is an high-performance actuator designed for robots, that has been created by the ROBOTIS company [31].
It has many features:

- Versatile because it has a vast range of applications;
- All of its parts are integrated in one module;
- Easily to be modified because it is reconfigurable and modular so many dynamixels can be chained together;
- Some of its registers contain information about its internal and external conditions;
- Stable and too efficient.

There are three different versions of Dynamixel as it is shown in the figure 5.1.
Dynamixel is used in robotics applications principally because it can give information about the torque control, the speed and the position.

FIGURE 5.1: Types of dynamixel [31].

5.3 Embedded System

An embedded system is a microprocessor or microcontroller based system that is realized to execute some determined tasks. It has three components: a hardware part, a software platform and a real time operating system.
Generally, many embedded systems are necessary to implement the tasks of the robots and to control real-time the actuators and the sensors of the robots.
In the TurtleBot3 there is as microcontroller an ARM Cortex-M7 to control the sensors and the actuator.

5.3.1 OpenCR

OpenCR (Open-source Control Module) is the embedded board used to control the operation of the TurtleBot3 robots and is developed to be compatible with ROS. The MCU supported by OpenCR is the STM32F7. That is a powerful microcontroller able to elaborate many data also with floating point computation.
There are many peripherals used to connect and control many devices as for example it makes possible the communication with Dynamixel of the Robot or sensors as LiDAR or camera.
The chip fixed in the middle of the OpenCR board integrates in one chip triple-axis accelerometer, gyroscope and magnetometer sensor, so has different uses (figure 5.2).

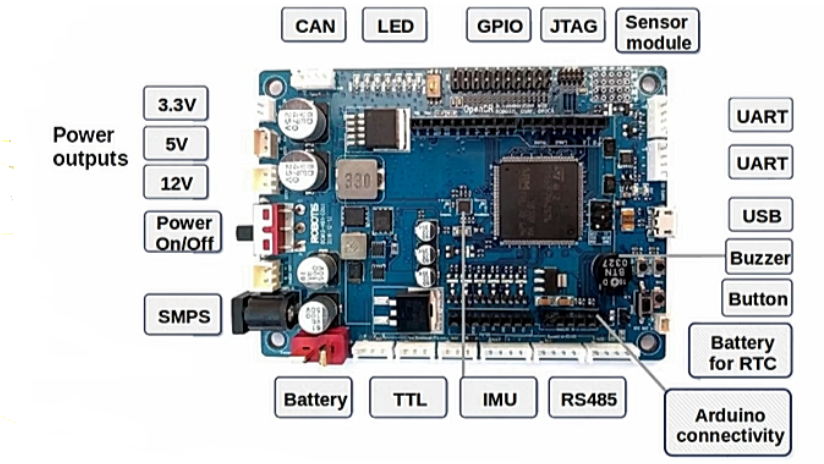


FIGURE 5.2: OpenCR interface configuration [32].

5.4 TurtleBot

TurtleBot is a standard platform of ROS and stands for the homonym robot used for many applications by students, developers and also beginners, in fact it is easy to understand. The name and the logo are inspired on the turtle animal.
Actually the used version is the third: TurtleBot3, which try to solve the problems or lacks of the previous versions and also to satisfy the user's requests. It uses the actuator seen in the previous chapter: Dynamixel for driving motions. Three are the robots of the third version: TurtleBot3 Burger, Waffle and Waffle Pi.
TurtleBot3 is a small programmable robot easy to modify, in fact, it is possible to realize different configurations of the robots adding or eliminating some sensors or changing the mechanical part structure (figure 5.3). Its low-cost price does not affect quality and performance.

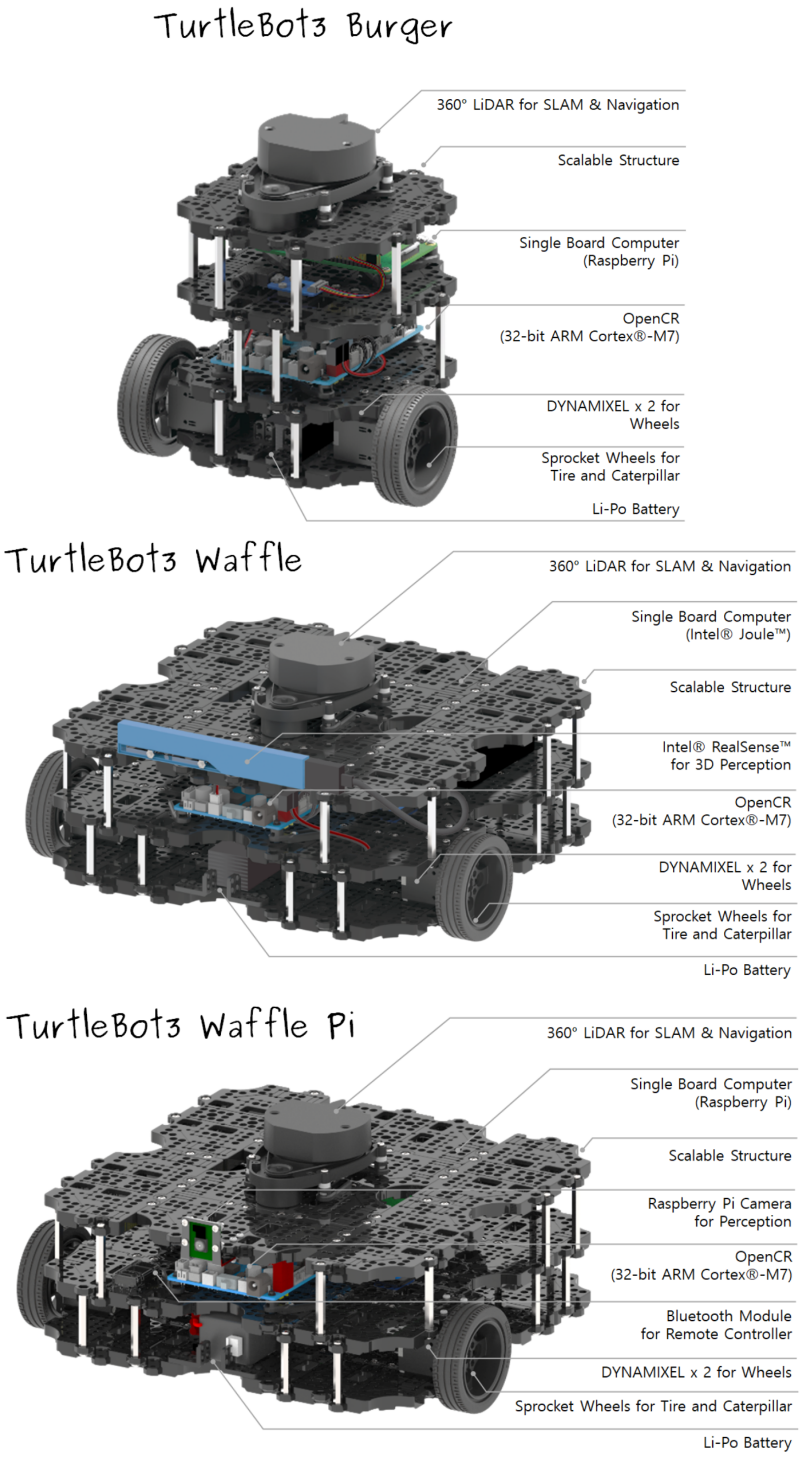


FIGURE 5.3: TurtleBot3 Hardware components [33].

5.5 Hardware components used

The problem of this thesis comes from the necessity of a real-time algorithm that is able to do person detection and elaborate these data to move the robot and doing person tracking. Of course all of these operations must be as instantaneous as possible because the risk is to lose the person to follow in case the robot moves away from the person's view.
A set of possible hardware has been analysed in order to find the best solution for this specific application. In the following paragraph the various combinations adopted depending on the different type of algorithm are explained and the reason why each of them is usable or it could be discarded.

5.5.1 Object detection hardware

In order to run the object detection algorithms more configurations have been tried:

1. Intel® RealSense™ Depth Camera D435i and the first configuration of Intel® Compute Stick (A+B1);
2. Intel® RealSense™ Depth Camera D435i and the second configuration of Intel® Compute Stick (A+B2);
3. Intel® RealSense™ Depth Camera D435i and NVIDIA® Jetson™ (A+C).

A) Intel® RealSense™ Depth Camera D435i

FIGURE 5.4: Intel® RealSense™ Depth Camera D435i [34].

It is used replacing the Intel® RealSense™ Camera R200 given inside the TurtleBot set. It is a stereo-camera able to guarantee high quality depth for many applications [34].
A vision processor and an Intel module are put united into a small form factor yields to guarantee an optimal device, which can be use both by developers and for production. In fact, it is a low-cost camera able to interact with the environment. As it belongs

to the D400 series of cameras, it can be integrated easily and has as support platform Intel RealSense SDK 2.0.
It is ideal for robotic applications such as navigation or object recognition.
The *i* at the end of the camera serial name stands for the presence of IMU in the camera, which puts together many sensors with gyroscopes to detect both movements and rotations in 6DoF.

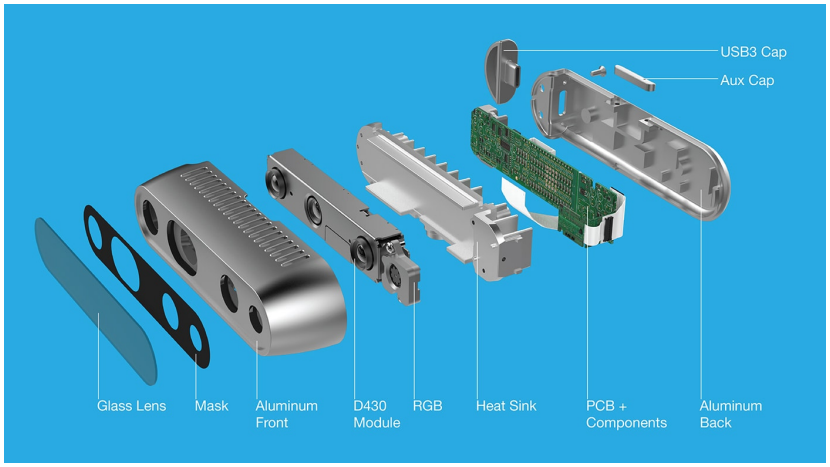


FIGURE 5.5: Internal hardware components of Intel® RealSense™ Depth Camera D435i [34].

With a D4 processor the D435i camera is composed by a RGB module, a left and a right infrared cameras separated by a wide IR projector, so it is possible to use four different camera types:

- RGB with resolution up to 1920x1080;
- Infrared 1;
- Infrared 2;
- Depth (made using the two infrared cameras as stereo-camera) with resolution up to 1280x720, 90° of diagonal FOV and reaches at most 90 FPS.

The images from the stereo-camera are already rectified from the camera, so it is not necessary the use of algorithm for the rectification.
The range of application of this camera depends on the lighting condition, but generally is from 0.105 [m] to 10 [m].
In the figure 5.6 it is shown the main important technical specifics of this camera, for any other detailed information see the complete datasheet [35].
In order to not lose information from the streaming of the D435i camera it is necessary to connect it to other devices using a USB 3.1 Gen 1, this has been one of the problems to be solved in this thesis. In fact, to do person tracking in the indoor environment it has been chosen a TurtleBot3, but it has as single board computer: a *Raspberry Pi*,

Features	Use Environment: Indoor/Outdoor Image Sensor Technology: Global Shutter, 3µm x 3µm pixel size	Maximum Range: Approx. 10 meters. Accuracy varies depending on calibration, scene, and lighting condition.
Depth	Depth Technology: Active IR Stereo Depth Field of View (FOV): 87°±3° x 58°±1° x 95°±3°	Minimum Depth Distance (Min-Z): 0.105 m Depth Output Resolution & Frame Rate: Up to 1280 x 720 active stereo depth resolution. Up to 90 fps.
RGB	RGB Sensor Resolution & Frame Rate): 1920 x 1080 RGB Frame Rate: 30 fps	RGB Sensor FOV (H x V x D): 69.4° x 42.5° x 77° (+/- 3°)
Major Components	Camera Module: Intel RealSense Module D430 + RGB Camera	Vision Processor Board: Intel RealSense Vision Processor D4
Physical	Form Factor: Camera Peripheral Length x Depth x Height: 90 mm x 25 mm x 25 mm	Connectors: USB-C* 3.1 Gen 1* Mounting Mechanism: One 1/4-20 UNC thread mounting point. Two M3 thread mounting points.

FIGURE 5.6: Technical specifications of Intel® RealSense™ Depth Camera D435i [34].

which does not support this type of USB, so it has been necessary to find a solution using a different hardware. Three solutions have been taken in consideration and here they are showed detailed including the reason that led to the choice of one of them only.

B) Intel® Compute Stick

It is a small device able to convert the HDMI desktop into a complete computer. It is wireless connected. Its RAM memory reaches 4GB and is as an integrate storage of 64GB.
The small dimensions are optimal for its use, in fact, it is easily integrated in the TurtleBot3 structure. Moreover, it has a USB 3.0 port, so it is possible to connect the camera to it and to process the image data without losing information.
The Intel Compute Stick (figure 5.7) has been used as an image processing computer in the TurtleBot3 supported by wireless connection of a Remote PC in two different configurations in order to find which has been the best solution to use:

FIGURE 5.7: Intel® Compute Stick.

1. The master has been set in the Remote PC, from which also the algorithm has been launched and run. In the Intel Compute Stick only the camera has been run;

2. The master has been set in the Remote PC and both the camera and the algorithm have been run in the Intel Compute Stick.

Before the related explanation it is important to underline that these tests have been done using the TurtleBot3 Burger present on the LIM and only the *Haarcascade classifier* algorithm because *Y.O.L.O.* needs different hardware component to run, Intel Compute Stick is not sufficient, so it is necessary another solution to run the deep learning algorithm in real-time.

First solution result:
The camera and the classifier have been mostly synchronized: the execution frequency of the algorithm has been often lower than the one necessary by the camera to refresh the image, so the movements of the robot have been fluid and fast. However, the movement of the Burger has been not coherent with the developed algorithm and that has been particularly evident in the linear velocity. In fact, sometimes the detection algorithm has not worked correctly because the Intel Compute Stick needed more time to update the depth camera matrix values. In this case, the distance robot-person detected has been closed to 0 [m], so there are two consequences:

- if the person is too near the robot, the robot moves rightly;

- if the person is far away from the robot, the robot moves intermittently forward and backward.

This problem was not noticed before in the simulation environment because the camera has been directly connected to the PC with an extremely performant graphics card, so the RealSense has run in real-time ensuring proper TurtleBot3 movements.

It has not been possible to solve this problem in any way, not even trying to adjust the frequency of the algorithm or stopping it for a set time. It cannot be usable.

Second solution result:
The camera and the classifier have run correctly. The control algorithm has given optimal result, but the computational time has been extremely high, so the movements of the robot have been too slow even if coherent with the detected distance both in linear velocity and angular velocity, exactly as during the virtual simulations. This problem is related to the absence of a video card in the Intel Compute Stick, so the algorithm runs only at computational level increasing the run time mostly when the classifier detects the person. Differently when the person has not been detected the run time has been too fast. Also this problem has not been possible to solve, so this solution is not performant.

According to the explained experiments, even if the use of the Intel Compute Stick could seem to solve the Intel® RealSense™ Depth Camera D435i necessity of a USB 3.0, both these configurations cannot be used for this thesis. The request of a real-time response of all the hardware components, software platform and algorithms is the most important thing to be respected, so it has been necessary to find a new device.

C) NVIDIA® Jetson™

The NVIDIA® Jetson™ is a System-on-Module able to guarantee high power efficiency and performance. Inside it contains DRAM, CPU, PMIC, flash memory storage and GPU.
The Jetson presentations are too high even if it has a small form factor, so that it is able to run algorithm of deep learning at high level performance. Depending on the applications there are many types of Jetson.
In this work thesis the necessity of an accurate and real-time execution is fundamental, so combining this with the hardware availability of the LIM it has been chosen the *NVIDIA Jetson AGX Xavier developer kit.*

Jetson AGX Xavier Series
It is a computer whose first application was destined to robots, drones and autonomous machines. It contains a set of AI tools able to train and run neural networks rapidly.
The developer kit of the Jetson Xavier (figure 5.8) makes possible many types of AI robotic applications. The kit is supplied with many software libraries as NVIDIA JetPack, DeepStream SDKs, CUDA®, cuDNN, and TensorRT, so it is ready to be used.
In the following figure 5.9 the technical specifications of the used card are shown.
The use of the NVIDIA Jetson AGX Xavier developer kit solves some problems previously analysed. In fact, the presence of an USB type C guarantees the RealSense

FIGURE 5.8: Jetson Xavier Developer Kit [36].

optimal operation during both the simulation environment and the tests on the robot. It is used as the PC of the TurtleBot3 with a support Remote PC wirelesses connected. This is the optimal solution found for this thesis project because, with the use of only one hardware, it is possible to run the two selected algorithms with a great efficiency also in the one of deep neural networks. However, the dimensions of this Jetson are not so limited, so there has been the necessity to change the TurtleBot3 components and structure in order to integrate perfectly all of them in the robot. Of course, the TurtleBot3 Burger could not be used anymore due to its compact and reduced dimensions, so the changes have been applied only in the TurtleBot3 Waffle and the result has been a creation of a new robot called *Jaffle*.

JETSON AGX XAVIER

GPU	512-core Volta GPU with Tensor Cores
CPU	8-core ARM v8.2 64-bit CPU, 8MB L2 + 4MB L3
Memory	16GB 256-Bit LPDDR4x | 137GB/s
Storage	32GB eMMC 5.1
DL Accelerator	(2x) NVDLA Engines*
Vision Accelerator	7-way VLIW Vision Processor*
Encoder/Decoder	(2x) 4Kp60 | HEVC/(2x) 4Kp60 | 12-Bit Support
Size	105 mm x 105 mm
Deployment	Module (Jetson AGX Xavier)

DEVELOPER KIT I/OS

Developer Kit I/Os	Jetson AGX Xavier Module Interface
PCIe X16	x8 PCIe Gen4/x8 SLVS-EC
RJ45	Gigabit Ethernet
USB-C	2x USB 3.1, DP (Optional), PD (Optional) Close-System Debug and Flashing Support on 1 Port
Camera Connector	(16x) CSI-2 Lanes
M.2 Key M	NVMe
M.2 Key E	PCIe x1 + USB 2.0 + UART (for Wi-Fi/LTE) / I^2S / PCM
40-Pin Header	UART + SPI + CAN + I^2C + I^2S + DMIC + GPIOs
HD Audio Header	High-Definition Audio
eSATAp + USB3.0 Type A	SATA Through PCIe x1 Bridge (PD + Data for 2.5-inch SATA) + USB 3.0
HDMI Type A/eDP/DP	HDMI 2.0, eDP 1.2a, DP 1.4
usD/UFS Card Socket	SD/UFS

FIGURE 5.9: Technical specifications of the Jetson Xavier Developer Kit [37].

