Politecnico di Torino

Master's Degree in Mechatronic Engineering

Master's Degree Thesis

DEEP COMPUTER VISION FOR HUMAN-ROBOT INTERACTION AVAILABILITY EVALUATION

Supervisors

Prof. Marcello CHIABERGE

Dott. Simone ANGARANO

Dott. Mauro MARTINI

Dott. Chiara BORETTI

Candidate

Carlo SIMONE

December 2024

Abstract

The introduction of social robots in human-populated environments generates many different challenges to study for a safe and robust collaboration. To support the humans in different collaborative tasks, an autonomous robot should be able to recognize where target objects and persons are and, nonetheless, whether any person is available for interaction. The application of Deep Learning methods in the field of Computer Vision led to relevant improvements in object detection, image segmentation and other meaningful perception tasks.

The aim of this thesis is to make use of state-of-the-art convolutional neural networks to build a real-time software system that allows a robot to recognize humans and objects in real-time using a simple RGBD camera, and exploit this information to estimate if the detected humans are available for human-robot interaction based on a scoring system.

The system makes use of two neural networks to acquire human poses and object pixel masks. Each person is tracked in the video stream with a unique ID, and objects are associated to persons according to the relative distance and the persons' gaze direction. After data processing, a score is computed for each person detected taking into account position, gaze direction and interaction with objects in the latest camera frames. The robot, basing on these scores, may choose which is the best human to interact with and what action to perform.

Tests carried out in diverse indoor situations in an office environment, showed that the proposed method gives promising and coherent results, even though there is room for improvement.

Acknowledgements

This work wouldn't have been possible without the support and patience of Simone, Mauro, Chiara and Andrea, my supervisors at Pic4Ser, who made sure that I kept going on even in the hardest times.

Table of Contents

List of Tables VI

List of Figures VII

Acronyms X

1 Introduction 1

2 State of the art 3

2.1 Human pose estimation and action detection 3

2.2 Human gaze estimation 5

3 Theoretical fundamentals 8

3.1 Stereoscopic vision 8

3.2 Kalman filters 10

3.3 Machine Learning regression 13

3.3.1 Regression models 13

3.4 Neural networks 17

3.4.1 Neurons and layers 18

3.4.2 Neural network structure 20

3.4.3 Neural network types 21
3.4.4 Convolutional neural networks for computer vision 24

4 Attention estimation system 26

4.1 Scope of the system 26
4.1.1 Human distraction and interest factors 26
4.2 Methods and algorithms used 28
4.2.1 Foreword: the global coordinate frame 28
4.2.2 Ultralytics YOLO neural networks 28
4.2.3 Simple Online and Realtime Tracking 32
4.2.4 Gaze estimation 36
4.2.5 Gaze evaluation 49
4.3 System workflow 51
4.3.1 RGBD frame acquisition 52
4.3.2 Instance segmentation and pose estimation 53
4.3.3 Data production and gaze estimation 55
4.3.4 SORT update 55
4.3.5 Persons database update 56
4.3.6 Non-present persons' update 61
4.3.7 Inactive persons' deletion 61
4.3.8 Association of objects to persons 62
4.3.9 Persons' scores computation 68
4.3.10 Graphical outputs 71

5 System implementation 76

5.1 Hardware devices used 76
5.1.1 Intel® NUC 76

5.1.2 Intel® RealSense™ d435 77
5.2 Software libraries used 78
5.2.1 NumPy 78
5.2.2 SORT 78
5.2.3 OpenCV 78
5.2.4 Ultralytics YOLOv8 79
5.2.5 TensorFlow and Keras 79
5.2.6 `matplotlib` 79

6 System testing 80
6.1 Samples from tests 81
6.1.1 Test 1: elevator room 81
6.1.2 Test 2: crowded waiting room 83
6.1.3 Test 3: hallway 85
6.1.4 Test 4: office 87
6.2 Results 88

7 Conclusion and future work 90

Bibliography 91

List of Tables

3.1 Symbols used in Kalman predictor/corrector equations 11
3.2 Common activation functions 19
3.3 Computer vision tasks 25

4.1 Recap of the tested ML gaze estimation methods 48
4.2 Object classes selected for instance segmentation 54
4.3 `Object` class attributes 57
4.4 `Person` class exclusive attributes 58

List of Figures

2.1 Head, arms and legs' bounding boxes according to Chakraborty et al.'s paper 4

2.2 Action classes in the UCF50 dataset 5

2.3 Xia et al.'s experimental setup 6

2.4 Geometrical data used for yaw and pitch angles' calculations 7

3.1 Computation of 3D coordinates of a point using stereoscopic vision 9

3.2 Schematic view of a Kalman predictor-corrector. The top half $\mathcal{S}$ represents the actual system, while the bottom half $\mathcal{K}_{pc}$ is the predictor-corrector. 12

3.3 Example of linear regression on 1D data 14

3.4 Example of polynomial (cubic) regression on 1D data 15

3.5 Example of application of binary decision tree classifier on a set of 2D data 16

3.6 Random forest conceptual scheme 17

3.7 Example of a fully connected feedforward neural network structure . 22

3.8 Possible pooling methods on a 5×5 matrix with window size 3 and stride 2 23

3.9 Structure of a shallow CNN 24

4.1 Result of instance segmentation with YOLOv8 30
4.2 YOLOv8 pose estimation keypoints 31
4.3 COCO-WholeBody keypoints 32
4.4 First choice of input data 40
4.5 Second choice of input data 41
4.6 Plot of symmetric logarithm function 44
4.7 Plot of symmetric exponential function 45
4.8 Plot of scalar product loss function 48
4.9 Plot of score multiplier due to distance from the camera. 71
4.10 `cv2 Jet` color map 73
4.11 RGB and depth frame 73
4.12 Example of 3D scene reconstruction 74
4.13 Example of score graphs plotting 75

6.1 Frame taken from test 1 82
6.2 Frame taken from test 2 84
6.3 Frame taken from test 3 86
6.4 Frame taken from test 4 87

Acronyms

AI

artificial intelligence

CNN

Convolutional Neural Network

DL

Deep Learning

ELU

Exponential Linear Unit

LR

Linear Regression

IoU

Intersection over Union

MAE

Maximum Absolute Error

ML

Machine Learning

MSE

Mean Square Error

NLR

NonLinear Regression

NN

Neural Network

ReLU

Rectified Linear Unit

RF

Random Forest

SORT

Simple Online Realtime Tracking

Chapter 1

Introduction

The constantly growing presence and skill level of automated machines of any kind, including robots, is a huge opportunity for the progress of humanity. Robots are often able to execute physical tasks faster and more efficiently than humans, and the range of scenarios in which they can operate grows wider everyday. There are, though, a few relevant limitations that all robots must face:

- most robots still require support from humans for logistical aspects such as maintenance, battery substitution, and so on;
- service robots are specifically designed to operate alongside humans, or to assist them in some way.

For these reasons, it is crucial for robots to recognize humans, their activities and their interactions with themselves, the objects around them and the environment, and to infer when and in which situations it is possible to request human intervention without causing trouble.

One of the most powerful techniques that a robot can use to explore and "understand" the sorrounding environment is Computer Vision, a class of software

methods aimed at processing images and videos and extracting useful features and information from them. Many Computer Vision methods rely on Machine Learning or Deep Learning techniques and tools, such as Convolutional Neural Networks, complex computational structures able to self-teach how to isolate and detect patterns in images and convert them in useful data.

Depth cameras are a valiant ally for any Computer Vision system. They allow the system to process three-dimensional data without the struggle that would be needed to infer the distance of objects from ordinary images.

Most robots, though, have to deal with constraints that represent limiting factors for their possibility to use advanced software solutions. The most modern and powerful Computer Vision neural networks require an amount of computational resources that not every robot can afford. Other key limitation may be represented by the battery consumption, that has to be kept within reasonable bounds, and the frame rate, that cannot be too low.

The aim of this thesis work is to develop a system that a small and not particularly powerful moving robot, in need of human intervention to perform a task, can use to recognize objects and persons in the sorrounding environment and, basing on their location and attitude, determine which person is the most suitable for the task that has to be performed.

Chapter 2

State of the art

2.1 Human pose estimation and action detection

One of the key tasks that the system has to perform is to recognize humans in the scene and to understand what they are doing, at least to some extent. This task is not trivial at all, because pose estimation itself requires complex computations, and associating poses to actions is even harder beacuse of the many different body poses that may correspond to the same actions and viceversa. Many different approaches were tried.

Chakraborty et al.[1]'s method tries to detect the poses of head, legs and arms by checking bounding boxes of fixed sizes, combining them together, and trying to train a model similar to Support Vector Machines to recognize 6 human actions: walking, jogging, running, boxing, clapping or waving hands. Although the approach is

[1]Bhaskar Chakraborty et al. «View-invariant human-body detection with extension to human action recognition using component-wise HMM of body parts». In: *2008 8th IEEE International Conference on Automatic Face & Gesture Recognition*. 2008, pp. 1–6. DOI: 10.1109/AFGR.2008.4813302.

interesting, the choice of actions makes it unsuitable for this project.

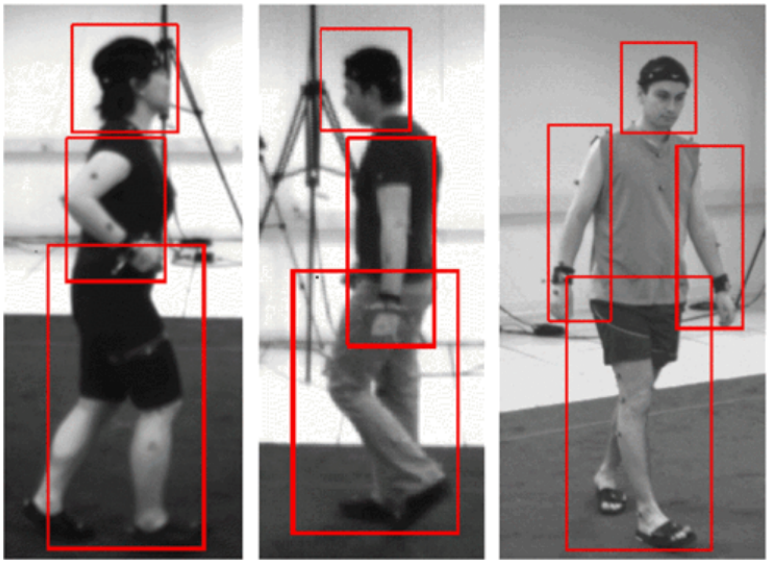


Figure 2.1: Head, arms and legs' bounding boxes according to Chakraborty et al.'s paper

Reddy and Shah[2], instead, used the UCF50 dataset, a selection of more than 6,000 videos of persons executing 50 different actions, shown in Figure 2.2, and developed a classifier that takes into account both motion features (i.e. the evolution of the (x, y, t) 3D cuboids related to some features) and a context descriptor that takes into account key frames' evolutions. This approach, though, presents two issues: the need of building an appropriate dataset for the actions actually needed for scopes different from the original paper's, and the fact that the method cannot be used in real time.

[2]Kishore K. Reddy and Mubarak Shah. «Recognizing 50 human action categories of web videos». In: *Machine Vision and Applications* 24.5 (July 2013), pp. 971–981. ISSN: 1432-1769. DOI: 10.1007/s00138-012-0450-4.

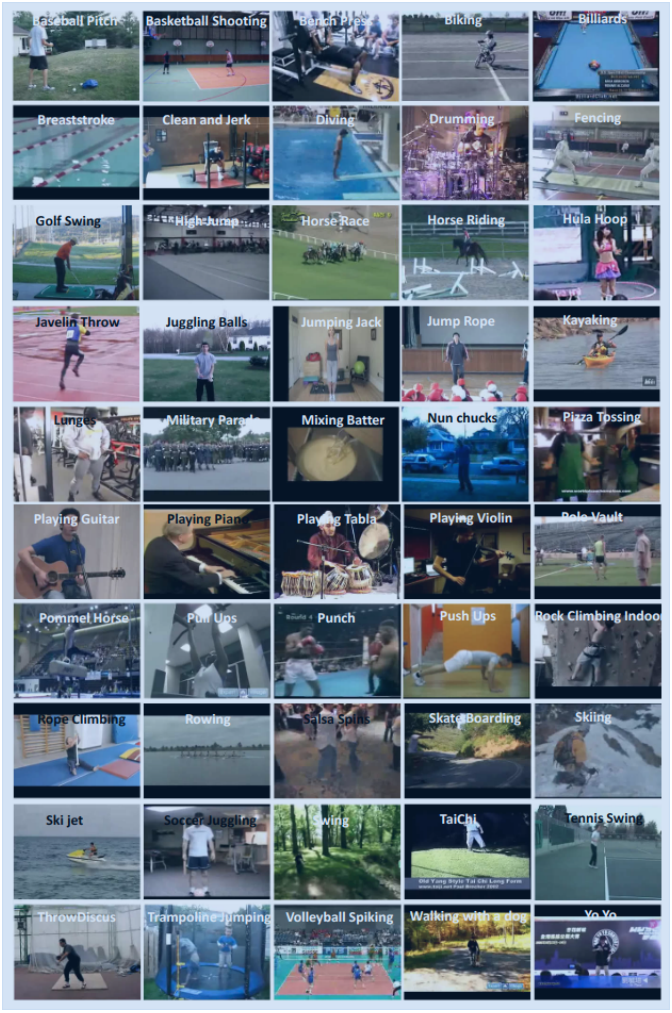


Figure 2.2: Action classes in the UCF50 dataset

2.2 Human gaze estimation

Gaze estimation plays a crucial role in determining whether a person is interacting with other persons or objects, and represents a non-trivial task to be performed.

Xia et al.[3] proposed a method based on eye corner and pupil detection from

[3]Li Xia et al. «Accurate gaze tracking from single camera using gabor corner detector». In: *Multimedia Tools and Applications* 75.1 (Jan. 2016). DOI: 10.1007/s11042-014-2288-4.

physical and cromatic features of the eye. Their method also allows to track the pupil's position when the subject is blinking. Their results are good, but the method relied on high-definition images of subjects looking directly at a monitor, taken by a camera close to the subjects' faces. Also, the model has to be calibrated prior to starting actual usage, and this necessity clashes with the operating environment of this work's project.