5.5.2 Person tracking hardware

JAFFLE

The name Jaffle comes from the union between the Jetson and the Waffle.
It is an improvement of the TurtleBot3 Waffle in hardware and structure. The new robot uses the Jetson Xavier developer kit as PC which takes the data information from the RealSense depth camera D435i and the LiDAR and is able to communicate to the OpenCR card, which gives the command of movements to the motors.
In the Jetson integration we have three main problems:

1. the necessity of a Wi-Fi connection instead of the classical used of the Ethernet cable;
2. the problem of power supply without cable connection to the plug;
3. the not immediate compatibility between the Jetson and the RealSense camera used.

Solutions:

1. The Wi-Fi necessity has been easily solved using a USB Wi-Fi.
2. The power supply problem has been exceeded using an old battery which has been modified in order to respect the technical specifications required from the Jetson Xavier developer kit. The battery used is a LiPo 4S and in order to guarantee its correct work it has been created a sensor relevant battery discharge, which emits sounds when the battery needs to be recharged.
3. Many attempts have been done in order to solve the problem of compatibility of these two devices. In fact, to work with ROS the RealSense camera needs some packages and installations which are easy to obtain for common PC or Jetson TX1 or TX2, but too complicated for the Jetson Xavier. However, in the end they have been able to communicate and work together.

The result is the *Jaffle* (figure 5.10).

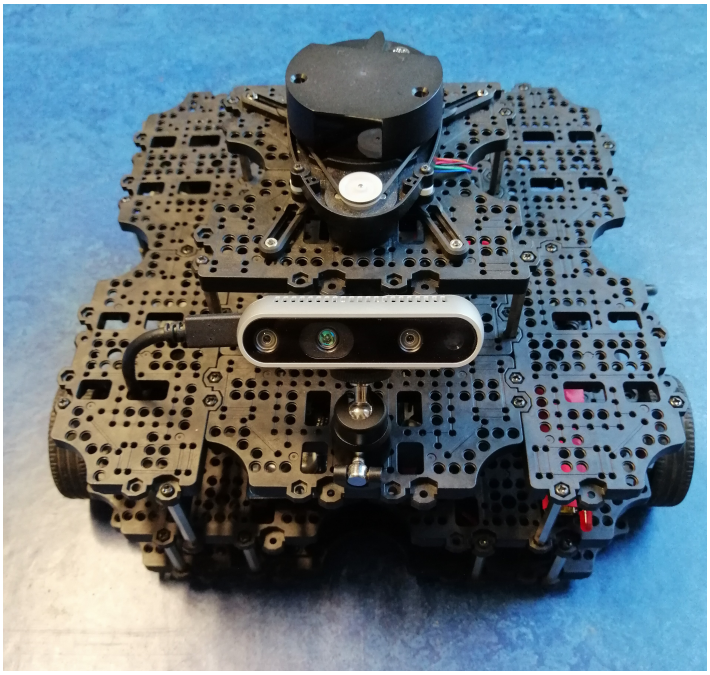


FIGURE 5.10: Jaffle.

5.5.3 Final choice Hardware components

The previous hardware component analysis has given as final result the union of the following hardware elements:

- Turtlebot3 waffle;
- Jetson Xavier Developer Kit;
- Intel® RealSense™ Depth Camera D435i;
- Battery LiPo 4S + sensor relevant battery discharge.

This has been considered the best choice to adopt in this work thesis.

5.6 Software used

5.6.1 Operating system and ROS platforms

Ubuntu is a free open-source operating system based on the Debian GNU/Linux distribution. The Ubuntu release used in this project is Ubuntu 16.04 LTS with the ROS Kinetic Klame platform installed inside.
ROS is a meta-operating system used for many combinations of hardware implementation. It has five characteristics:

1. Re-usability;

2. Communication-based program;
3. Support of development tools (visualization and simulation tools);
4. Active community;
5. Construction of an ecosystem.

The organization of the code must followed this scheme (figure 5.11):

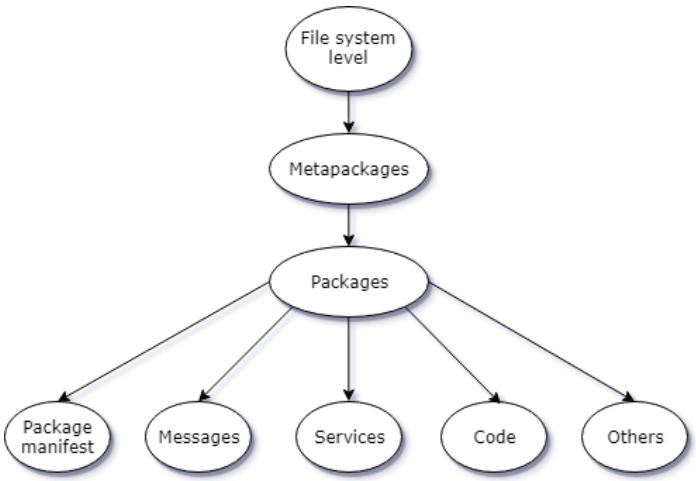


FIGURE 5.11: ROS code building organization.

- **Metapackage:** is a set of packages with the same purpose;
- **Package:** is a folder of files that are used for specific purpose. The files are executables, make-files or files indicating dependencies;
- **Node:** executable file that publishes a topic or subscribes to another.

The ROS execution structure is the following (figure 5.12):

- **Master:** is the top node which controls the topics and connects Subscribers with the Publishers;
- **Node:** process that performs the calculation. The ROS node is written using the library roscpp (C++) or rospy (Python);
- **Message:** is a data used in the topic.

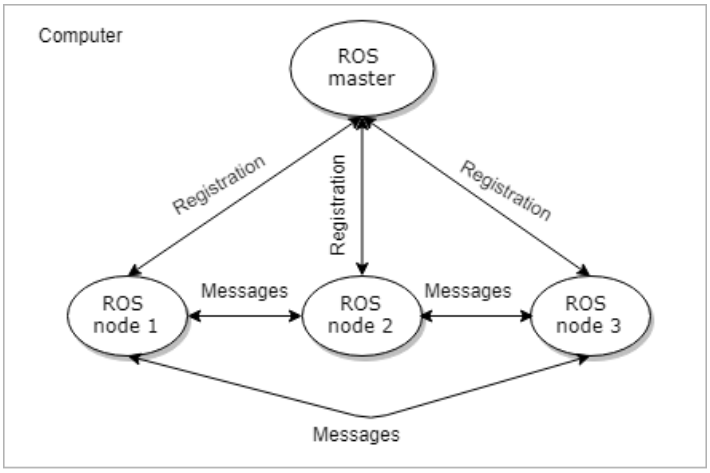


FIGURE 5.12: ROS execution structure.

5.6.2 Packages used in the project

- **librealsense2:** SDK 2.0 package used to communicate with the Intel® RealSense™ Depth Camera D435i [38];
- **realsense2_camera:** it is used for publishing the camera data using librealsense2 [39];
- **darknet_ros:** it is the package used for run Y.O.L.O. in ROS [40];
- **turtlebot3:** it is a meta-package containing many other packages internally used for configuring the robot motor and sensors and moving it [41].

5.6.3 Gazebo

Gazebo is a 3D environments simulator with high-quality graphics used especially in the robot navigation. It can run easily in the PC devices with ROS installed.
Gazebo provides robots, sensors and different choices of environments to have a realistic simulation of the robots' movement with its physics engine and the results are accurate. For this reason, it is the most used simulator for robotic applications.
It is possible to set different environments and light condition. The most common robots and sensors are already supported in this simulator, so it is possible to choose them or configured new ones, using a SDF file.
This simulator is used in this work thesis to test the correctness of the developed control algorithm.

5.6.4 RViz

RViz is the 3D visualization tool of ROS. It can display live the sensor information published into the ROS topics related to the active sensors (camera, LiDAR, IMU, ...).

It is usually used in the navigation applications, as for example obstacle avoidance, because it can construct a map and localize it inside the robot, so giving a goal the robot can move inside the map autonomously.
This visualization tool is used both for the live visualization of the topics of the Intel® RealSense™ Depth Camera D435i and for the obstacle avoidance part.

Chapter 6

Implementation

The implementation of a person tracking code consists of the union between an object detection algorithm and a control and regulation algorithm of the robot movements.
As in the previous chapters §2 §3 is anticipated, the object detection algorithms used for this work thesis are, for the already explained reasons, two types:

- *Haarcascade classifier*: cascade real-time algorithm belonging to the classical machine learning algorithms;
- *Y.O.L.O.*: Deep Neural Network real-time algorithm.

The movements of the Jaffle are controlled by the combination of linear and angular velocity regulation.
In the rest of the chapter it is explained how the implementation of the whole code is realized.

6.1 Person Detection

Person detection is done using the Intel® RealSense™ Depth Camera D435i. The camera is launched with ROS and many of its topics are used in the following algorithms.

6.1.1 Haarcascade Classifier

The Haarcascade Classifier is a real-time object detection method too old compared to the deep neural networks technology. However, from the previous analysis on the object detection algorithm, it seems to have a great performance and a low price, so it is the first algorithm developed and tried for this thesis.
As it has some troubles with the brightness, both RGB camera and Infrared camera are used in the code to avoid the not detected object problem. The Haarcascade classifier does person detection in two different, but parallel ways:

- Using **RGB camera** as first choice because it guarantees the best detection (the auto-exposure setting guarantees the best result also in backlight condition) and the depth camera set $aligned - to - color$ in order to obtain the same FOV and a match in the pixels.
- Using **Infrared camera** as second choice (generally it is used for all the distance detection and in darkness condition) and the depth camera set as $image - rec - raw$ in order to obtain the same FOV and a match in the pixels.

Independently from this choice it is found the position of the person in the space from the centre of the frame. The X and Y coordinates are in pixel instead the Z coordinate is in mm.
The Haarcascade classifier is introduced in the code simply uploading it as a .xml file with a cv2 function:

```
<variable_1> = cv2.CascadeClassifier(<path>/<classifier-name>.xml)
```

If the upload operation gives a positive result the video frames of the RealSense camera are converted in greyscale, eliminating many information, to facilitate subsequent operations. As both RGB and Infrared cameras are used in the algorithm, only the RGB one must be converted in greyscale. The other is already in this requested colour mode.
Then it is used another cv2 function in which many detection parameters are given to the classifier:

```
<variable2> = <variable1>.detectMultiscale(<grey-frame>,<scale_Factor>,
                  <min_Neighbours>,<min_Size>,<flags>)
```

the `<max_size>` parameter is not specified in this case, so it is automatically set to the default value.
The parameters set in the function are the detection settings, so they are calibrated dependently on the classifier used:

- `<scale_Factor>` : it indicates the factor of the image reduction.
- `<min_Neighbours>` : it is the number of neighbours that each rectangle should keep. It determines the quality of the detection: increasing this value means higher quality, but lower detections.
- `<min_Size>` : it is the maximum size of the detected object, so the larger objects are discarded.
- `<flags>` : it indicates the operation mode.

Person's 2D coordinates in the video frame

When the classifier detects a person in the video frame it is possible to compute the relative 2D coordinates. The operations are the same independently from the camera used (RGB or Infrared).
By using a cv2 function it is possible to design a rectangle around the person detected and know (see the figure 6.1) its left edge coordinates in pixel respect to the reference frame R0, the weight and the height, so consequently also the middle point of the bounding box respect to R0.

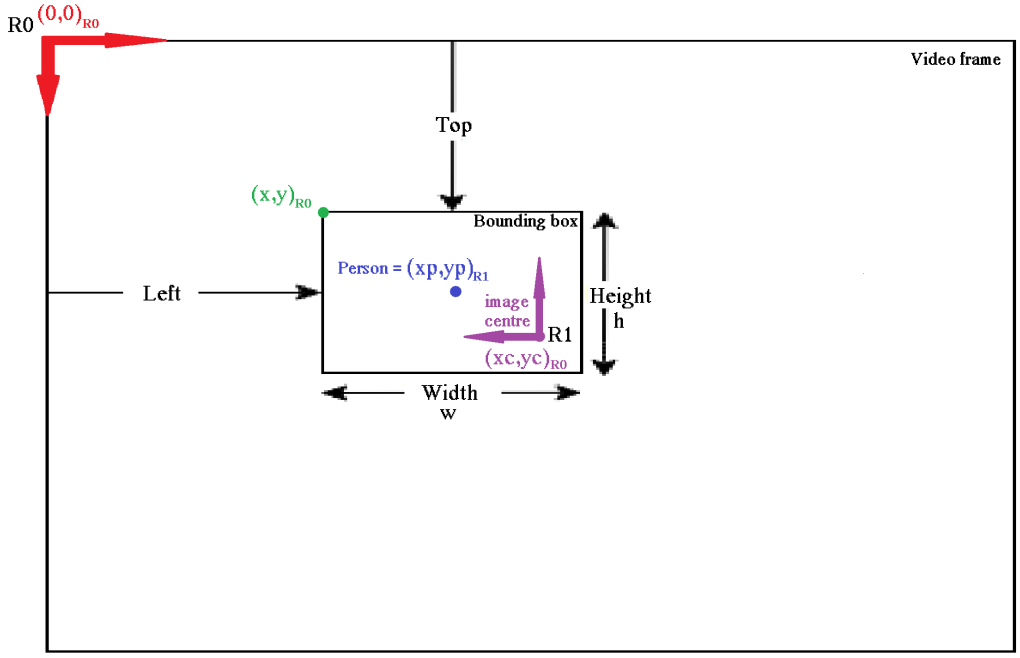


FIGURE 6.1: RF of the video frame.

By using the coordinates of the centre respect to R0 it is possible to calculate the coordinates of the centre of the person respect to the new reference frame R1:

$$x_p = x_c - \left(x + \frac{w}{2}\right)$$

$$y_p = y_c - \left(y + \frac{h}{2}\right)$$

where x_c and y_c are respectively 319 [pixel] and 239 [pixel] because the image resolution considered is 640x480.
The x and y coordinates of the person are necessary to locate the person in the 2D space and are fundamental to create the angular control algorithm, so to adjust the rotation of the robot (yaw).

Distance between the robot and the person

It is extracted from the depth camera topics, both the one aligned to the RGB camera and the one related to the infrared camera, the image in terms of matrices of values. The dimension of each matrix is the same of the resolution of the image acquired and in each pixel position there is a value in millimetres representing the distance between the camera and the object or walls found by the depth camera.
In order to find the right value of the distance between the camera and the person detected by the classifier is enough to have the depth matrix and the 2D coordinates in pixel of the centre of the person's bounding box, previously computed. The distance requested is the value of the depth matrix in the coordinate of the person's centre.
During the code developing, it has been tried to find the depth position of the person using as right distance the one calculated using the average of the depth matrix values related to the whole bounding box of the person. However, this solution has been discarded because sometimes the distance computed had completely incorrect values and also created discrepancies in the parallel use of the depth camera aligned to the RGB and the one related to the infrared.
The z coordinate completes the 3D localization of the person in the space and it is important to realize a linear velocity control algorithm that regulates the forward or backward movement of the Jaffle.

Detection situations

1. NOTHING DETECTED: if nothing is detected the robot stops. Differently, if the robot loses the person, it stops after some seconds in which it continues to run with the previous velocity commands;

2. ONE PERSON DETECTED: the robot follows the movement of the person.

It should be considered also a third situation: MORE THAN ONE PERSON SIMULTANEOUSLY DETECTED, but the Haarcascade classifier is unable to recognise the presence of more people. It generally detects one of them in a not deterministic order, so for example if there are two people in the indoor environment, during the first algorithm cycle it is detected the person1, at the second is detected the person2, at the third person2, at the fourth person2, at the fifth person1 etc... This causes a quick inversion of the robot movement because it does not understand which person has to follow. For this reason, this third case cannot be taken into account and in case of the use of this algorithm for person tracking it is necessary the presence of only one person in the indoor environment.

Two-code approach

The Haarcascade classifiers available for the person are many because also different type of face classifiers is included. However, as the robot has to follow the person, it is

assumed that the person is often identified with a back turn, so all the face classifiers became useless and therefore they have been discarded.
The remaining classifiers are: upper-body, lower-body and full-body. Depending on the depth camera used it is difficult to detect the complete person body on its range, so the full-body classifier is the weakest between the three and the most robust one is the upper-body because it is a small part of the body and the position and inclination of the RealSense camera increase its performance.
Two codes have been implemented:

1. Using only the upper-body Haarcascade classifier;
2. Using the three listed classifiers in decreasing cascade: full-body, lower-body and upper-body to try to increase the global robustness of the person detection. The cascade order of these classifiers is shown in the figure 6.2.

Both the two codes have been tested in simulation and in real indoor environment, but they are more or less the same. In fact, as the more robust cascade classifier is the upper-body one, it should be the only one that would detect the person in the frames in both the two codes. No detection related to the other two classifiers have been registered during the tests.
Moreover, the use of three classifiers in cascade makes the algorithm slow and not real–time, so it is better to use the code with only one classifier to have better results and performance.

In order to clarify all other doubts about the use of the Haarcascade classifier in this thesis it is reported the Flow Chart (figure 6.3) of the developed algorithm.