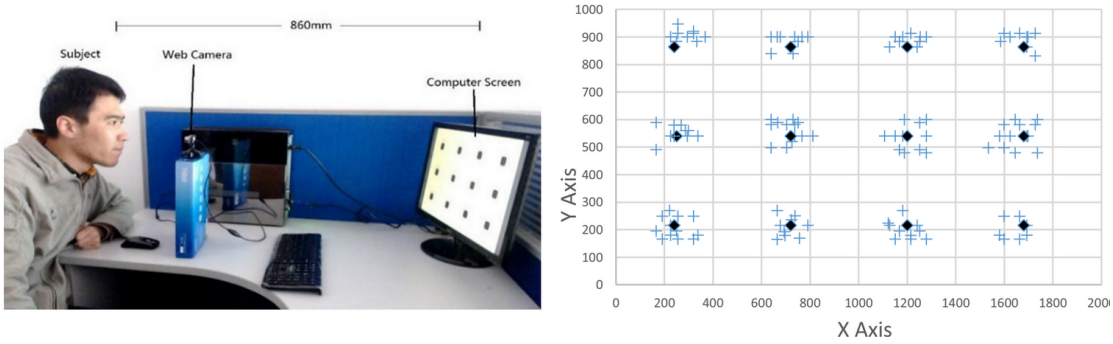
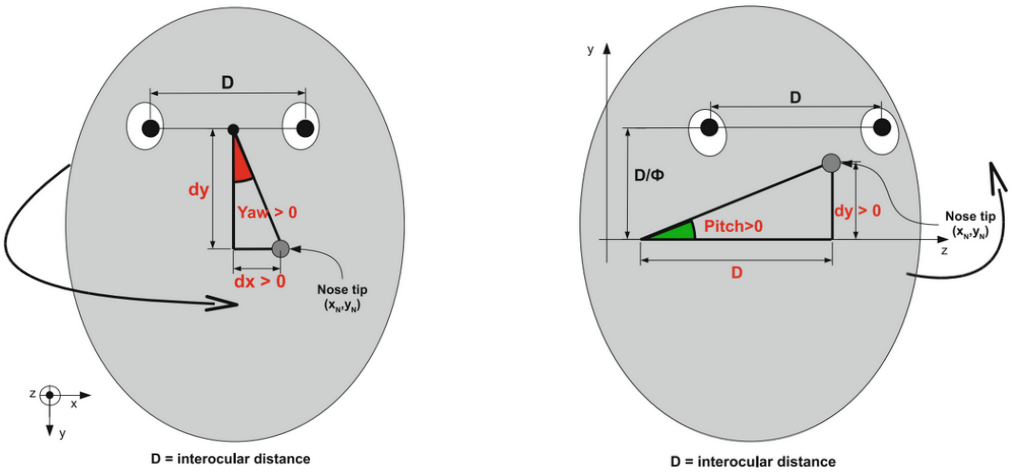


Figure 2.3: Xia et al.'s experimental setup

A simpler and more widely applicable approach is the one chosen by Marcialis et al.[4], which is based on known facial proportions, derived from Leonardo da Vinci's anatomical studies, and tries to compute the roll, pitch and yaw angles of the head through trigonometric operations on angles and body segment lengths, as shown in Figure 2.4.

The main advantage of this approach is that it only requires to know the 2D location of three points of the subject's face and it theoretically works fairly well whatever the distance between the camera and the subject is. Even this method, though, has a notable drawback: the authors themselves state that the method is

[4]Gian Marcialis et al. «A novel method for head pose estimation based on the "Vitruvian Man"». In: *International Journal of Machine Learning and Cybernetics* 5 (Feb. 2013), pp. 111–124. DOI: 10.1007/s13042-013-0188-y.



(a) **(b)**

Figure 2.4: Geometrical data used for yaw and pitch angles' calculations

accurate only if the nose tip is not too far from the position it has when the person is looking straight at the camera. This excludes many poses that may occur in real operational situations, such as persons looking up, down or to one side.

Chapter 3

Theoretical fundamentals

In this chapter the mathematical and theoretical aspects of the tools and algorithms used in the distraction score computation system will be described.

3.1 Stereoscopic vision

One of the most obvious problems to solve in most Computer Vision contexts is the localization of objects in the 3D space. Many different techniques are used. One of the most common ones is the same that the human brain uses to detect depth: it is called stereoscopic vision and relies on two cameras close to each other.

Let P be a point in space and L_1, L_2 the lenses of two cameras with equal focal length λ at a distance B from each other, as shown in the figure below. Given x_1 and x_2, horizontal coordinates of the representation of the point P in each image, the X and Z coordinates of the point in a coordinate system cenetered in the lens

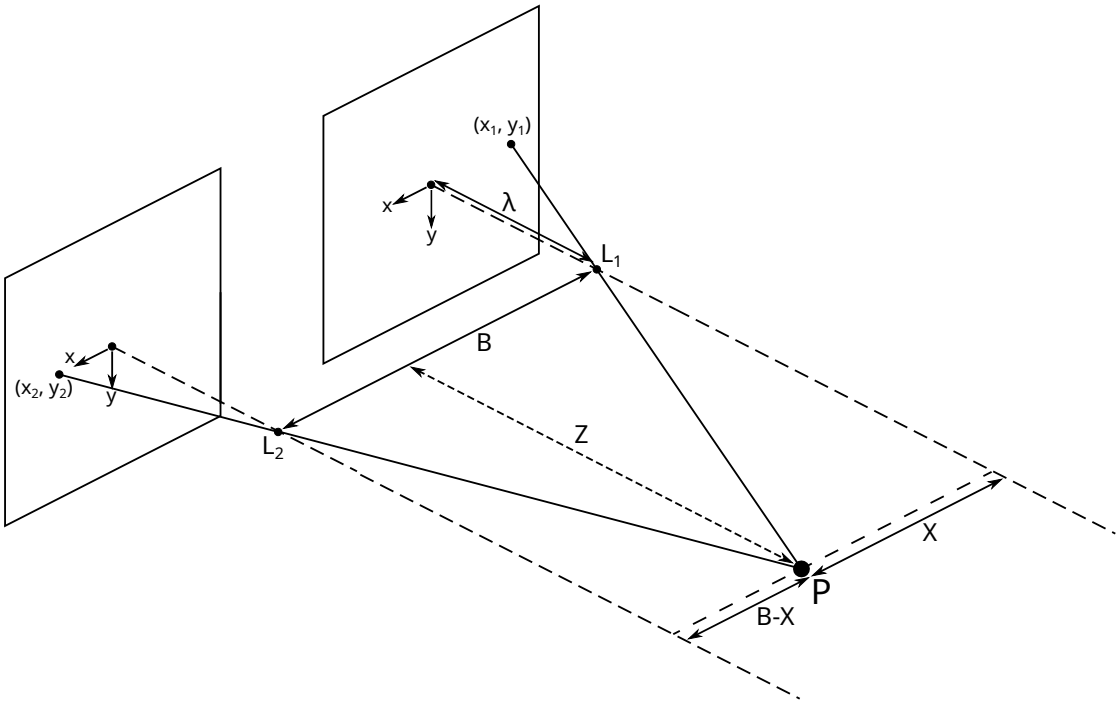


Figure 3.1: Computation of 3D coordinates of a point using stereoscopic vision

L_1 can be computed by solving this linear system:

$$\begin{cases} -x_1 : \lambda = X : Z \Rightarrow -x_1 Z = \lambda X \\ x_2 : \lambda = (B - X) : Z \Rightarrow x_2 Z = \lambda B - \lambda X \end{cases}$$

Summing the two equations:

$$(-x_1 + x_2)Z = \lambda B \Rightarrow Z = \frac{\lambda B}{x_2 - x_1}$$

Thus

$$X = -\frac{x_1 Z}{\lambda} = -\frac{x_1}{x_2 - x_1} = \frac{x_1}{x_1 - x_2}$$

Knowing Z, it is possible to compute also the Y coordinate.

3.2 Kalman filters

A Kalman filter is a mathematical tool that allows to estimate and predict the evolution of a dynamic system characterized by some degree of noise. Given a multiple input, multiple output system

$$\begin{cases} \mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) + \mathbf{v_1}(k) \\ \mathbf{y}(k) = \mathbf{C}\mathbf{x}(k) + \mathbf{v_2}(k) \end{cases} \tag{3.1}$$

in which $\mathbf{x}$ is the *state vector*, $\mathbf{y}$ is the *measurement vector*, $\mathbf{u}$ is a vector of *control inputs*, and $\mathbf{v_1}$ and $\mathbf{v_2}$ are white noises (i.e. funtions with Gaussian random values centered in 0), a Kalman filter is able to predict the future values of the state $\mathbf{x}$ (and in this case it is more properly called *Kalman predictor*) or to 'smoothen' its current or past measured values.