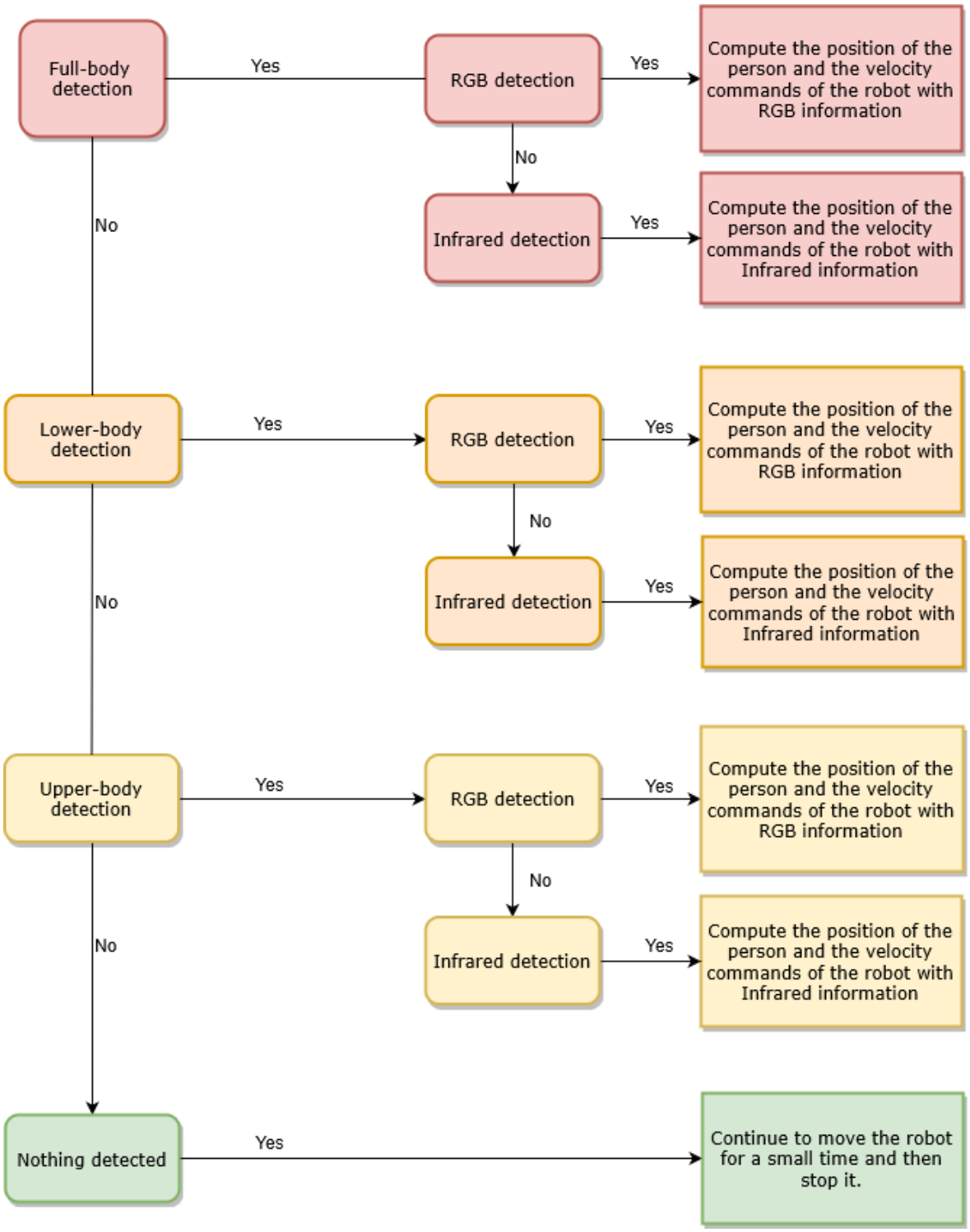


FIGURE 6.2: Scheme of the three Haarcascade classifiers in cascade.

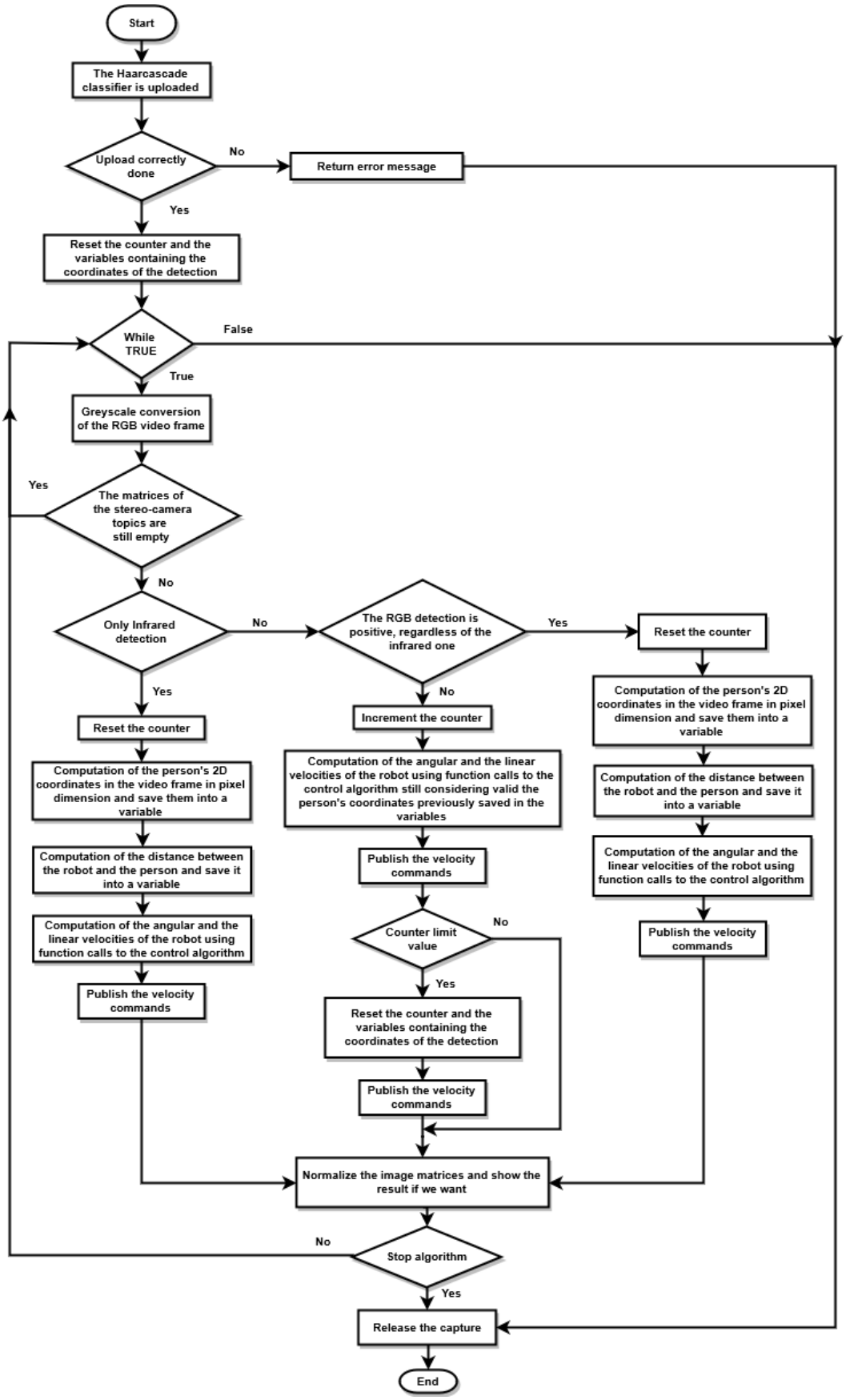


FIGURE 6.3: Flow Chart of the 'Haarcascade classifier' algorithm.

6.1.2 Y.O.L.O.

Y.O.L.O. is a Deep Neural Network real-time object detection method extremely accurate.
It operates applying a neural network to the full image, which is divided into regions. In each region the bounding boxes are weighted by the predicted probabilities and the predictions are made with a single network evaluation. This means that is extremely fast.
Its version 3 is considered by now the fastest and most performing network tested on COCO dataset.
Surely it is more complicated than the Haarcascade method because needs more computation time and it is more expensive because it runs with the support of high level hardware components. In fact, to run this algorithm it has been used the Jetson Xavier developer kit §5.5.1.
In this method only the RGB camera topic and its related depth camera are used, so the Infrared camera information are not considered for two main reasons:

- Firstly, because the results of the only RGB camera are already too accurate and, as the network computations are too complex, it is preferable to not increase the computational time with other calculations, which could affect making the algorithm no longer real-time;
- Secondly, because, even if from the tests carried out it is possible to see the good behaviour of the network in Infrared mode, the number of correct detections are inferior. This happens because the network is trained and tested with the COCO dataset, which contains only coloured images, so it should be necessary to make a new dataset of infrared images and retrain and test the network with it. This operation is too long and not necessary in this thesis project.

How Y.O.L.O. works in ROS

The use of Y.O.L.O. network inside ROS platform is not so intuitive. The problem is solved using a GitHub package called **darknet_ros** [40]. This package subscribes to the topic of the camera and using Y.O.L.O. it publishes some topics related to the detections:

- `found_object` : it gives the value related to the number of objects detected in the frame.
- `bounding_boxes` : it is an array containing the information of the class detected with its related probability and the position and size of the bounding boxes in pixel dimension.
- `detection_image` : it publishes the initial image including the bounding box of the objects detected.

- `check_for_objects` : there are some further information about the detections, as the status, the feedback and the result. These depend mainly on the detection percentage set as a parameter before launching the node.

It is possible to launch the darknet_ros node using different versions of Y.O.L.O. To do that, it is necessary to set the weight and configuration files related to the version chosen in the correct folders and launch the node.
In this project the version of Y.O.L.O. used are three:

- Y.O.L.O.v2;
- Tiny-Y.O.L.O.v3;
- Y.O.L.O.v3.

These versions are chosen in order to guarantee optimal real-time performances of the detection.

Dataset

The dataset chosen affects the results because it contains all the resources used during the training phase of the network. For this thesis the use of a huge dataset of images which contains different classes other than person is fundamental because the network can learn to identify also different elements, edges and reduce the wrong detections.

- COCO

Common Object in COntext (COCO) is one of the most common dataset used for the training of the neural networks for object detection applications. It contains 80 classes of objects with their respectively marks and bounding boxes.

Re-Training

The training of a network from a scratch requires a lot of time and a huge computational time. Moreover, the dataset adopted has to be extended enough to include also the classes that do not need to be recognised. To avoid that problem a pre-trained model is used as the starting point and then it is re-used for other purposes. This approach is called *Transfer Learning*: "Transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned." [42]. The idea is that the whole objects in the images are defined from shadows, edges and colours, so the first part of different networks is similar. Then the network can be re-trained for different specific tasks, so to recognise specific classes. In this way the user could do a re-training with low-powered computer and a smaller dataset.

As the scope of the thesis is doing person tracking, the use of the pre-trained networks for the entire COCO dataset is not necessary, so it is done a *Transfer Learning* re-training of the networks Y.O.L.O.v2, Tiny-Y.O.L.O.v3 and Y.O.L.O.v3 just for the class *person*.

The hardware used for the re-training is a machine mounted an i7-9700K Intel CPU, 32 GB of RAM memory and 32 GB of swap memory. The most important components of the systems are the two available GPUs:

- #1 NVIDIA GeForce RTX 2080 Ti with 11 GB GDDR6 memory;
- #1 NVIDIA GeForce RTX 2080 with 8 GDDR6 memory.

The realization of a good re-training of the networks depends on the creation of a dataset of people and on the light modification of the network [43].

- Dataset of People

Even if the first training of the three Y.O.L.O. networks used was done using the COCO dataset, which contains 80 classes of object including *person*, it has been necessary to re-train the networks just for the class person hoping to obtain better mAP results and a more robust system. The creation of a new dataset has been done selecting a set of figures in .jpg format and making bounding box around the people. The bounding boxes related to each image are defined into a .txt file using the following format:

```
<object-class> <x> <y> <width> <height>
```

where:

- `<object-class>` is an integer number referred to the object from 0 to (classes-1);
- `<x> <y> <width> <height>` are normalized float values referred to the centre of the rectangle bounding box, the width and the height of the image which can be equal to $[0.0 \div 1.0]$.

The images of person used to create the People dataset were extracted from *Open Images v4* [44]. Open Images v4 is a 9 million images dataset with their related labels and objects bounding boxes. These images are too different and contains many objects for each one in several scenarios. The dataset is divided into train, validation and test to prevent overlaps.

The extraction of the person class images has been done using *OIDv4_ToolKit* [45]. The toolkit is able to download both a single image and the images related to one class or more specific classes of objects providing also the .txt files of information of the respectively labels with the correct format.

The images used for the re-training were 6001: 5401 images for the train phase and 600 images for the test phase.

- Network Modifications

During the re-training the networks used have to be modified to detect only people. The number of classes inside the .cfg file has been reduced from 80 classes of the COCO dataset to 1 class (person), so the output of the network can be only person or nothing in order to increase the accuracy of the object detection for this specific class. Also other parameters have been modified:

- *batch*: it is the set of images used during the training of each epoch. In Y.O.L.O.v2 and Tiny-Y.O.L.O.v3 this value is equal to 64 instead in Y.O.L.O.v3 it is 32 and the subdivisions value has been modified depending on the memory available and on the type of networks. The subdivisions values used are: 8 for Y.O.L.O.v2; 4 for Tiny-Y.O.L.O.v3; 8 for Y.O.L.O.v3.

- *max_batches*: it is the max number of batches used and it is equal to $(classes * 2000)$.

- *filters*: it is the value of filters in the 3 convolutional networks before each Y.O.L.O. level and it is equal to $(classes + 5) * 3$.

- *input image dimension*: every networks have a predefined dimension which depends on the dimension of the images used. In this project the image taken from the Intel® RealSense™ Depth Camera D435i had dimension 480x640, so the input image dimensions of the networks were imposed equal to 416x416.

- *learning rate*: this parameter changes in different ways during the training, so it is possible to identify three steps depending on the descent theory: until the 200^{th} step it is multiply by 10; from 200^{th} to the 500^{th} step it is reduced by 10; from 500^{th} to 700^{th} is again reduced by 10.

Person's 2D coordinates in the video frame

As it is explained above, the results of the detection are published by the darknet_ros node in differents topics. The topic of `darknet_ros/bounding_boxes` gives directly the pixel values of the angles of the bounding box, related to the detected object, respect to the R0 reference frame (figure 6.1). Using the same equation explained for the Haarcascade algorithm §6.1.1 it is possible to find x_p and y_p coordinates of the centre of the person respect to R1 (figure 6.1) and use them in the control algorithm.

Distance between the robot and the person

The computation of the z coordinate of the person detected is done in the same way of the Haarcascade algorithm §6.1.1. The only difference is that it is related only to one depth camera topic: the one aligned to the RGB camera, so it does less computations.

Detection situations

There can be three possible detection situations:

1. NOTHING DETECTED: if nothing is detected the robot stops. Differently if the robot loses the person, it stops after some seconds in which continues to run with the previous velocity commands;
2. ONE PERSON DETECTED: the robot follows the movement of the person;
3. MORE THAN ONE PERSON SIMULTANEOUSLY DETECTED: the robot stops for 15 min. After that, it restarts the detection and continues to remain stopped if it finds again more than one person.

In order to clarify all other doubts about the use of Y.O.L.O. networks in this thesis it is reported the Flow Chart (figure 6.4) of the developed algorithm.

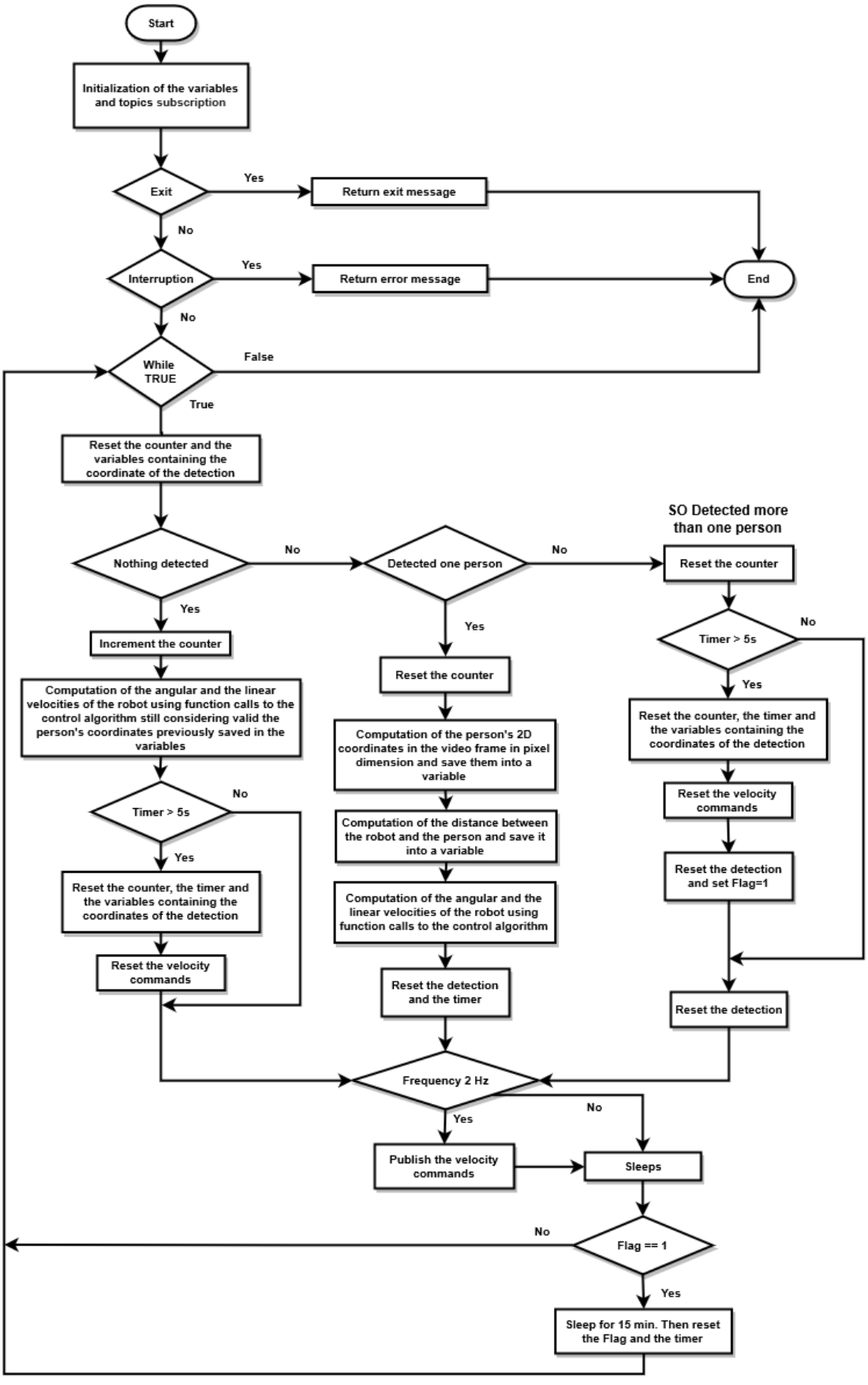


FIGURE 6.4: Flow Chart of the algorithm of the 'Y.O.L.O.' networks.

6.2 Control Algorithm

6.2.1 Introduction

The control of a dynamic system is one of the most important thing to do in order to limit the wrong behaviour of the automated objects. The input data of a system must be controlled continuously in order to have right output values.
In this thesis work the control algorithm developed is able to regulate the movements of the Jaffle according to the tracking necessity. The input data are the information receiving from the object detection algorithms, so in this case the position of the person in the space and the output are the computed values of angular and linear velocities of the robot.
The angular and linear velocity are regulated in different ways and the movement of the Jaffle is the result of the combination of both of them simultaneously.