Kalman filters are recursive. For this reason, computations only require to know data from the previous time instant.

In this project an implementation of the *1-step Kalman predictor* in *predictor-corrector form* with no control input will be used (i.e. the $\mathbf{B}\mathbf{u}(k)$ term will not appear). This means that the Kalman predictor will be used to compute $\hat{\mathbf{x}}(N+1|N)$, the predicted state vector for the next (discrete) time instant $N+1$ basing on previously predicted states and measured data (until instant N).

In the following equations, the notation in Table 3.1 will be used.

Symbol	Description	Size
$\mathbf{x}(a)$	State vector at instant a	$n \times 1$
$\mathbf{y}(a)$	Measurement vector at instant a	$m \times 1$
$\mathbf{A}$	State transition matrix	$n \times n$
$\mathbf{V_1}$	Process noise matrix	$n \times n$
$\mathbf{v_1}(a)$	Process noise $\sim \mathcal{N}(0, \mathbf{V_1})$	$n \times 1$
$\mathbf{C}$	Measurement matrix	$m \times n$
$\mathbf{V_2}$	State covariance matrix	$m \times m$
$\mathbf{v_2}(a)$	Measurement noise $\sim \mathcal{N}(0, \mathbf{V_2})$	$m \times 1$
$\mathbf{P}(a)$	Predicted state covariance matrix for instant a	$n \times n$
$\mathbf{P_0}(a)$	Updated (a posteriori) state covariance matrix at instant a	$n \times n$
$\hat{\mathbf{x}}(a\|b)$	Predicted/filtered state vector estimate for instant a depending on instant b (predicted if $a > b$, filtered if $a = b$)	$n \times 1$
$\mathbf{e}(a)$	innovation (difference between actual and predicted measurement) at instant a	$m \times 1$
$\mathbf{K_0}(a)$	Kalman gain matrix at instant a	$n \times m$
$\mathbf{S}(a)$	Innovation covariance at instant a	$m \times m$
$\mathbf{I_n}$	Identity matrix of size n	$n \times n$

Table 3.1: Symbols used in Kalman predictor/corrector equations

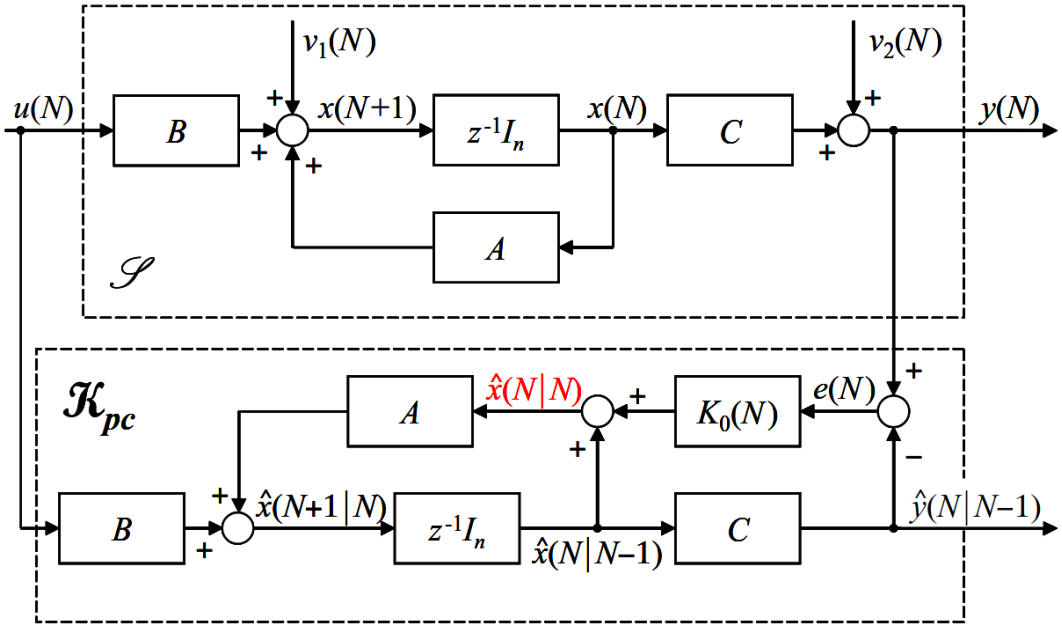


Figure 3.2: Schematic view of a Kalman predictor-corrector. The top half $\mathcal{S}$ represents the actual system, while the bottom half $\mathcal{K}_{pc}$ is the predictor-corrector.

This model works in two phases:

Update

Innovation	$\mathbf{e}(N) = \mathbf{y}(N) - \hat{\mathbf{x}}(N\|N-1)$
Innovation covariance	$\mathbf{S}(N) = \mathbf{C}\mathbf{P}(N)\mathbf{C}^T + \mathbf{V_2}$
Optimal Kalman gain	$\mathbf{K_0}(N) = \mathbf{P}(N)\mathbf{C}^T \left[\mathbf{S}(N)\right]^{-1}$
Filtered state estimation (corrector)	$\hat{\mathbf{x}}(N\|N) = \hat{\mathbf{x}}(N\|N-1) + \mathbf{K_0}(N)\mathbf{e}(N)$
Updated state covariance	$\mathbf{P_0}(N) = \left[\mathbf{I_n} - \mathbf{K_0}(N)\mathbf{C}\right]\mathbf{P}(N)$

Prediction

Predicted state estimation (predictor)	$\hat{\mathbf{x}}(N+1\|N) = \mathbf{A}\hat{\mathbf{x}}(N\|N)$
Predicted state covariance	$\mathbf{P}(N+1) = \mathbf{A}\mathbf{P_0}(N)\mathbf{A}^T + \mathbf{V_1}$

3.3 Machine Learning regression

Machine Learning (ML) is a set of techniques used to try to find relationships between known data inputs and outputs in order to minimimize the prediction errors on unknown data.

Learning can be either *supervised*, if the outcomes corresponding to input data are known, or *unsupervised* if no outcome measurement is available. Supervised learning problems, in turn, can be *classification* problems, in which the possible outcome can only be chosen from a finite set of elements, or *regression* problems, in which the outcomes are single or multiple numerical values. For the scopes of this project, only regression problems will be covered.

3.3.1 Regression models

Linear regression

Linear regression is one of the simplest and most widespread Machine Learning methods, and it is also widely used in other fields like statistics.

Given a batch of N data which are functions of n inputs $x_1, x_2, \ldots, x_n$, a linear regressor computes the linear functions

$$y = f(x_1, x_2, \ldots, x_n) = a_1 x_1 + a_2 x_2 + \cdots + a_n x_n + q$$

in order to minimize a given error function (usually the mean square error, aka MSE, or the maximum absolute error, aka MAE). Linear regression method can be extended to multiple output cases by using one regressor for each output component.

It is easy to notice that this method is not the most suitable for problems in which the relationships between inputs and outputs include polynomial or transcendental

functions.