6.2.2 Angular Velocity

The angular velocity is regulated by using the information of the dx position in pixel of the person detected in the video frame.
According to the green reference frame of the stereo-camera, shown in the figure 6.5, the dx position is considered positive when the person is on the left side of the video frame, differently on the right side it is a negative value. Consequently, when the centre of the person detected is in the upper side dy is positive, differently is negative.

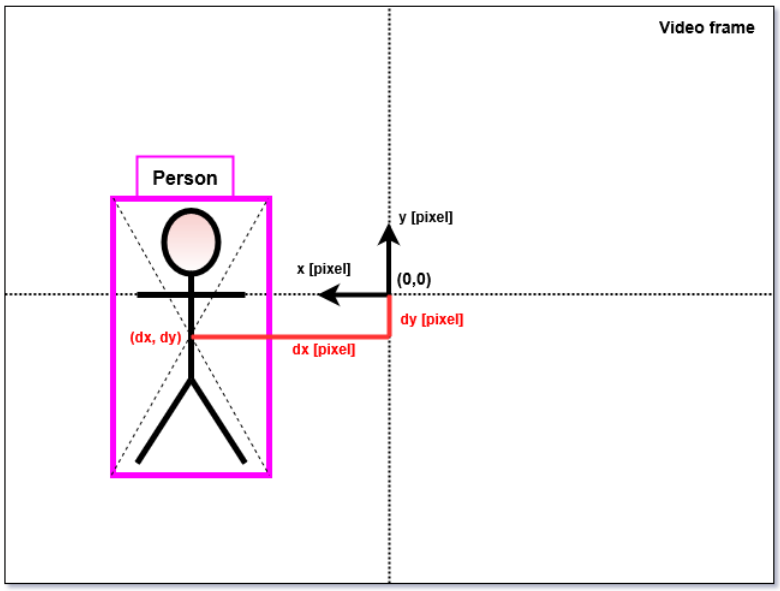


FIGURE 6.5: 2D pixel coordinates of the person in the video frame.

The dy position value is not important for the angular velocity computation because the robot cannot move up or down in the Y-axis, but only in the X or Z axes.
The rotation around the Y-axis has been set with a parabolic course because this makes the movements of the Jaffle smooth without jerks.
The function generated has as input the dx position in pixel and gives as output the angular velocity according to the following formulas:

$$v_{angular\theta} = \frac{max_{vel} * dx_{pixel}^2}{320^2} \left[\frac{rad}{s}\right] if dx_{pixel} > 0$$

$$v_{angular\theta} = \frac{min_{vel} * dx_{pixel}^2}{320^2} \left[\frac{rad}{s}\right] if dx_{pixel} < 0$$

$$v_{angular\theta} = 0 \left[\frac{rad}{s}\right] if dx_{pixel} = 0$$

The terms max_{vel} = 1.8 [rad/s] and min_{vel} = -1.8 [rad/s] are the upper and the lower limit of the angular velocity of a TurtleBot3 Waffle, so consequently of also the Jaffle. The number 320 [pixel] stands for the max number of pixels for each side (left and right) of the video frame because the resolution of the image received from the RealSense camera is 640x480.
In the following figure 6.6 is represented the angular velocity behaviour developed:

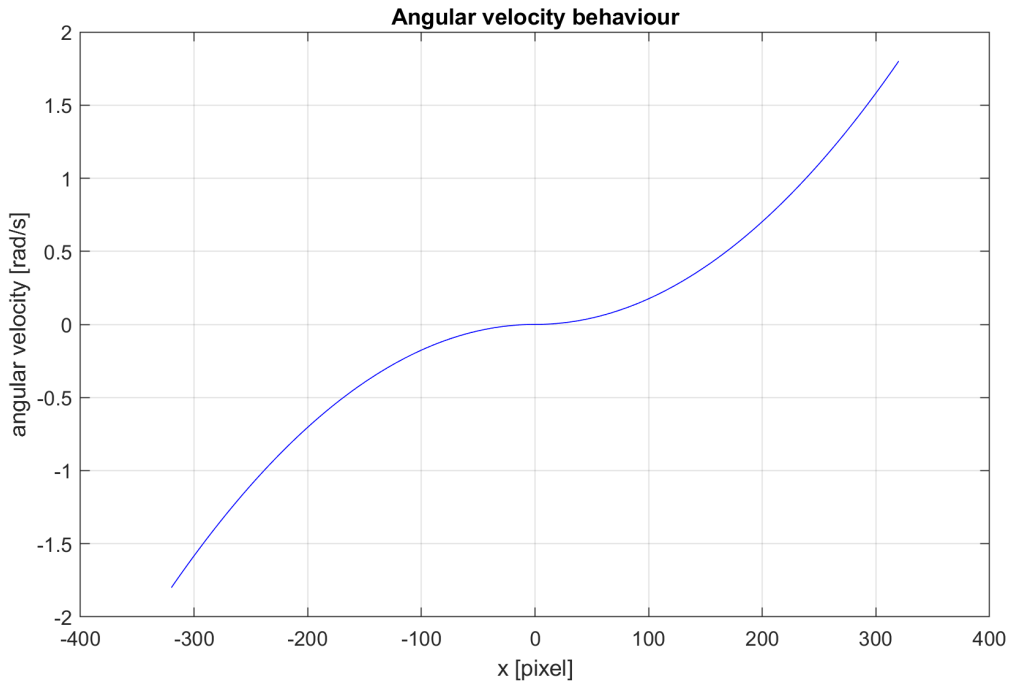


FIGURE 6.6: Angular velocity control behaviour.

The whole values and robot movements are coherent to the movements of the person both in simulation environment and in tests environment.

6.2.3 Linear velocity

The linear velocity is more trivial to compute because there has been the necessity to set optimally two limit distances behind which the robot changes its movement. This is why this control can be divided in three regions, depending on the distance between the Jaffle and the person detected. In the first one this distance is superior compared to the set upper-limit, so the robot moves straight on in a linear proportional way until it reaches the maximum speed of the robot and saturate at that value. In the second region there is a stop condition, in fact, the robot is not so distant nor near the person and remains at that distance to avoid losing the person to track. The zero value is also assigned when the distance is 0 [m] only to avoid special case in the code. In fact, the depth value obtained from the Intel® RealSense™ Depth Camera D435i has a limit of 0.105 [m], so nothing should be detected at a lower distance. The final region is between the limit of the camera: 0.105 [m] and the distance lower-limit. In this condition the Jaffle goes back in a linear proportionally way until it reaches its minimum speed and saturated at this value.
As it is explained the linear velocity is just a matter of distance between the robot and the person detected. This information is received from the stereo-camera and it is the linear velocity control function. Depending on it, the velocity is in a precise region, so it is possible to compute the linear velocity using the depth measure as unknown. Here are reported the formulas responsible of these computations:

$$v_{linearx} = depth_m * m1 + q1 \left[\frac{m}{s}\right] if depth_m > m_{vel_{upperlimit}}$$

$$v_{linearx} = 0 \left[\frac{m}{s}\right] if m_{vel_{lowelimit}} < depth_m \leq m_{vel_{upperlimit}}$$

$$v_{linearx} = 0 \left[\frac{m}{s}\right] if depth_m = 0[m]$$

$$v_{linearx} = depth_m * m2 + q2 \left[\frac{m}{s}\right] if depth_m \leq m_{vel_{lowelimit}}$$

At the beginning the lower and upper distance limits have been clumsily estimated, but then, during the tests, they have been changed and the optimal values are found (table 6.1)

Limits	**Haarcascade classifier**	**Y.O.L.O.**
m_vel_upperlimit	1.2 [m]	1.9 [m]
m_vel_lowerlimit	0.9 [m]	1.7 [m]

Table 6.1: Lower and upper distance limits.

The values $m1$, $q1$, $m2$ and $q2$ are found using the equation of the straight line passing through two points. In the following table 6.2 it is shown the final value chosen:

Straight line	Points	Haarcascade classifier	Y.O.L.O.
1° : (m1, q1)	*P1*	(1 [m], 0.13 [m/s])	(1 [m], 0.23 [m/s])
	P2	(3 [m], 0.26 [m/s])	(3 [m], 0.26 [m/s])
2° : (m2, q2)	*P1*	(1 [m], -0.13 [m/s])	(1 [m], -0.23 [m/s])
	P2	(0.3 [m], -0.26 [m/s])	(0.3 [m], -0.26 [m/s])

TABLE 6.2: From points to straight lines.

The behaviour of the linear velocity for the Haarcascade classifier and the Y.O.L.O. application is reported respectively in the figures 6.7 and 6.8.

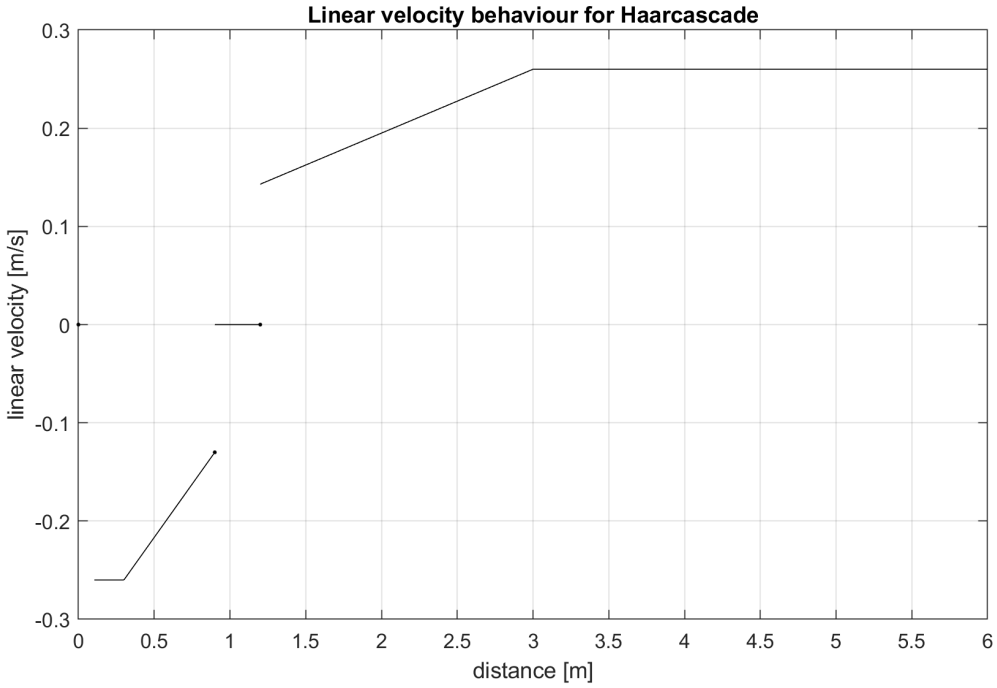


FIGURE 6.7: Linear velocity control behaviour in Haarcascade algorithm.

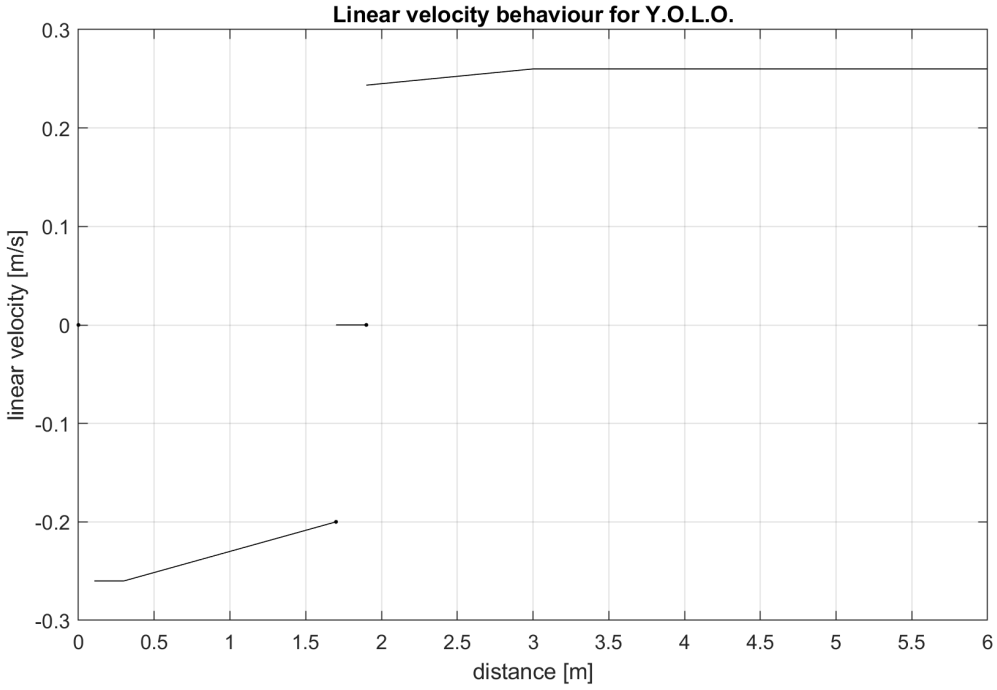


FIGURE 6.8: Linear velocity control behaviour in Y.O.L.O. algorithm.

Chapter 7

Obstacle Avoidance

One of the limits of the developed algorithms is the presence of obstacles. In fact, when the Jaffle tracks the person in the indoor environment could find some obstacles in its path, so it is necessary to prevent the robot from colliding with obstacles and therefore losing the person it has to follow.
For these reasons, the introduction of a real-time obstacle avoidance support to the algorithms used so far can be considered a great upgrade of this project.

7.1 Introduction

The *Obstacle Avoidance* topic is one of the main problem in mobile robotics. The goal of the autonomous navigation is to find the best and optimized path from the initial position of the robot to the goal to reach, taking into account the presence of object in the environment, so supporting the algorithm with obstacle avoidance competence. The autonomous navigation algorithms can be divided in two main categories depending on the type of control that is adopted:

1. *Global Path Planning Method*;
2. *Local Motion Control*.

7.1.1 Global Path Planning Method

Global path planning needs the a priori model of the map of the environment, on which the robot moves, and computes the shortest path from the initial pose of the robot in this map to the goal it has to reach.
Generally, it is used in the indoor environments or in the partially or totally known outdoor environments. Once known the map, these algorithms are able to update real-time the information on the map, in case the environment information rapidly changes, so to recalculate the path to follow in order to reach the goal.
The algorithms related to this method are A*, which gives an optimal global path in the static environment, and D*, an upgrade of A* used in the dynamic environment.

7.1.2 Local Motion Control

The local motion control is generally used for real-time motion of the robot inside unknown environment with the sensors, that are able to identify the obstacle and generate a motion path able to avoid collisions.
This method does not need the a priori knowledge of the environment map, so it can generate a new path when the environment changes.
The obstacle avoidance in an unknown environment is more complex to realize because a small position error can affect in the wrong movement of the robot, so a wrong map is reconstructed and a wrong path is generated. Regardless of this limitation, the obstacle avoidance techniques are usually faced in absence of environment information, so this method is the most common used.
These algorithms can be divided in two types depending on their adopted approach:

- Directional: Potential field method, Virtual Force Field, Vector Field Histogram and Nearness Diagram algorithm;
- Velocity space-based: Curvature Velocity method and Dynamic Window method.

7.2 Obstacle Avoidance Implementation

A great autonomous navigation robot should have the integration of a global path planning algorithm supported by a local motion control, so that the global system with a priori map can estimate the optimized path inside the map and the local system modifies this path in case of presence of further unpredicted obstacles, using sensors.

In this project the methods based on global path planning have been discarded because the best idea is to create a system that works well in any indoor environment, without having a priori information about the map. The absence of this constraint generates a global solution, more practical because it does not require specific modifications based on the different surrounding environments.
Moreover, the obstacle avoidance integration must not affect the performance of the person tracking execution, so it is necessary a real-time algorithm with local motion control.

The best idea found is to integrate an obstacle avoidance algorithm with dynamic and real-time goal. The dynamic goal is the position of the person's centre respect to the robot, so, as the goal changes with the movement of the person, also the robot modifies its path depending on it and avoiding the obstacles. The update goal replaces the previous one, so the robot does not reach the goal even if the person stops.
Obviously, some limits are set on the robot, which can never reach the position of the person, but remains at a certain distance between the goal:
yaw_goal_tolerance=0.05 [rad] and xy_goal_tolerance=0.5 [m].

The obstacle avoidance integration is done simply using the `move_base` node provided into the turtlebot3_navigation package.
Here is shown the high-level view of the move_base node (figure 7.1).

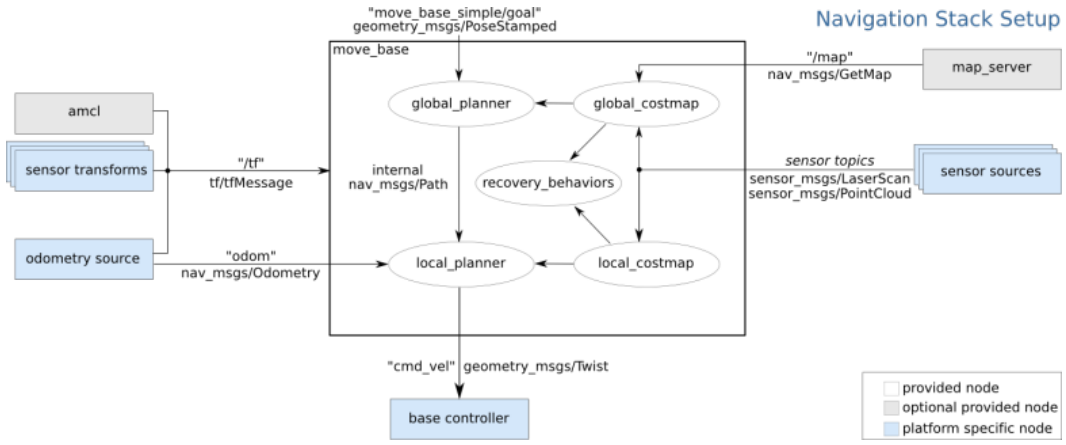


FIGURE 7.1: Move_base navigation stack [46].

By running this node, the robot tries to reach the dynamic goal with a pre-set tolerance. In absence of dynamic obstacles, the robot movement will be linear optimized, differently when the robot is stuck it performs recovery behaviours. In fact, it does an in-place rotation in order to find free space and if this does not end correctly, the robot removes the obstacles from the map and performs another in-place rotation. Now, if the robot fails, it advises the user about the impossibility to reach the goal and it stops its navigation, differently it finds another path and reaches the goal.

7.2.1 Estimation of the goal-pose of the person

The main problem faced is the computation of the pose of the person in the 3D environment in order to give the correct goal to the robot for the autonomous navigation. As it is explained before §6, the person is localized in the space, but not all the 3D coordinates are expressed in meters. In fact, for now the X and Y coordinates in the frame are measured in pixel and only the depth distance is in meters.
To compute the position of the person in the space is necessary to:

- convert the pixel coordinates into the camera coordinates to compute the position of the goal respect to the base_footprint RF;
- compute the YAW angle and consequently the quaternion of the goal respect to the base_footprint RF;
- with some transformation matrices calculate the goal position referred to the map RF.

From Pixel coordinates to Camera coordinates

By considering the figure 7.2 it is possible to obtain the 3D coordinates (X, Y, Z) of an object in the space through two conversions:

1. from Pixel coordinates (u, v) to Image coordinates (x, y);
2. from Image coordinates (x, y) to Camera coordinates $(Xcam, Ycam, Zcam)$.

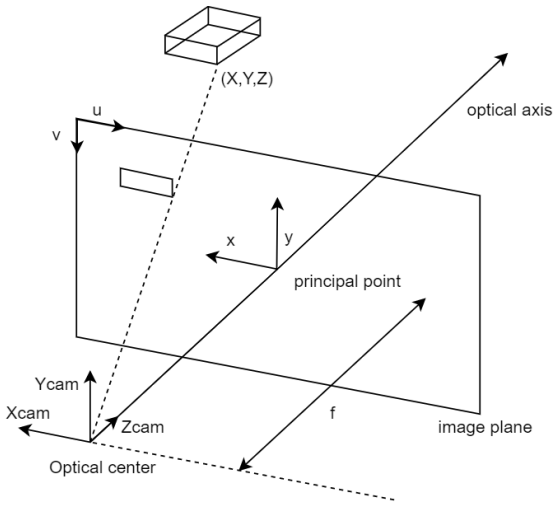


FIGURE 7.2: From 2D image projection to 3D coordinates.

1. From Pixel coordinates (u, v) to Image coordinates (x, y):

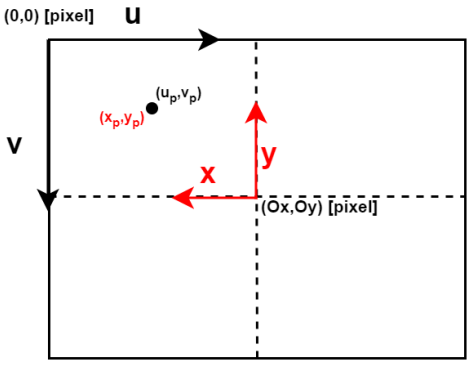


FIGURE 7.3: From pixel coordinates to image coordinates.

This conversion is simple; it is necessary only to do a subtraction operation in pixel using the following equations:

$$\begin{cases} x_p = -(u_p - Ox)[pixel] \\ y_p = -(v_p - Oy)[pixel] \end{cases}$$

2. From Image coordinates (x, y) to Camera coordinates $(Xcam, Ycam, Zcam)$: The Z coordinate is the value obtained from the depth camera matrix §6.1.1, instead $f_x = 613.2378540039062[pixel]$ and $f_y = 612.938232421875[pixel]$ are the focal lengths of the Intel® RealSense™ Depth Camera D435i in the pinhole model respectively of the x and y axis.

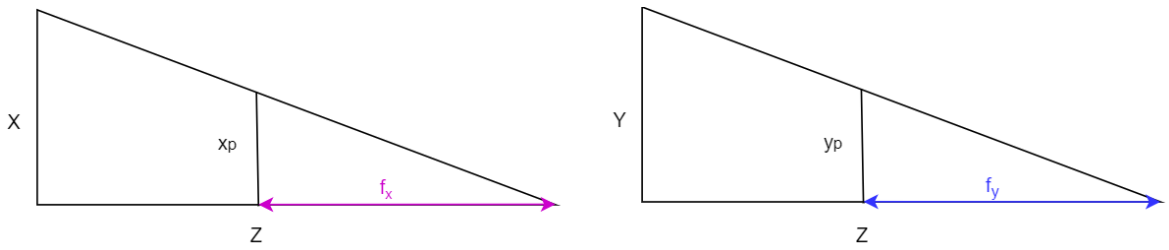


FIGURE 7.4: From image coordinates to camera coordinates.

$$\begin{cases} X = \dfrac{x_p * Z}{f_x}[m] \\ Y = \dfrac{y_p * Z}{f_y}[m] \\ Z = Z_{depth-camera}[m] \end{cases}$$

From Camera coordinate to base_footprint RF

Once obtained the X,Y,Z coordinates in meters of the centre of the person to track, it is possible to compute the goal respect to the base_footprint reference frame. This goal must be published giving these information:

- Position: x = Z [m], y = X [m], z = 0 [m] because no vertical movement;
- Orientation: x, y, z, w. These are the quaternions, easily computed using a python function: `<quaternion=trans.quaternion_from_euler(0,0,yaw)>`, related to the position of the centre of the person respect to the base_footprint reference frame. In this case, as the robot does not move in vertical direction, x = 0 and y = 0, differently z and w depend on the YAW angle (see the figure 7.5).

The YAW angle is equal to:

$$YAW = atan\left(\frac{y}{x}\right)[rad] \tag{7.1}$$

where x is a depth distance, so always positive (≥ 0.105 [m], which is the limit of the Intel® RealSense™ Depth Camera D435i), and y is positive if the person is on the left and negative if the person is on the right.

By using the coordinate position of the goal and its yaw angle it is possible to compute the transformation matrix from the base_footprint RF to the goal:

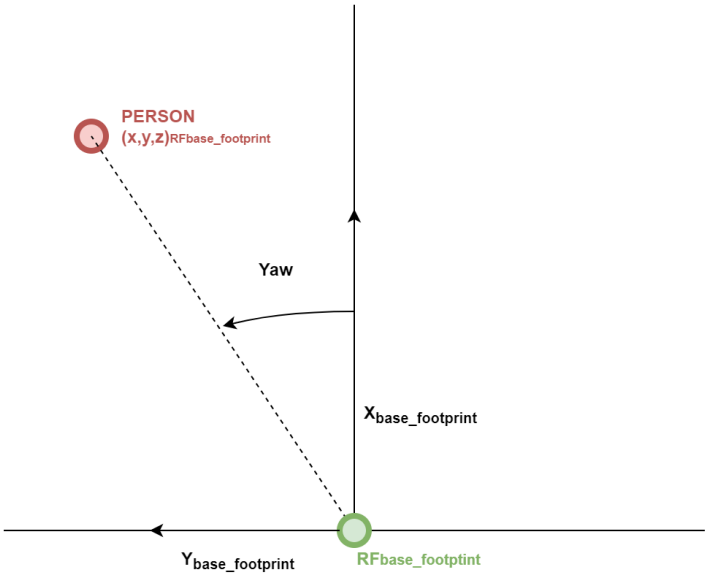


FIGURE 7.5: $RF_{base\_footprint}$ and YAW angle.

$$T_{basefootprint\_goal} = \begin{bmatrix} cos(\psi_{bg}) & -sin(\psi_{bg}) & 0 & x \\ sin(\psi_{bg}) & cos(\psi_{bg}) & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Goal in map RF

The next passage to do is to compute the position of the goal respect to the **map** reference frame.
The transformation tree is the following:

- `/map`: the coordinate frame fixed to the map;
  - `/odom`: the coordinate frame related to the odometry only;
    - `/base_footprint`: the base of the robot on the floor.

The `/odom` data are just respect to the map, so using the transformation matrices from odom to base_footprint and from base_footprint to the goal it is possible to compute the person's pose, so the goal respect to the map:

$$T_{map\_goal} = T_{map\_odom} * T_{odom\_basefootprint} * T_{basefootprint\_goal} \tag{7.2}$$

Now from the T_{map_goal} it is possible to find the position of the centre of the person as position and orientation coordinates respect to the map RF and this information is published into the `/move_base_simple/goal` and updated at each detection.

7.2.2 Detection situations

There can be three possible detection situations:

1. NOTHING DETECTED: if nothing is detected the robot stops. Differently, if the robot loses the person, it stops after some seconds in which continues to run with the previous velocity commands trying to reach the last goal received;
2. ONE PERSON DETECTED: the robot follows the movement of the person using the obstacle avoidance algorithm, setting as goal the person's centre;
3. MORE THAN ONE PERSON DETECTED: the robot stops for 15 min. After that, it restarts the detection and continues to remain stopped if it finds again more than one person.

Chapter 8

Results and Conclusions

In this section are presented the results obtained from the different object detection techniques, comparing them and founding the best one to use for this scope, and also from the integration of an obstacle avoidance support.

8.1 Haarcascade Classifier algorithm

As it is explained previously §6.1.1, the implementation of the Haarcascade Classifier has been done using both RGB camera and infrared camera. This double cameras approach increases the efficiency of the detection, also solving the back-light problem. The idea of the use of three Haarcascade Classifiers in cascade has not been suitable because the detections are always obtained with the upper-body classifier, the most robust one, moreover this solution increases drastically the computational time of the algorithm, so the movement of the robot are too slow.
In order to optimize the performance of the algorithm the best solution is to use only the upper-body classifier.
The tests of this method are done firstly on Gazebo simulator and then on the Jaffle, using for simplicity also the frontal face classifier.

8.1.1 Qualitative Results

The results obtained are the best reachable with the classical machine learning algorithms. In fact, using the frontal face classifier the algorithm runs extremely fast on the Jetson Xavier Developer Kit and the movements of the robot are coherent with the control algorithm implemented.
The main problem is due to the presence of false positives, generally detected at low depth distance, which makes the movement of the robot not fluid.
By using the upper-body classifier the algorithm is not robust because the person is not always detected and even if it runs on the Jetson Xavier Developer Kit, it is slow and not performant.

As the request of the project is to create an algorithm able to do person tracking generally the person is followed by behind and the face is seen only in case of inversion of direction movement of the person, so the frontal face classifier is not sufficient for this application. Moreover, even if this classifier would be enough, the problem of false positives would make this approach unusable anyway for this application, so this method is discarded.
Here are shown some results obtained with this algorithm. Each figure has the same structure:

- on the top the detections both in Infrared and RGB mode are displayed;
- on the left the velocity commands given to the robot are provided;
- on the right the information about the detection and the position of the person, if it is found, are supplied.

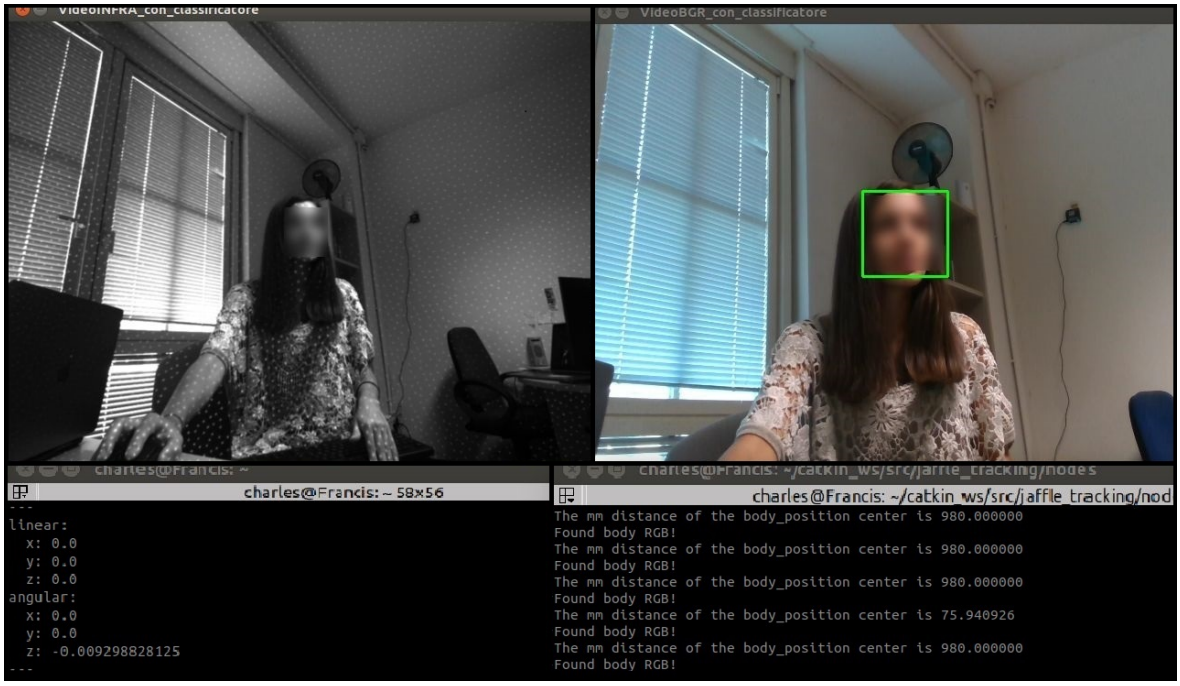


FIGURE 8.1: Haarcascade classifier: RGB detection.

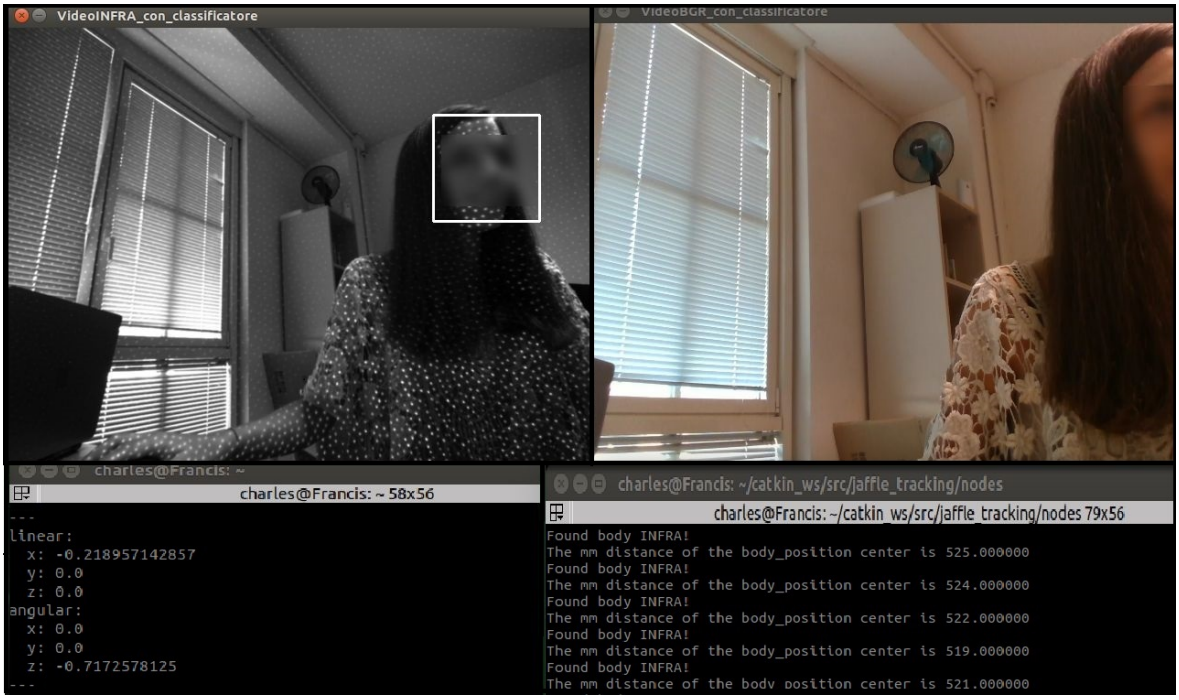


FIGURE 8.2: Haarcascade classifier: Infrared detection.

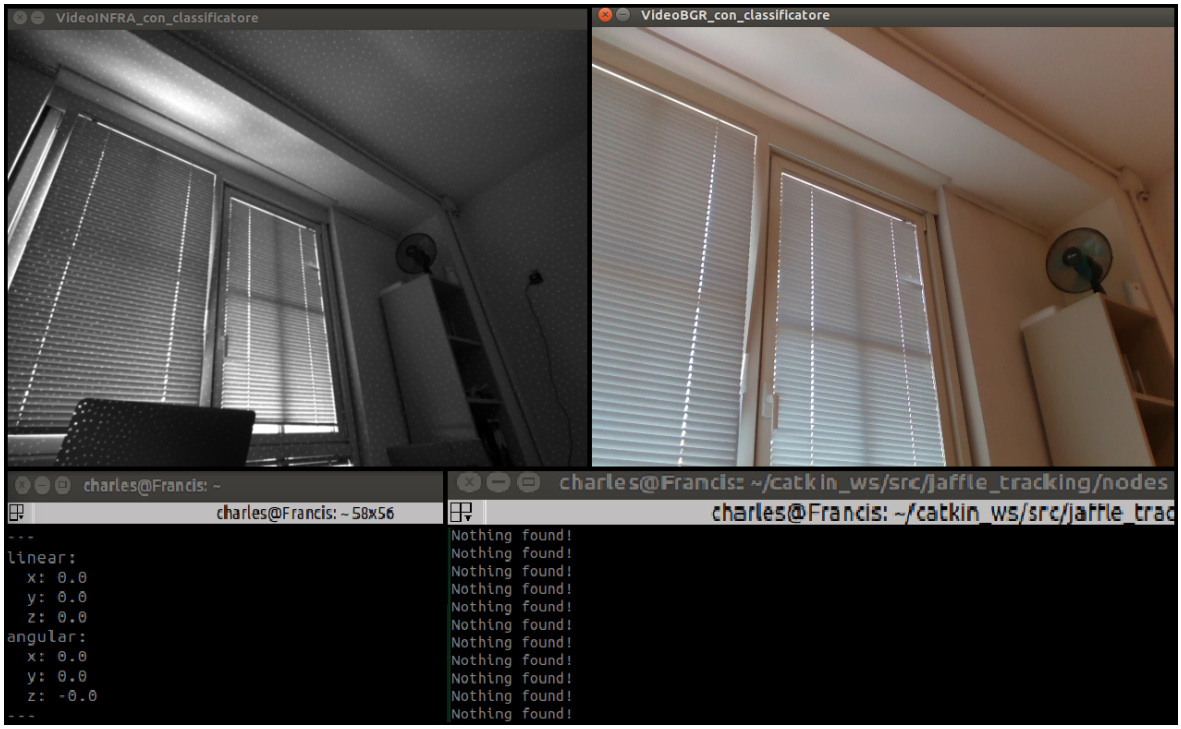


FIGURE 8.3: Haarcascade classifier nothing detected.