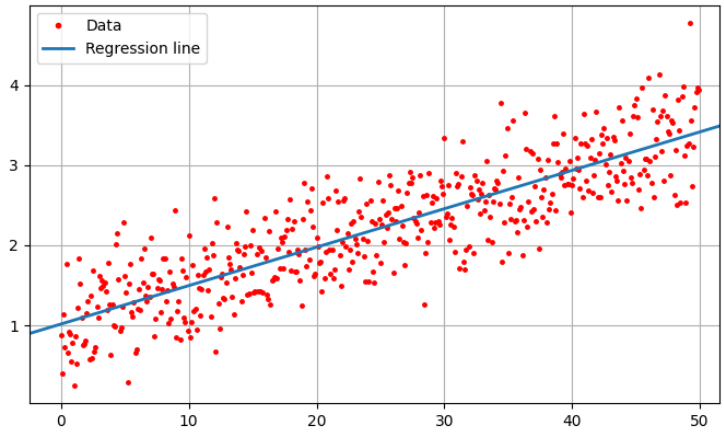


Figure 3.3: Example of linear regression on 1D data

Polynomial regression

A way to find more complex relationships between inputs and outputs is to use regressors with higher degree terms. As an example, a 2^{nd} degree (quadratic) n-input regressor has the following structure:

$$y = f(x_1, x_2, \ldots, x_n) = q + \sum_{i=1}^{n} a_i x_i + \sum_{i=1, j \geq i}^{n} b_{ij} x_i x_j = \\ = q + a_1 x_1 + a_2 x_2 + \cdots + a_n x_n + b_{11} x_1^2 + b_{12} x_1 x_2 + \cdots + b_{nn} x_n^2$$

q is the bias term, the first sum contains 1^{st} degree (linear) terms, and the second one contains 2^{nd} degree (quadratic) terms, each with its own weight. It is theoretically possible to increase the maximum degree at will, but the complexity of the computation grows polynomially with respect to the increase of degree d,

since the number of factors to compute is

$$\binom{n+d}{d} = \frac{(n+d)!}{n!d!}$$

where n is the number of input features and d is the maximum degree. It is thus necessary to find an acceptable tradeoff between accuracy and complexity of the model.

Exactly like linear regressors, polynomial ones can be used to infer multiple outputs by using one equation for each output.

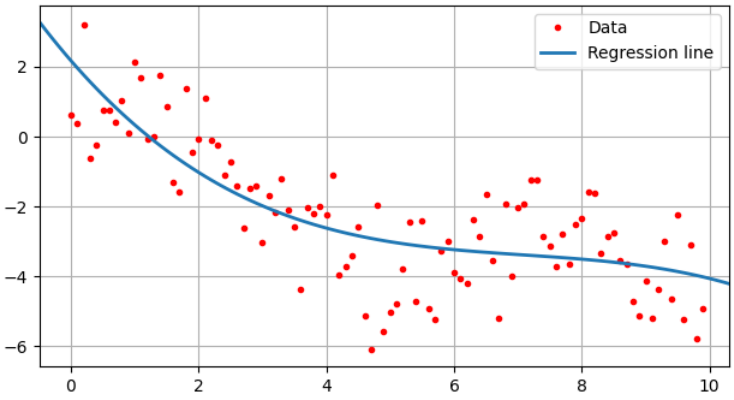


Figure 3.4: Example of polynomial (cubic) regression on 1D data

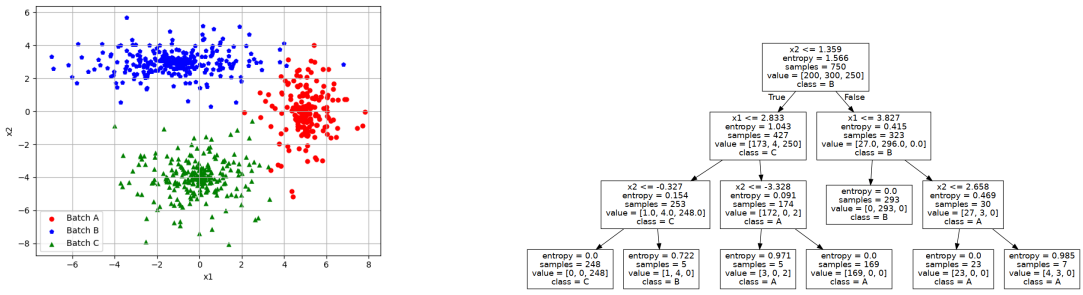
Decision trees

A decision tree is a classifier that operates by partitioning the instance space basing on binary checks on one input at a time.

The decision threshold is chosen in order to minimize the 'scattering' of data. In other words, each branching of the decision tree splits a portion of the n-dimensional feature space in two parts trying to make sure that data in each part

are as homogeneous as possible.

Decision trees are used for classification rather than regression, i.e. they are only able to generate a finite number of outputs, but if the tree is deep enough it is able to generate a very high number of output values, and this allows to obtain small errors on most inputs. The main drawback of using decision trees, though, is that this method tends to overfit the training data. In order to try and attenuate this phenomenon, it may be worth trying some ensemble method.



(a) Data **(b)** Decision tree

Figure 3.5: Example of application of binary decision tree classifier on a set of 2D data

Random forests (RF)

One of the most popular ensemble methods is Random Forests, proposed by Breiman in 2001[5]. In order to reduce overfitting, a certain number of trees are generated, and all the results are taken into account (in case of random forest regressors the mean of all predictions is computed) to form the final output. This allows to generate different regressors that use different logics to perform the same task.

[5]Leo Breiman. «Random Forests». In: *Machine Learning* 45.1 (Oct. 2001), pp. 5–32. ISSN: 1573-0565. DOI: 10.1023/A:1010933404324.

Two techniques are used to generate trees different from each other:

- bagging, i.e. generating random training sets by picking N times from the original training dataset with replacement;
- limiting the number of input features on which each node can operate.

The main drawbacks of Random Forests are the lower accuracy of each single tree they are made of, and their higher computational cost with respect to single decision trees.

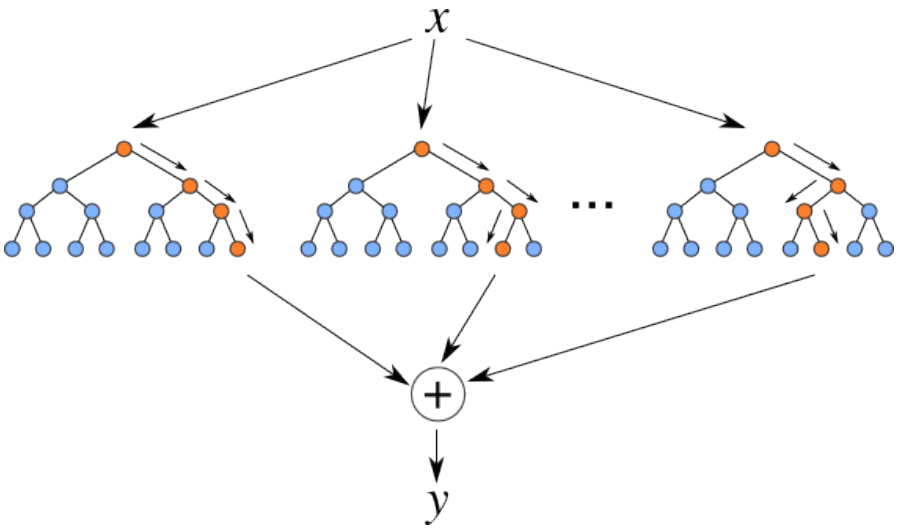


Figure 3.6: Random forest conceptual scheme

3.4 Neural networks

The diffusion of neural networks marked the beginning of a new era in the field of Machine Learning and started a new branch of research on its own, which is known as Deep Learning. The advancements in the research on Deep Learning, combined with a slight reduction in the power and cost requirements, allowed a growing

number of companies and researchers to develop and use a huge variety of neural networks for many different applications including, of course, object detection and segmentation.

Exactly like most supervised learning models, neural networks are trained on labelled data made of inputs $\mathbf{x}$ and corresponding outputs $\mathbf{y}$, with the aim of generalizing the same procedure to unknown inputs and producing outputs with the lowest possible error.