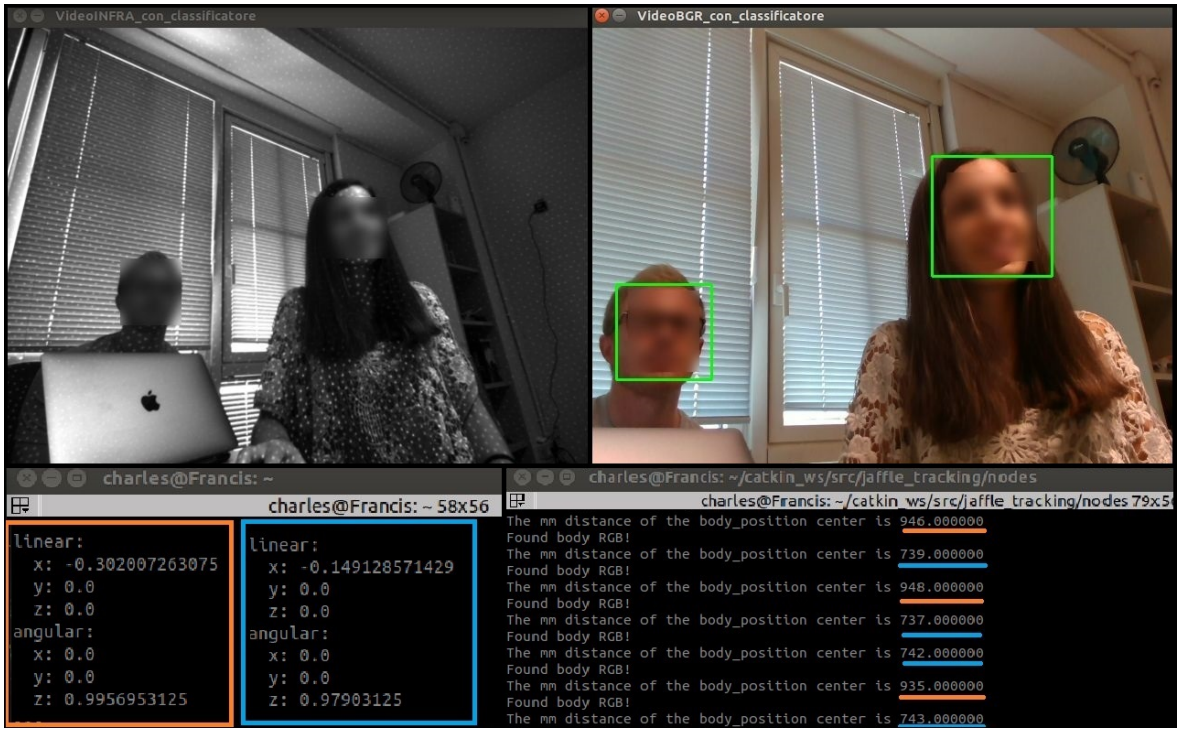


FIGURE 8.4: Haarcascade classifier: two people detected. The detection is casual, there is not alternation, but randomly it detects previously one person and then another, as it can see in the underlined values of the depth distance. The light blue are the detections referred to the right person and the orange ones are referred to the left person. This is the reason of the impossibility of knowing the number of people present in the frame.

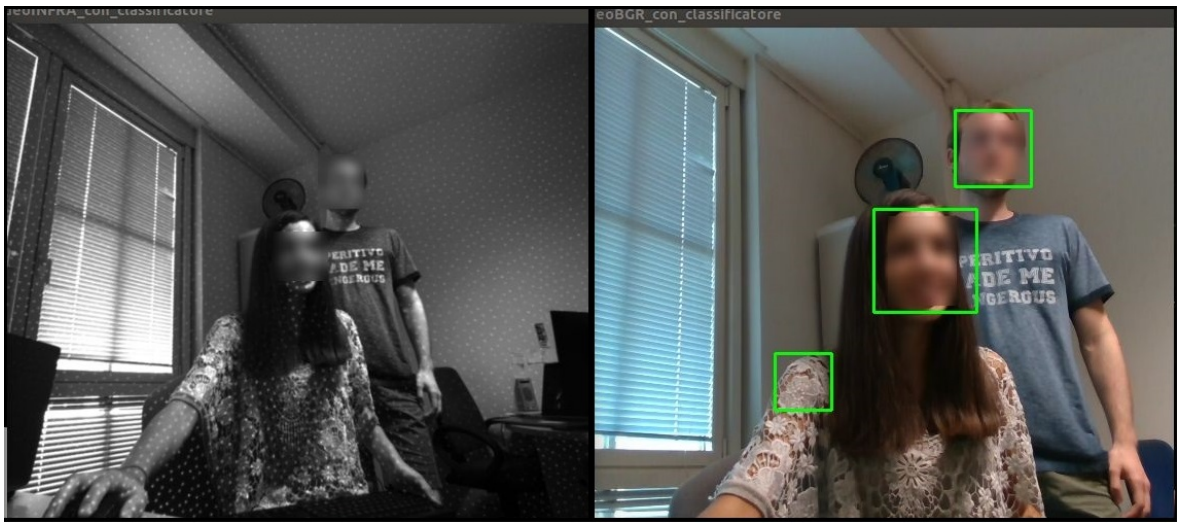


FIGURE 8.5: Haarcascade classifier: example of false positive.

8.2 Y.O.L.O.

Three types of Y.O.L.O. are used in this project:

1. Y.O.L.O.v2;
2. Tiny-Y.O.L.O.v3;
3. Y.O.L.O.v3.

8.2.1 Results post re-training

AP and mAP

As it has been anticipated in the §6.1.2 the three Y.O.L.O. networks used are re-trained for just the class person and in the following table 8.1 is shown the mAP and AP results of this operation (see §4.1.2 and §4.1.2):

Architecture	**Dataset**			
	COCO		Person	
	AP	mAP	AP	mAP
Y.O.L.O.v2	56.81%	23.30%	54.65%	16.24%
Tiny-Y.O.L.O.v3	19.21%	5.16%	49.30%	8.17%
Y.O.L.O.v3	69.11%	28.99%	68.64%	28.46%

Table 8.1: AP and mAP of each network related to COCO and Person datasets.

From the table it is possible to see that only the smallest network (Tiny-Y.O.L.O.v3) has a good improvement on the AP and mAP. This happens because the "big" networks are just too robust and give great result also before the re-training. It is possible to highlight decreasing results on both the Y.O.L.O.v2 and Y.O.L.O.v3 networks that can be easily justify. In fact, the networks trained for a huge number of classes are able to associate all the other features or edge in the image to something, so the model is very robust. With the re-training this capacity can be partially lost and this is why the AP and mAP suffer a small percentage reduction in Y.O.L.O.v3 and a more substantial one in Y.O.L.O.v2. Anyway, the re-training is important because reduces the processing image time with a consequent rising up of the FPS for the real-time applications.

FPS improvements

Testing the networks on the Jetson Xavier Developer Kit it is possible to obtain the following FPS results (table 8.2):

Architecture	Frames per second
Y.O.L.O.v2	7.3 - 9.0 FPS
Tiny-Y.O.L.O.v3	18.0 - 27.0 FPS
Y.O.L.O.v3	3.5 - 5.0 FPS

Table 8.2: FPS obtained by each network.

These results affect the velocity of the detection algorithm and consequently make the person tracking application real-time.

Precision, Recall, F1-score and average IoU

Other parameters are extrapolated from the re-training of the networks: precision, recall, F1-score and average IoU (see §4.1.2 and §4.1.1) and are shown in the following tables: 8.3 and 8.4.

Parameters	Y.O.L.O.v2		Tiny-Y.O.L.O.v3		Y.O.L.O.v3	
	COCO	Person	COCO	Person	COCO	Person
Precision	0.41	0.57	0.30	0.67	0.44	0.79
Recall	0.54	0.58	0.22	0.45	0.65	0.63
F1-score	0.47	0.57	0.25	0.54	0.53	0.70
average IoU	30.76%	41.32%	21.98%	47.80%	30.04%	60.44%

Table 8.3: Re-training parameters related to IoU=50%.

Parameters	Y.O.L.O.v2		Tiny-Y.O.L.O.v3		Y.O.L.O.v3	
	COCO	Person	COCO	Person	COCO	Person
Precision	0.23	0.24	0.15	0.24	0.26	0.46
Recall	0.30	0.24	0.11	0.16	0.38	0.36
F1-score	0.26	0.24	0.13	0.19	0.31	0.40
average IoU	18.99%	20.29%	12.43%	19.99%	21.71%	38.52%

Table 8.4: Re-training parameters related to IoU=75%.

These results highlight the importance of the re-training operation because there is an improvement on each parameter.

8.2.2 Results obtained from Jaffle tests

The three networks are tested previously in Gazebo and then directly on the Jaffle and the results are the following:

- Y.O.L.O.v2:
  The person detection algorithm is fast, but not performant for one main reason: there is a huge problem with the false positives (FP), as it is shown in figure 8.9. For this reason the movements of the robot are mostly wrong because several times the network detects the correct person and a false positive, so the Jaffle stops, according to the control algorithm. This makes this network unusable for person tracking application.

  Here are reported the results obtained from this network, taking into account the legend in the table 8.5.

Legend	
yellow	Detection information.
green	Class detected in the frame with its percentage and the FPS of the network.
red	Number of objects detected.
blu	Bounding box coordinates.
pink	Velocity commands given to the robot.

TABLE 8.5: Legend required for a correct interpretation of the results of the Y.O.L.O. networks.

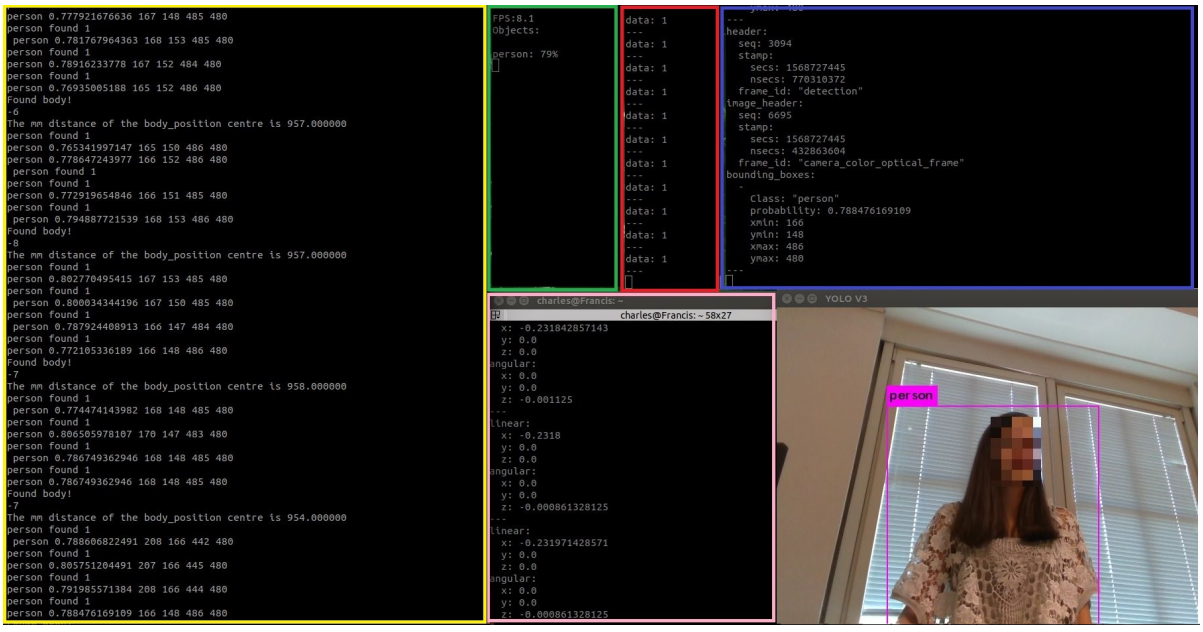


FIGURE 8.6: Y.O.L.O.v2: one person detected (see the table 8.5).

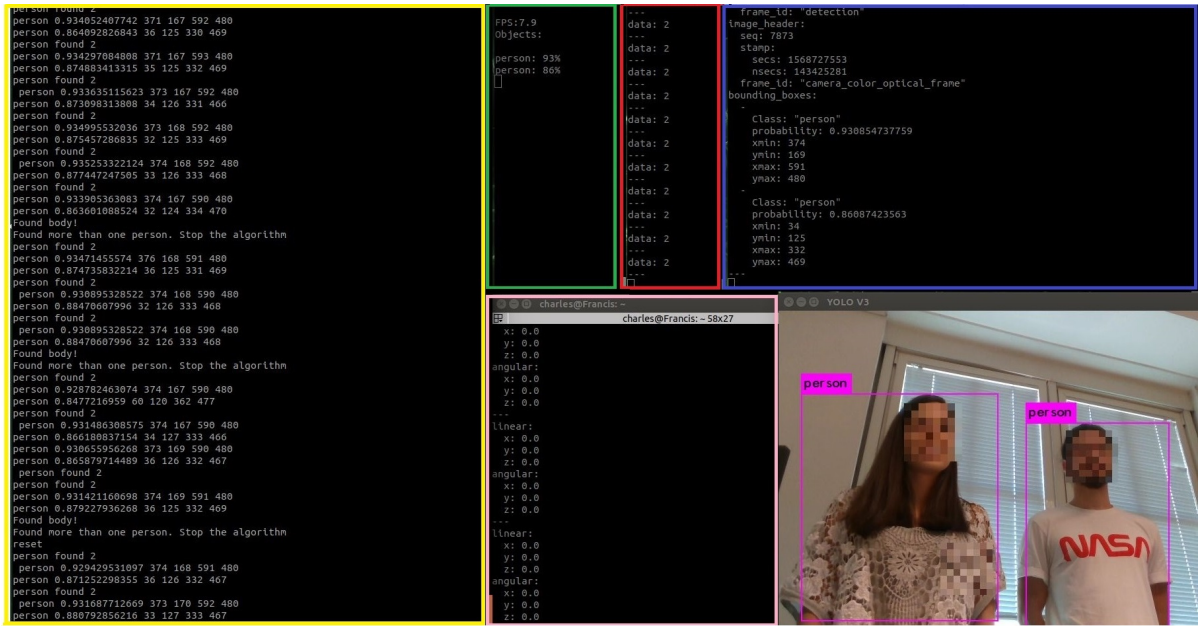


FIGURE 8.7: Y.O.L.O.v2: two people detected (see the table 8.5).

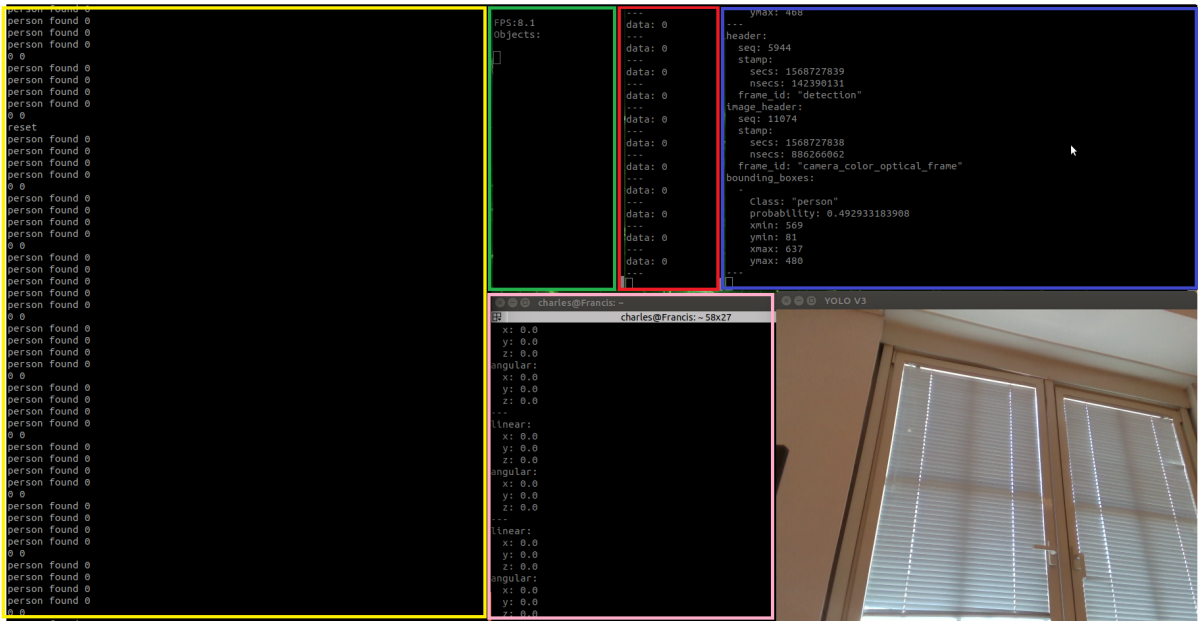


FIGURE 8.8: Y.O.L.O.v2: nothing detected (see the table 8.5).

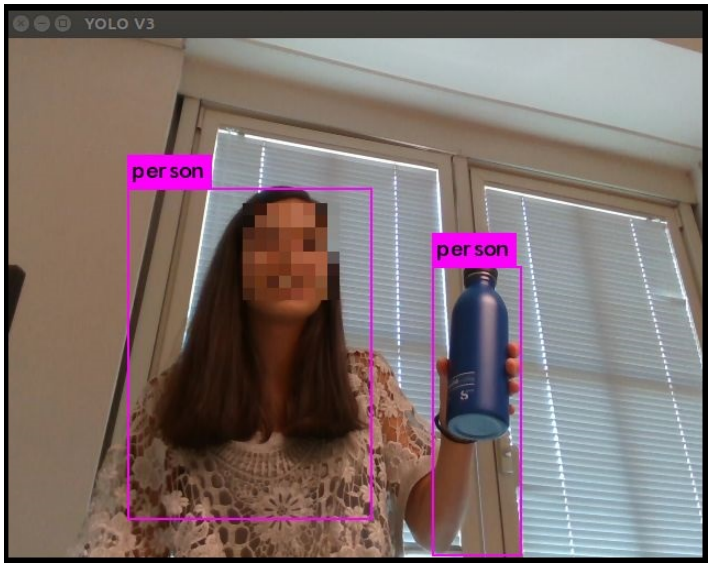


FIGURE 8.9: Y.O.L.O.v2: false positive detection.

- Tiny-Y.O.L.O.v3:
  The person detection algorithm is extremely fast and performant. The improvement given to the network by the re-training is high, so the person tracking is done with optimal real-time results and a perfect control of the movements.

  Here are reported the results obtained from this network, taking into account the legend in the table 8.5.

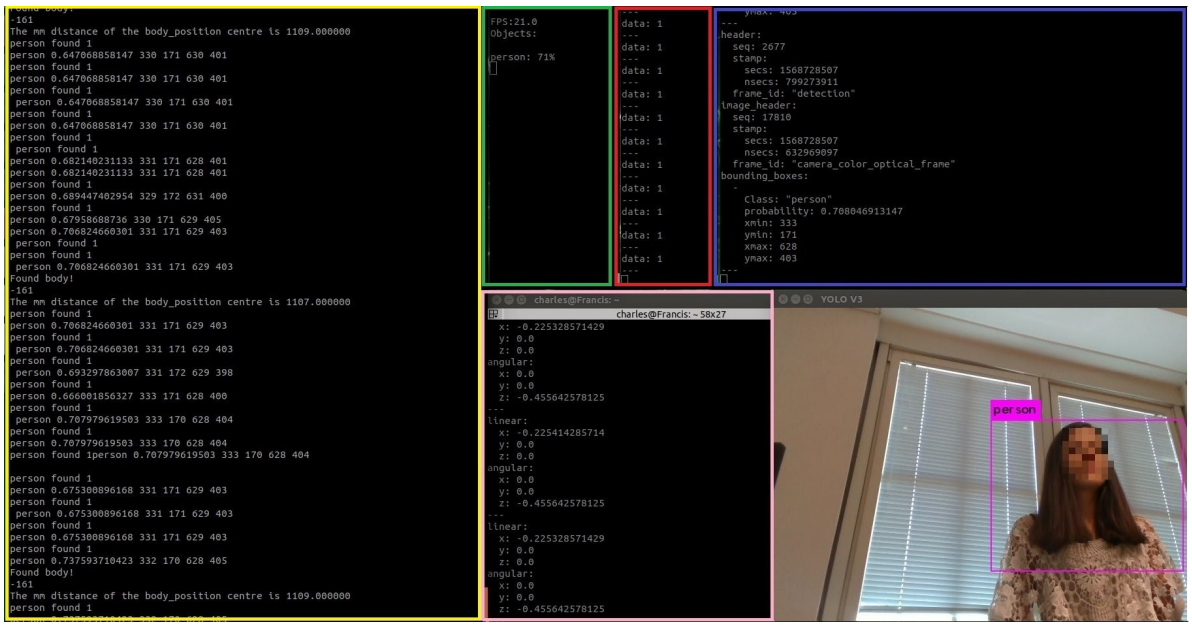


FIGURE 8.10: Tiny-Y.O.L.O.v3: one person detected (see the table 8.5).

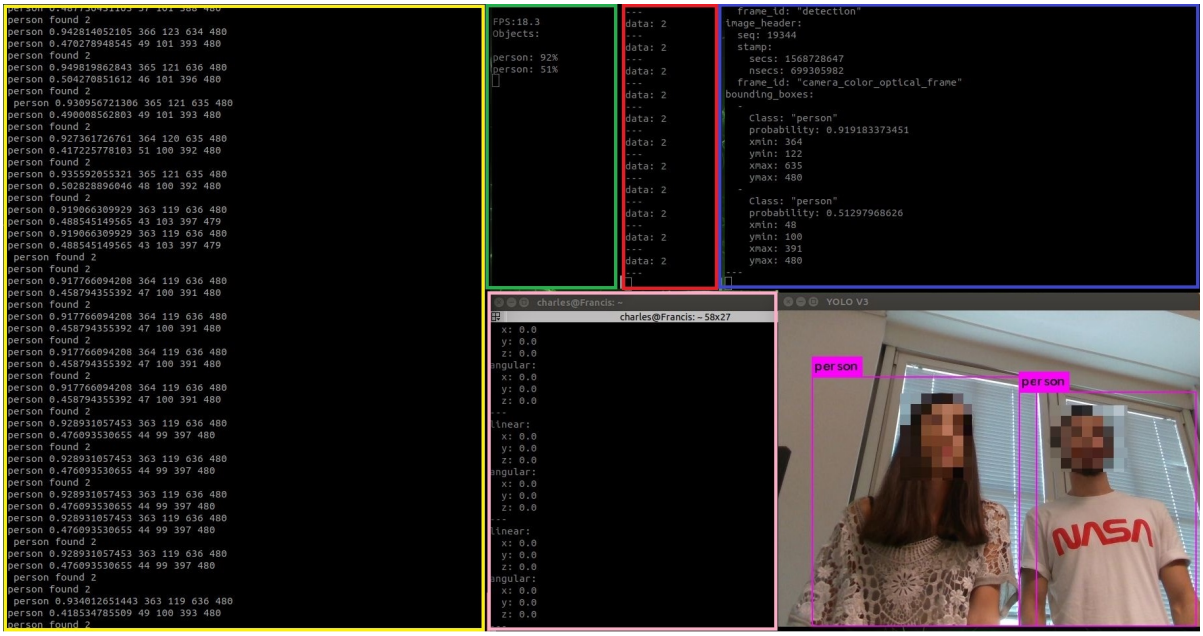


FIGURE 8.11: Tiny-Y.O.L.O.v3: two people detected (see the table 8.5).

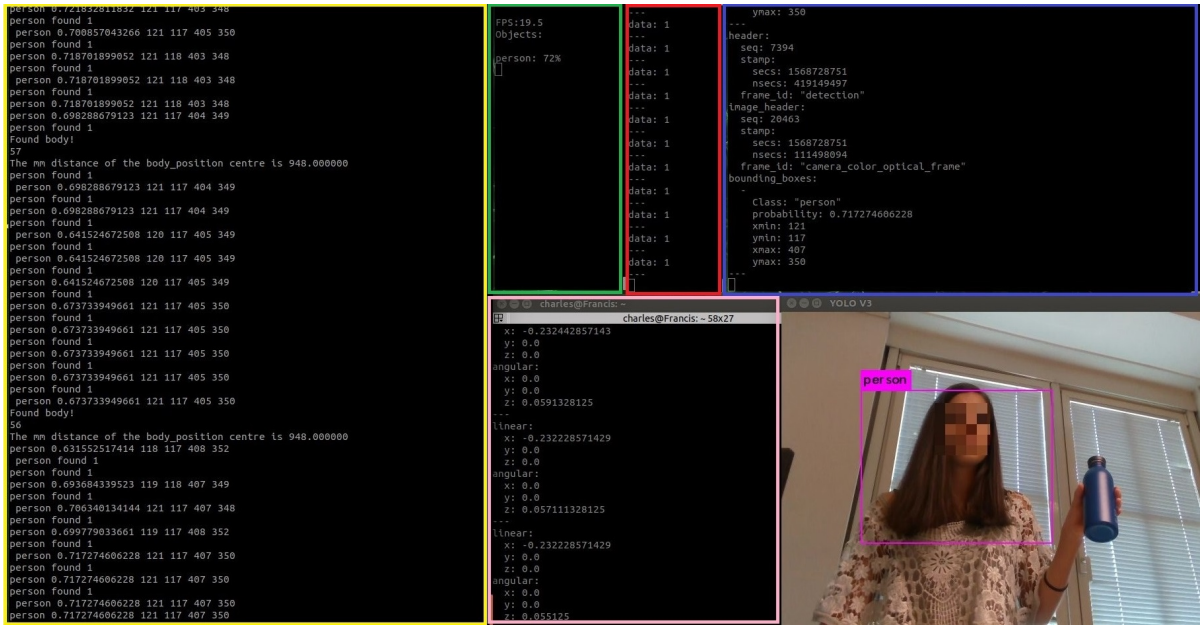


FIGURE 8.12: Tiny-Y.O.L.O.v3: no false positive (see the table 8.5).

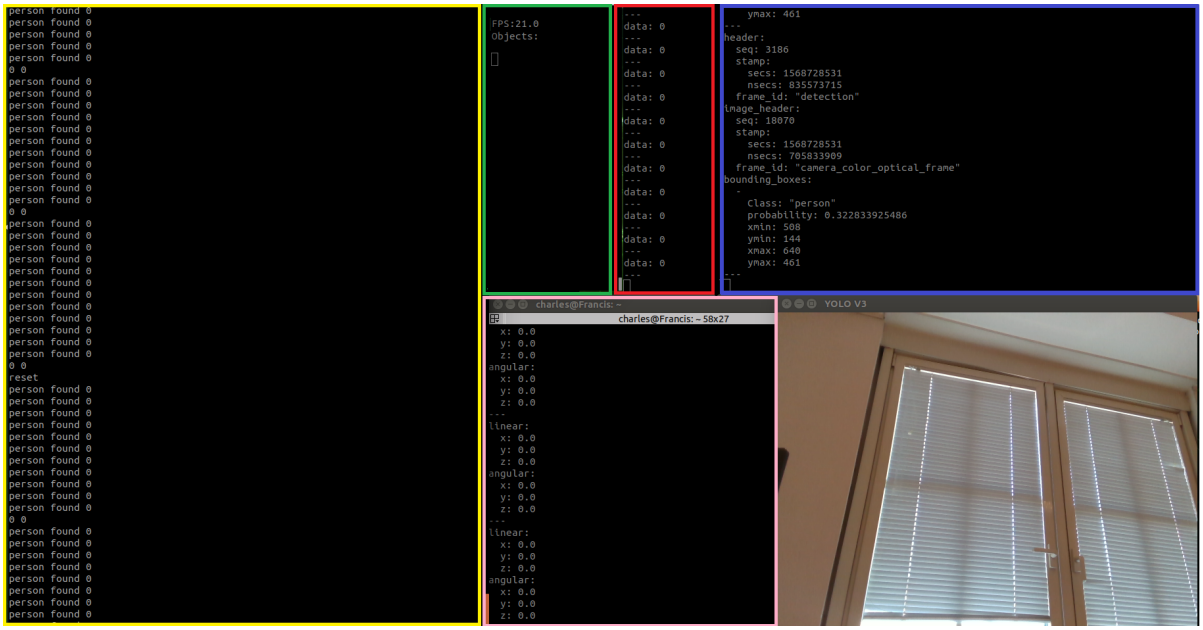


FIGURE 8.13: Tiny-Y.O.L.O.v3: nothing detected (see the table 8.5).

- Y.O.L.O.v3:
  The person detection algorithm in not so fast because the network is very complex and does heavy computations, differently from the other two networks, however the results are optimal. The movement of the robot is fluent and coherent with the displacement of the person, so it is not affected from the not so high FPS.

  Here are reported the results obtained from this network, taking into account the legend in the table 8.5.

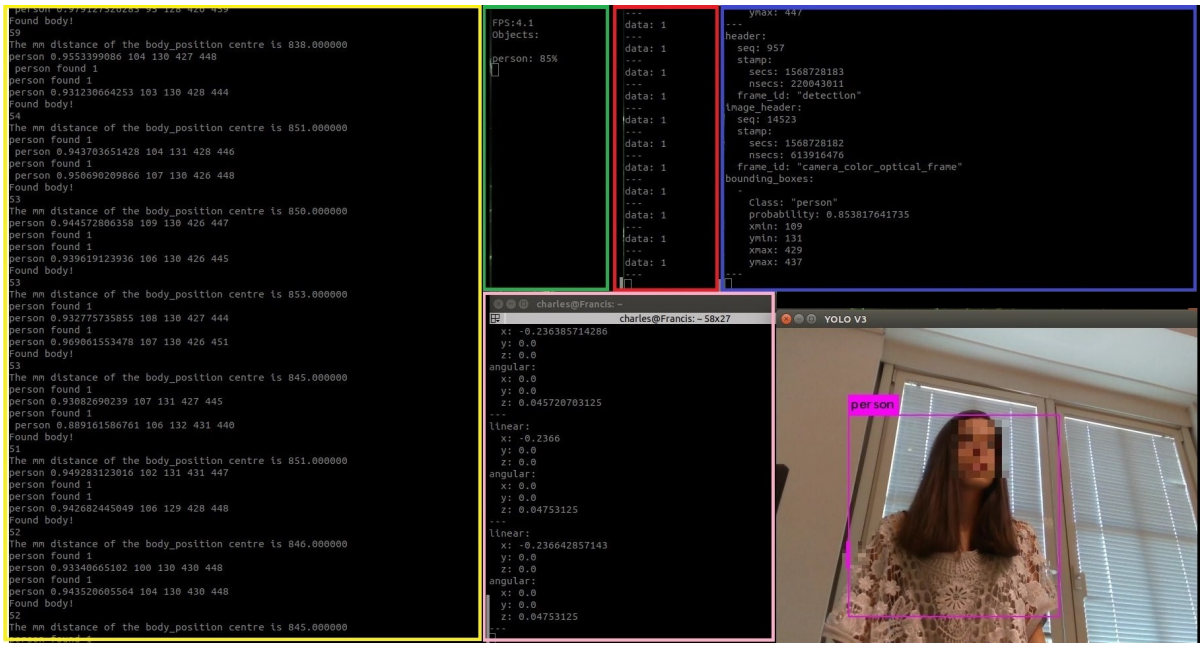


FIGURE 8.14: Y.O.L.O.v3: one person detected (see the table 8.5).

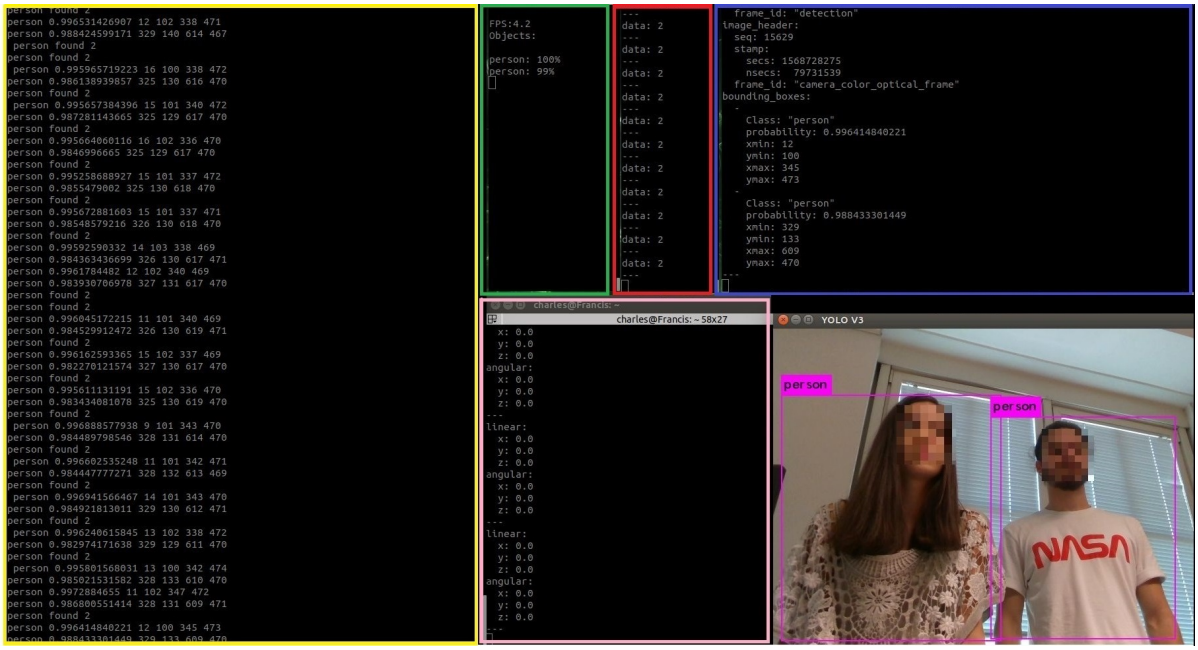


FIGURE 8.15: Y.O.L.O.v3: two people detected (see the table 8.5).

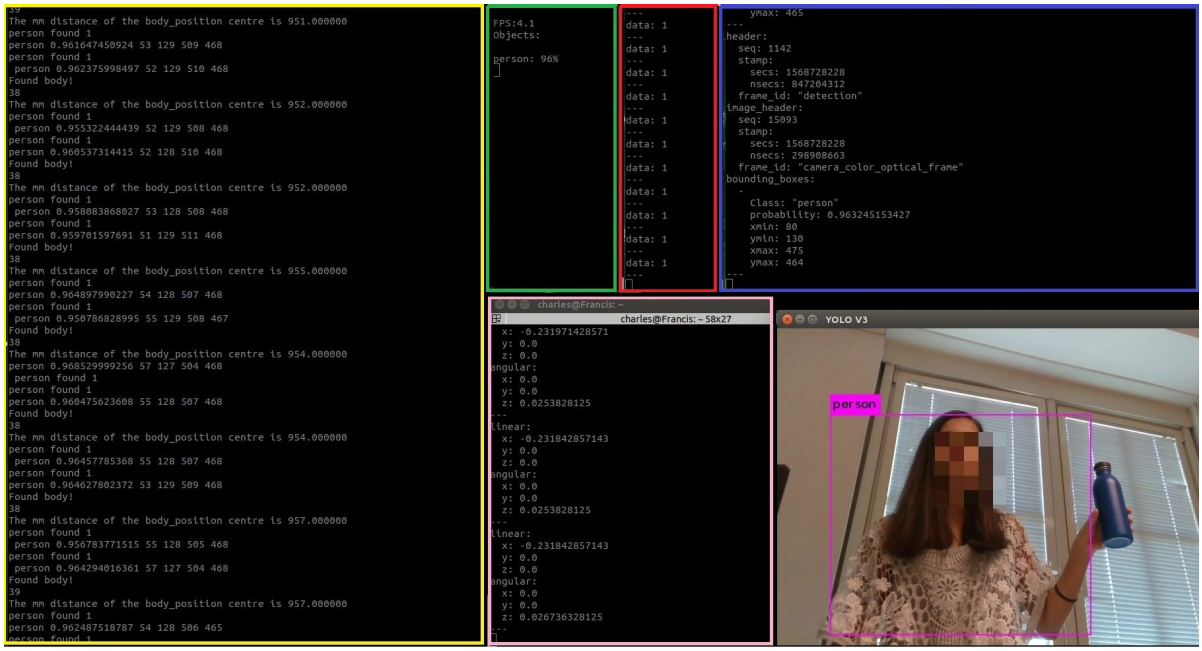


FIGURE 8.16: Y.O.L.O.v3: no false positive (see the table 8.5).