Neural networks can perform classification or regression tasks, depending on their internal setup.

3.4.1 Neurons and layers

The base component of all neural networks is the *neuron*, which is inspired by organic neural cells and, like them, receives signals (in this case numerical values) from upstream neurons and generates a single signal for downstream ones. The output signal is equal to

$$y = f(b + \mathbf{w}^T\mathbf{x}) = f(b + \sum_{i=1}^{n} w_i x_i)$$

where $\mathbf{x}$ is the vector composed by the n inputs coming from the previous layer of neurons, $\mathbf{w}$ is a weighting vector, b is a bias term and f is called *activation function*. The weight and bias terms actuate a linear combination of the inputs, while the activation function is used to introduce non-linearity elements to the network and to regularize the output in order to avoid instability. Examples of activation functions can be found in Table 3.2.

Neurons are organized in *layers*. Each layer is usually composed of neurons

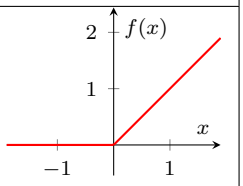
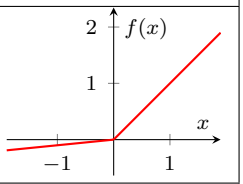
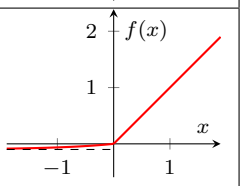
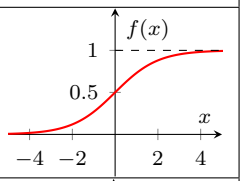
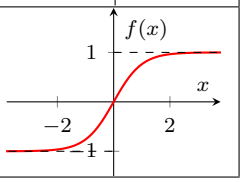
Name	Equation	Plot
ReLU	$\begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$	
Leaky ReLU	$\begin{cases} \alpha x & x < 0 \\ x & x \geq 0 \end{cases}$	
ELU	$\begin{cases} \alpha(e^x - 1) & x < 0 \\ x & x \geq 0 \end{cases}$	
Sigmoid	$\frac{e^x}{1 + e^x}$	
tanh	$\frac{e^x - e^{-x}}{e^x + e^{-x}}$	

Table 3.2: Common activation functions

identical to each other and is connected to a certain number of neurons in the immediately previous and following layers.

What sets neural networks apart from simpler Machine Learning models is the fact that training is non-deterministic and recursive. It aims at minimizing the value of a given cost function $J(\mathbf{W}, \mathbf{b})$, which depends on the results achieved by the current state of the network, by updating the weights $\mathbf{W}$ and biases $\mathbf{b}$

throughout the whole network by performing the following operations:

$$w \leftarrow w - \alpha \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial w} \quad \forall w$$

$$b \leftarrow b - \alpha \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial b} \quad \forall b$$

Training is performed by generating the outputs corresponding to certain inputs and trying to modify the weights and biases' values in order to try and make the cost function decrease. This method, called *gradient descent*, aims at finding the minimum of the cost function by moving the values of the weights and biases with steps whose amplitude depends on a factor α called *learning rate*. Higher learning rates speed up training, but may also lead the cost function to move away from local minima, while lower values make training slower but allow to reach more precisely the minima of the cost function.

If all neurons of each layer started from the same value, their evolution would be identical. This is why their weights have to be initialized (usually randomly) in order to have different values.

3.4.2 Neural network structure

All neural networks are made of:

- an *input layer* with the same size as the input data;
- an *output layer* made of a number of neurons equal to the number of desired outputs (for regression) or possible choices (for classification, except for the binary case in which there is only one output neuron);

- in most cases, a certain number of *hidden layers* between the input and output ones. The number of neurons in each hidden layer is highly variable.

In general a higher number of hidden layers allows to find more complex relationships between inputs and outputs but slows down both training and inference.

3.4.3 Neural network types

> "There is a tool for every task, and a task for every tool."
>
> *Tywin Lannister, from "A storm of swords" by George R.R. Martin*

Many different NN architectures were proposed throughout the years. Here the ones more useful for this project will be explained.

Fully connected feedforward network

Most neural networks are feedforward, i.e. data are only transmitted in the input to output direction, from a layer to the following one.

The simplest type of network is the fully connected network, in which, as shown in Figure 3.6, all neurons of each layer receive the values of all neurons from the previous layer and transmit their result to all neurons of the following layer. This structure is quite simple and is generally used when dealing with purely numerical data.

Even though the network is fully connected, in some cases a technique called dropout is used during training: it consists in "ignoring" some neurons at each step, in order to avoid that single neurons become too relevant and to prevent overfitting.

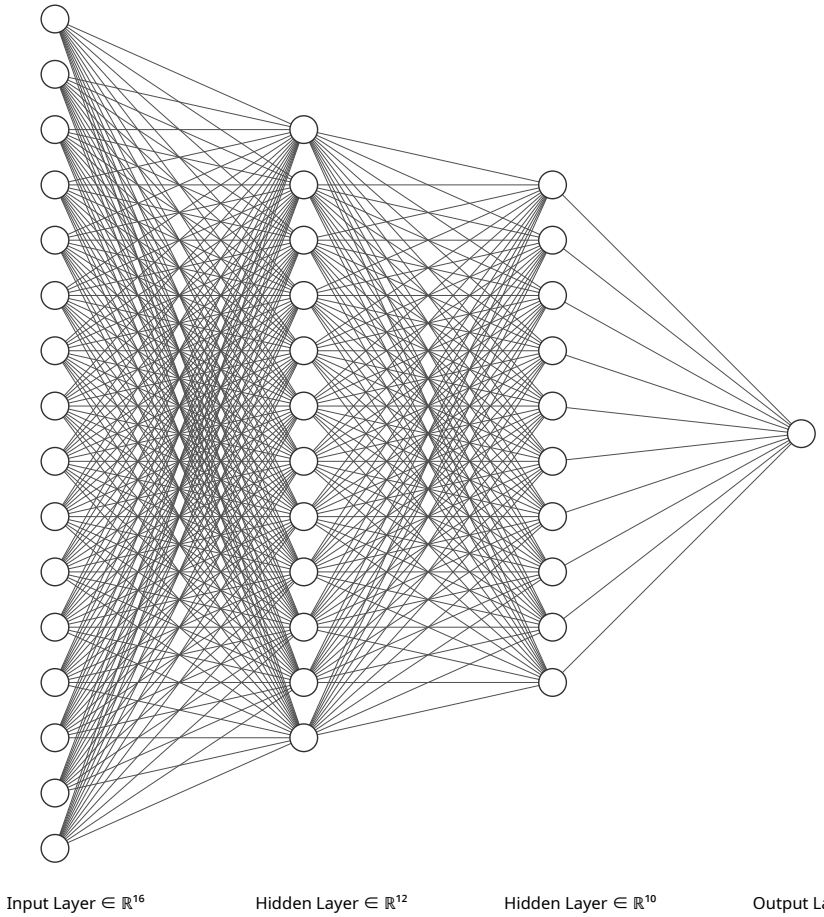


Figure 3.7: Example of a fully connected feedforward neural network structure

Convolutional neural network

Images can be considered as tensors of numbers: a digital color image of size $W \times H$ can be represented as a $W \times H \times 3$ tensor (the value 3 corresponds to the three color channels of RGB images).