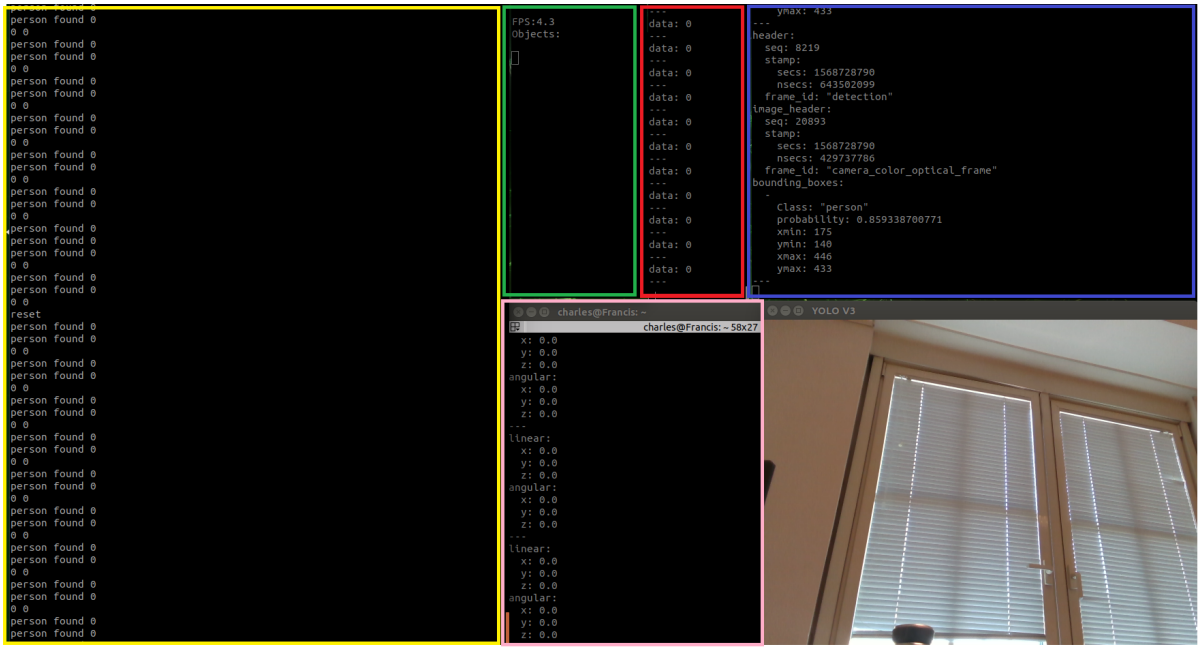


FIGURE 8.17: Y.O.L.O.v3: nothing detected (see the table 8.5).

The results obtained highlight how the Deep Neural Network algorithms are more robust respect to the Machine Learning classical ones. In Particular, the behaviours of the Tiny-Y.O.L.O.v3 and Y.O.L.O.v3 are coherent with the requests of the thesis project and the best one to use is Tiny-Y.O.L.O.v3 because reaches the same results with higher FPS, so best performance with lower computational complexity.

8.3 Obstacle Avoidance

The tests of the obstacle avoidance implementation have been made in two different steps:

1. It has been tested the correctness of the detection of the goal-person respect to the `\map` RF using RViz and the robot stopped in order to evaluate the correct update of the goal coherently to the displacement of the person;

2. It has been tested the obstacle avoidance algorithm in the indoor environment.

The obtained results are positives, the goal is updated correctly during the displacement of the person and the movement control respects the three detection situations explained in §7.2.2. The algorithm works exactly as expected, however it is not an optimal solution. In fact, this is a basic obstacle avoidance implementation, so it is possible to see that the trajectory is not optimized compared to the trajectory execution time and the obstacles are not separate from the compliance with kinodynamic constraints, as it is done in more robust algorithms. For these reasons, even if the tests have consistent results with those expected, this solution is not suitable for the person tracking project, but needs to be improved.

Here are reported the results obtained by integrating this algorithm of obstacle avoidance with the Tiny-Y.O.L.O.v3 network, taking into account the legend in the table 8.6.

Legend	
light blue	Configuration of the robot.
yellow	Detection information.
green	Class detected in the frame with its percentage and the FPS of the network.
red	RViz representation of the goal depending on the displacement of the person.
blu	Velocity commands given to the robot.
white	Odometry of the robot.

Table 8.6: Legend required for a correct interpretation of the results of the Obstacle Avoidance integration.

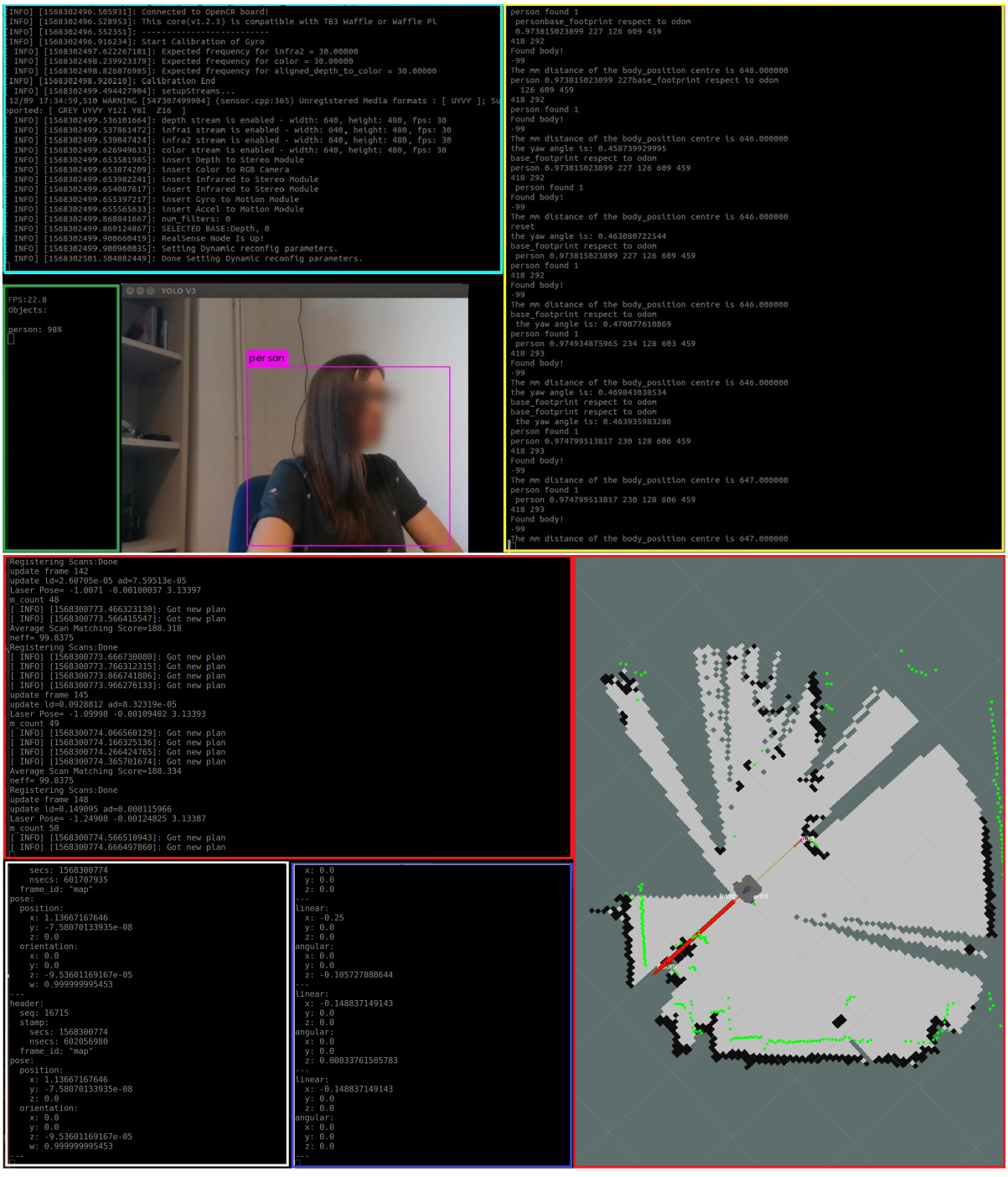


FIGURE 8.18: Activation of the LiDAR, detection of the person, identification of the goal, velocity commands of the robot (see the table 8.6).

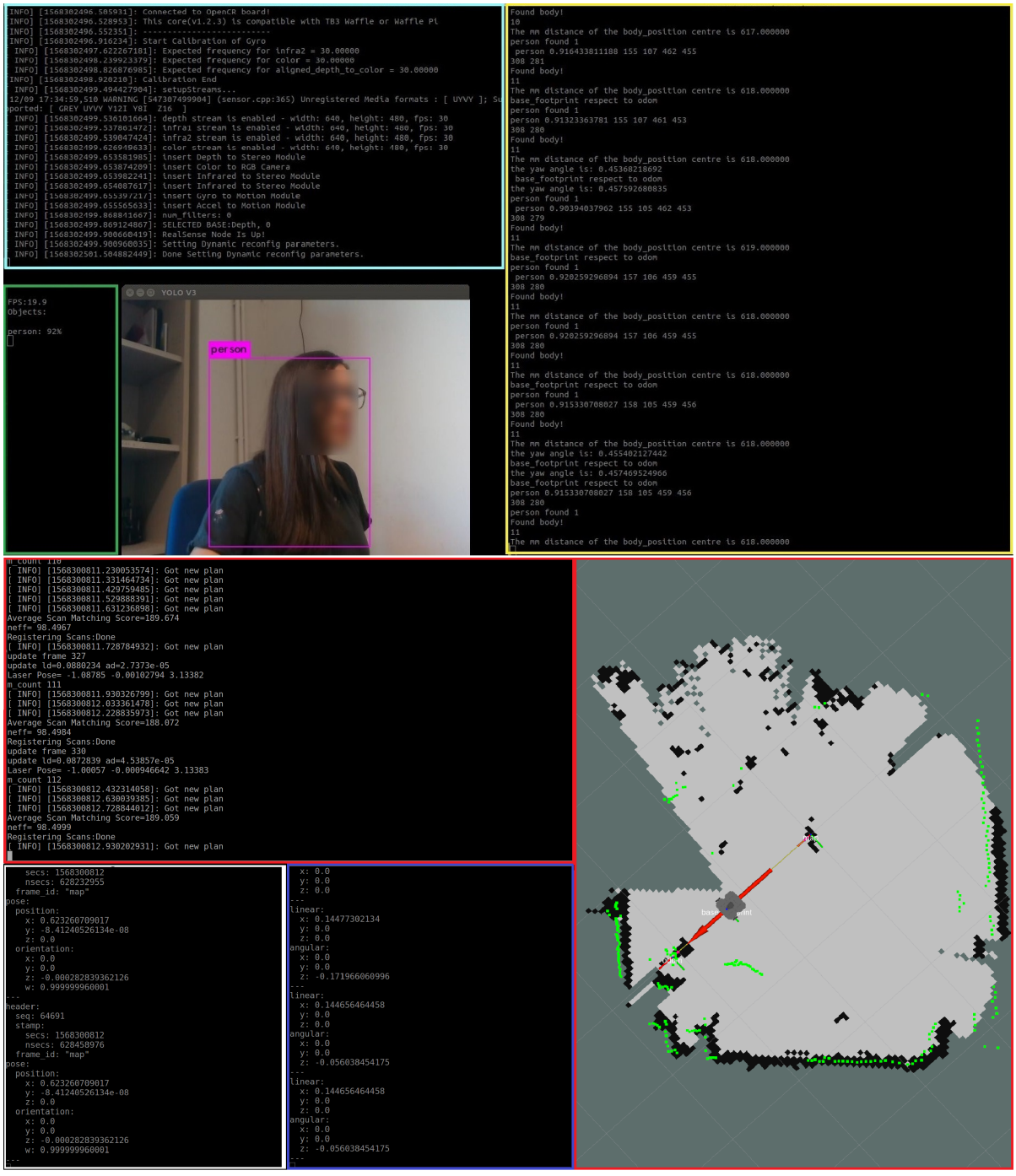


FIGURE 8.19: The goal pose is changed and consequently the velocity commands of the robot (see the table 8.6).

8.4 Conclusions and Future Works

The obtained results satisfy widely the requests of the thesis and are considered the first steps of a broad project in which many other service tasks will be integrated.

Next steps can be:

- find the more suitable and optimized obstacle avoidance algorithm which could be effectively integrated into the person tracking algorithm or use DeepQ Learning algorithms to reinforce the autonomous navigation;
- add other tasks for monitoring elderly people as for example a motion prediction algorithm with an integration of alarm messages in case of a domestic accident, such as a fall;
- try to reduce the number of sensors used for the same application, optimizing them, in anticipation of the realization of a less expensive marketable product.

Bibliography

[1] Oussama Khatib Bruno Siciliano. *Springer Handbook of Robotics*. 2nd. Springer-Verlag Berlin Heidelberg, 2016.

[2] *Intro to Deep Learning for Computer Vision*. URL: https://chaosmail.github.io/deeplearning/2016/10/22/intro-to-deep-learning-for-computer-vision/.

[3] *Going deep into object detection*. URL: https://towardsdatascience.com/going-deep-into-object-detection-bed442d92b34.

[4] C.P. Papageorgiou, Michael Oren, and Tomaso Poggio. "General framework for object detection". In: vol. 6: Feb. 1998, pp. 555–562. ISBN: 81-7319-221-9. DOI: 10.1109/ICCV.1998.710772.

[5] *Object Detection for Dummies Part 1: Gradient Vector, HOG, and SS*. URL: https://lilianweng.github.io/lil-log/2017/10/29/object-recognition-for-dummies-part-1.html.

[6] Paul Viola and Michael Jones. "Rapid object detection using a boosted cascade of simple features". In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. Vol. 1. IEEE. 2001, pp. I–I.

[7] *Haar Classifier in Face Detection*. URL: https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html.

[8] Paul Viola and Michael Jones. "Robust Real-Time Object Detection". In: vol. 57. Jan. 2001.

[9] Zhihui Wang et al. "A High Accuracy Pedestrian Detection System Combining a Cascade AdaBoost Detector and Random Vector Functional-Link Net". In: *TheScientificWorldJournal* 2014 (May 2014), p. 105089. DOI: 10.1155/2014/105089.

[10] Navneet Dalal and Bill Triggs. "Histograms of oriented gradients for human detection". In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. IEEE. 2005, pp. 886–893.

[11] *Dlib 18.6 released: Make your own object detector!* URL: http://blog.dlib.net/2014/02/dlib-186-released-make-your-own-object.html.

[12] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 1st. O'Reilly Media, Inc., 2017. ISBN: 1491962291, 9781491962299.

[13] *What do we learn from region based object detectors (Faster R-CNN, R-FCN, FPN)?* URL: https://medium.com/@jonathan_hui/what-do-we-learn-from-region-based-object-detectors-faster-r-cnn-r-fcn-fpn-7e354377a7c9.

[14] Jasper Uijlings et al. "Selective Search for Object Recognition". In: *International Journal of Computer Vision* 104 (Sept. 2013), pp. 154–171. DOI: 10.1007/s11263-013-0620-5.

[15] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[16] Ross Girshick. "Fast r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[17] Shaoqing Ren et al. "Faster R-CNN: towards real-time object detection with region proposal networks". In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 6 (2017), pp. 1137–1149.

[18] Wei Liu et al. "Ssd: Single shot multibox detector". In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

[19] *Understanding SSD MultiBox — Real-Time Object Detection In Deep Learning*. URL: https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab.

[20] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[21] Joseph Redmon and Ali Farhadi. "YOLO9000: better, faster, stronger". In: *arXiv preprint* (2017).

[22] *Batch normalization in Neural Networks*. URL: https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c.

[23] *Real-time object detection with YOLO*. URL: https://machinethink.net/blog/object-detection-with-yolo/.

[24] *K Means Clustering : Identifying F.R.I.E.N.D.S in the World of Strangers*. URL: https://towardsdatascience.com/k-means-clustering-identifying-f-r-i-e-n-d-s-in-the-world-of-strangers-695537505d.

[25] Joseph Redmon and Ali Farhadi. "Yolov3: An incremental improvement". In: *arXiv preprint arXiv:1804.02767* (2018).

[26] *Intersection over Union (IoU) for object detection*. URL: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

[27] *A Beginner's Guide to Object Detection*. URL: https://www.datacamp.com/community/tutorials/object-detection-guide.

[28] *Section 7 Advanced Evaluation Metrics*. URL: http://cs230.stanford.edu/section/7/.

[29] *mAP (mean Average Precision)*. URL: https://github.com/Cartucho/mAP.

[30] Brian Gerkey & William D. Smart Morgan Quigley. *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*. O'Reilly Media, 2015.

[31] *ROBOTIS*. URL: http://wiki.ros.org/action/show/robotis?action=show&redirect=ROBOTIS.

[32] . URL: http://xiaoyatec.com/wp-content/uploads/2017/07/opencr.png.

[33] *Hardware Setup*. URL: http://emanual.robotis.com/docs/en/platform/turtlebot3/hardware_setup/.

[34] *Intel® RealSense™ Depth Camera D435i*. URL: https://www.intelrealsense.com/depth-camera-d435i/.

[35] *Intel® RealSenseTM D400 Series Product Family*. URL: https://www.intelrealsense.com/wp-content/uploads/2019/07/Intel-RealSense-D400-Series-Datasheet-Jun-2019.pdf?_ga=2.73594705.237534047.1567067222-646256078.1557041861.

[36] . URL: https://devtalk.nvidia.com/default/topic/1036207/nvidia-announces-jetson-xavier/?offset=4.

[37] *Jetson AGX Xavier Developer Kit*. URL: https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit.

[38] *Overview*. URL: https://github.com/IntelRealSense/librealsense.

[39] *ROS Wrapper for Intel® RealSense™ Devices*. URL: https://github.com/IntelRealSense/realsense-ros.

[40] *YOLO ROS: Real-Time Object Detection for ROS*. URL: https://github.com/leggedrobotics/darknet_ros.

[41] *TurtleBot3*. URL: https://github.com/ROBOTIS-GIT/turtlebot3.

[42] Jason Yosinski et al. "How transferable are features in deep neural networks?" In: *CoRR* abs/1411.1792 (2014). arXiv: 1411.1792. URL: http://arxiv.org/abs/1411.1792.

[43] *Yolo-v3 and Yolo-v2 for Windows and Linux*. URL: https://github.com/AlexeyAB/darknet?files=1#yolo-v3-and-yolo-v2-for-windows-and-linux.

[44] Ivan Krasin et al. "OpenImages: A public dataset for large-scale multi-label and multi-class image classification." In: (2017). URL: https://storage.googleapis.com/openimages/web/index.html.

[45] Angelo Vittorio. *Toolkit to download and visualize single or multiple classes from the huge Open Images v4 dataset*. `https://github.com/EscVM/OIDv4_ToolKit`. 2018.

[46] *move_base*. URL: `http://wiki.ros.org/move_base`.