Ordinary neural networks consider each input separately, and this may be an issue when looking for local patterns and properties. This is why a new kind of layers, known as convolutional layers, able to process groups of adjacent pixels ignoring the rest, were introduced: they take into account small pieces of the image

only, significantly reducing the number of weights for each neuron (e.g. a fully connected network operating on a 50px × 50px image would need $50 \cdot 50 \cdot 3 = 7500$ weights per neuron in the first hidden layer, while a convolution layer working on 5px×5px chunks would only need 75) and allowing to find certain graphic patterns everywhere in the image. In fact each neuron of a convolutional layer works as a stand-alone filter and is able to learn a pattern different from the others, so an N-neuron convolutional layer can recognize up to N different features or patterns in any part of the image and convert them in information usable by the following layers. The output of a convolutional neuron is called feature map, and its 2D size depends on those of the picture and of the filter itself and on a parameter called stride, which represents how far samples are from each other.

Another kind of layer common in convolutional networks is the pooling layer: it allows to reduce its inputs' size by synthetizing the properties of the inputs in a smaller shape (for example by replacing the values of four pixels that form a square with their average, or with the maximum between their values).

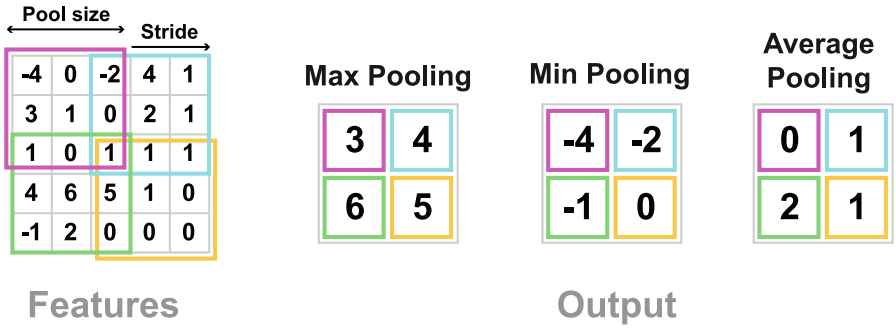


Figure 3.8: Possible pooling methods on a 5×5 matrix with window size 3 and stride 2

A complete convolutional network usually includes convolution, pooling and fully connected (also known as dense) layers. The final section of the network is usually made of dense layers and is preceded by a flattening layer that converts

the last feature maps (usually 3-dimensional) into a one-dimensional array.

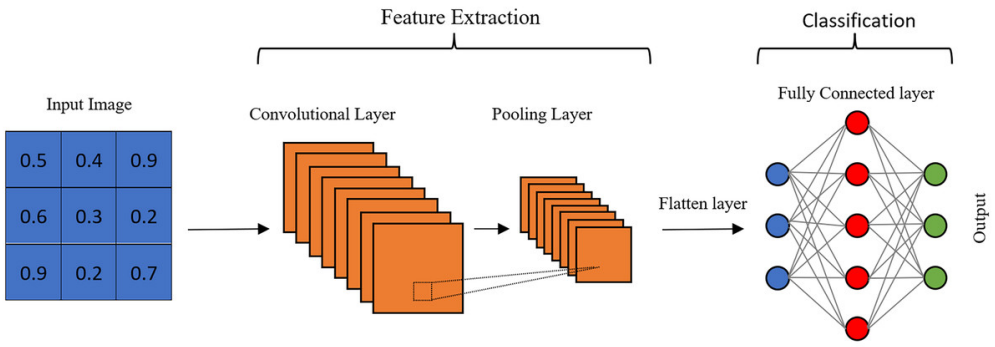


Figure 3.9: Structure of a shallow CNN

3.4.4 Convolutional neural networks for computer vision

Computer vision is defined as a field of AI aimed at extracting features and data from images and videos. It has a high number of applications in many different contexts, such as medical diagnosing and city traffic monitoring.

Most computer vision models rely on one or more convolutional neural networks appropriately designed and trained in order to maximize the performances on each specific task.

Some of the Computer Vision tasks that CNNs can perform are listed in Table 3.3.

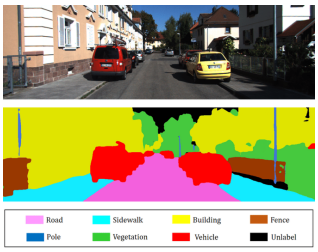
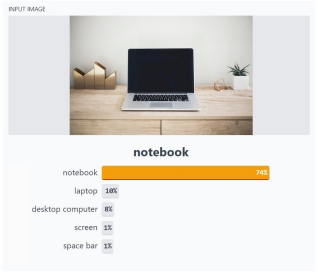
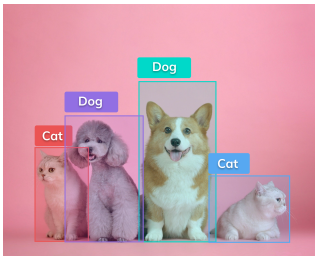
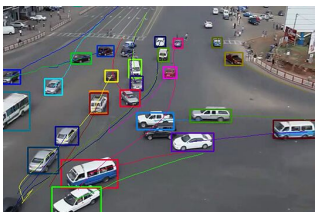
Task	Example	Aim and notes
Semantic segmentation		Split the whole image in regions corresponding to different classes
Image classification		Assign a class to the whole image basing on the objects contained in it
Object detection		Determine bounding boxes and classes for objects contained within the image, if any
Instance segmentation		Determine which pixels are part of each detected object
Object tracking		Keep track of the position and movement of one or more objects
Pose estimation		Detect the position of a certain number of keypoints of a subject

Table 3.3: Computer vision tasks

Chapter 4

Attention estimation system

4.1 Scope of the system

The aim of this project is to use Computer Vision methods to detect the persons near a robot that moves in a human-populated indoor scenario (such as an office) and compute a score able to define whether each of them is available to assist the robot in some kind of task, such as opening a door, pushing the call button of an elevator, or even to interact with the robot itself in other ways.

4.1.1 Human distraction and interest factors

As the robot needs to evaluate which person is the best to interact with, it is necessary to identify the possible sources of distraction for humans in an indoor scenario and the possible signals of availability for interaction with the robot.

Grouping

When two or more people are, and stay for a while, close to each other, it is likely that there is some interaction in process. This, of course, represents a huge distraction factor for the humans involved. The probability that each person is actually distracted tends to increase proportionally to the number of persons that are part of the group.

Interaction with objects

If a person is using some kinds of object, it is likely that they are distracted by it. The level of distraction depends on the specific object used and the way the human interacts with it. For example, it is more likely that a person is busy if they are holding a mouse and watching a monitor, rather than if they are just watching a monitor or TV screen.

Gaze towards the robot

On the other hand, if a person is staring at the robot, or their gaze is headed approximately towards it, it is very likely that they noticed the robot's presence and this might make them more interested in interacting with it. The probability that the person is intentionally looking at the robot increases with the duration of the gaze.

Distance from the robot

If two or more persons are in the same distraction conditions, the robot should obviously consider as better candidates for interaction the persons closer to itself, in order to optimize interaction times.

4.2 Methods and algorithms used

4.2.1 Foreword: the global coordinate frame

All of the geometrical locations mentioned in the following chapter refer to a unique reference frame defined as follows:

- the origin corresponds to the camera objective;
- the X axis points to the right;
- the Y axis points down;
- the Z axis is perpendicular to the image plane and points away from the camera;
- distances are usually expressed in millimeters, except where otherwise specified.

4.2.2 Ultralytics YOLO neural networks

YOLO (acronym for You Only Look Once) is a family of convolutional neural networks designed to perform various Computer Vision tasks:

- Image classification;
- object detection (even with oriented bounding boxes);
- instance segmentation;
- pose estimation.

This project makes use of YOLOv8, which was the state of the art in 2023.

This version of YOLO relies on EfficientNet[6], with its 71 layers and 19 million parameters, for feature extraction, and on NAS-FPN[7] for detection. YOLOv8 is able to recognize up to 15 objects in each picture.

Instance segmentation

Instance segmentation is a task somehow similar to object detection. Both tasks aim at recognizing which objects are present in a picture and both identify the bounding box sorrounding the object, but instance segmentation also recognizes which pixels in the box are actually part of the object. This aspect obviously makes the instance segmentation network slower and more complex than the object detection one. In the starting phases of the project's development, actually, it was chosen to use object detection in order to avoid slowing down the whole process of score computation, but it soon became evident that the bounding box was not sufficient to extract all the data needed for each object, so instance segmentation was chosen. More details can be found later in this chapter.

YOLOv8 instance segmentation model is availailable as pre-trained on the COCO (Common Objects in Context) dataset[8] and, as a consequence, it is able to recognize 80 different object classes, each one associated to a numeric label from 0 to 79.

In order to enhance the model's performance, though, it was decided to re-train the neural network only considering the object classes useful for this project, which

[6]Mingxing Tan and Quoc V Le. «EfficientNet: Rethinking model scaling for convolutional Neural Networks». In: (May 2019). DOI: 10.48550/arXiv.1905.11946.

[7]Golnaz Ghiasi et al. «NAS-FPN: Learning scalable feature pyramid architecture for object detection». In: (2019). DOI: 10.48550/arXiv.1904.07392.

[8]Tsung-Yi Lin et al. «Microsoft COCO: Common objects in context». In: (2014). DOI: 10.48550/arXiv.1405.0312.

are listed in 4.3.2. This allowed to increase the accuracy of the predictions regarding the objects of interest.

A confidence score in the [0,1] range is associated to each detection.

The pixel masks are represented in a format compatible with the OpenCV library, while the bounding boxes are expressed in the XYXY (i.e., X and Y coordinates of the top left and bottom right corners) and XYWH (top left corner X and Y, then width annd height of the box) formats.

Figure 4.1: Result of instance segmentation with YOLOv8

Pose estimation

YOLOv8 pose estimation network aims at detecting and marking where specific points of objects are, and also generates each object's bounding box.

The detected points depend on the dataset used to train the model. The pre-trained edition of YOLOv8 is trained on the COCO dataset, which features 17 keypoints of the human body, as shown in the image below.

Index	Key point
0	Nose
1	Left-eye
2	Right-eye
3	Left-ear
4	Right-ear
5	Left-shoulder
6	Right-shoulder
7	Left-elbow
8	Right-elbow
9	Left-wrist
10	Right-wrist
11	Left-hip
12	Right-hip
13	Left-knee
14	Right-knee
15	Left-ankle
16	Right-ankle

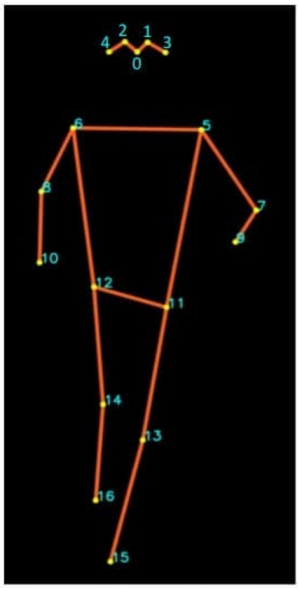


Figure 4.2: YOLOv8 pose estimation keypoints

In order to obtain more precise gaze estimations, though, the model was re-trained using a modified version of the dataset, called COCO-WholeBody[9], which includes 133 keypoints, most of whom on the hands and face. The keypoints that were added are the angles of the eyes, marked as 60, 63, 66 and 69 in the image below, and became keypoints 17 to 20 in the new version of the YOLOv8 pose estimation network.

[9]Sheng Jin et al. «Whole-Body Human Pose Estimation in the Wild». In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2020.

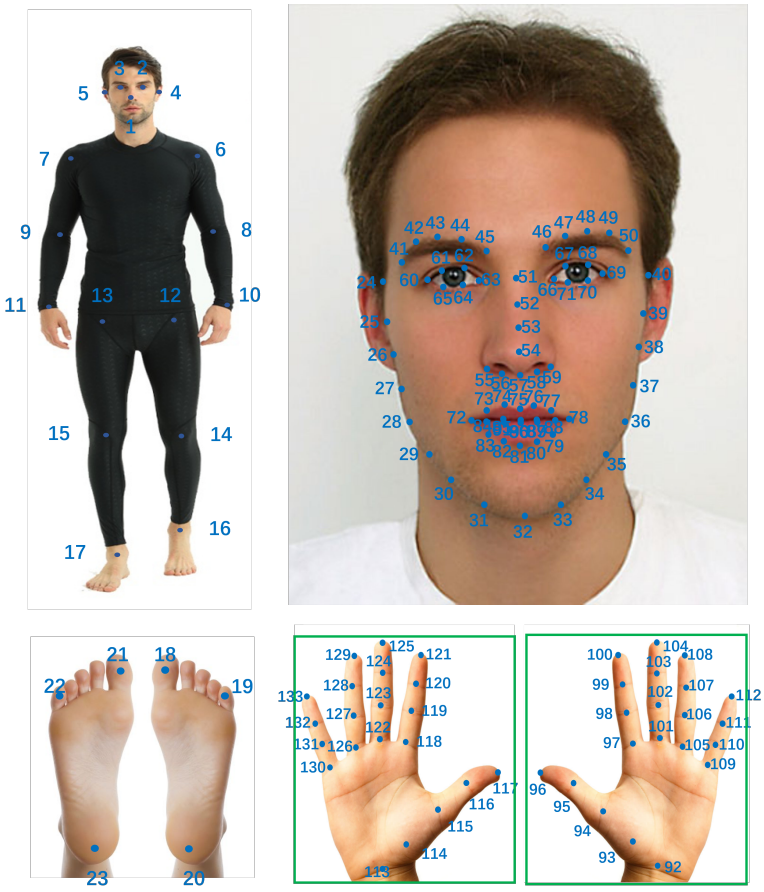


Figure 4.3: COCO-WholeBody keypoints

4.2.3 Simple Online and Realtime Tracking

In some computer vision applications it is important not only to detect objects, but also (sometimes especially) to keep track of their position in the image or another coordinate system over time. To achieve this result it is necessary to:

- create a data structure for storing tracked objects' data;
- detect the objects from in the video input of interest (this is usually done by using an object detection neural network);

- check whether each detected object corresponds to any of the objects that are already being tracked and: if it is not and some conditions are met, add the detected object to the tracked ones and assign an ID to it; if it is, update the given object's data;
- delete from the the system the objects that have not been detected for a suitable time (or number of frames).

The second point of the list above is performed by the YOLOv8 pose estimation neural network, as explained in the previous section, while the operations related to the object container can be easily managed using dedicated classes.

For the scopes of this project the points taken into account to define the objects (more precisely the persons) to track are the corners of the bounding boxes containing the persons. These points are then converted into the four variables used by the SORT algorithm (described below).

Other choices of the points, such as taking into account the single points that form the pose skeletons, would have been more accurate but also way more expensive computationally.

The most complex part of this small algorithm is the association of the new bounding boxes to the existing objects and viceversa.

The SORT algorithm

The method developed by Bewley et al.[10] defines the tracked bounding boxes by using four parameters: the x and y coordinates of the center of the bounding box (called u and v in the original paper), the size s and the aspect ratio r of the box.

[10] Alex Bewley et al. «Simple online and realtime tracking». In: *2016 IEEE International Conference on Image Processing (ICIP)*. 2016, pp. 3464–3468. DOI: 10.1109/ICIP.2016.7533003.

The aforementioned parameters are not directly provided by the pose estimation network,which provides different kinds of data, namely the coordinates of the top left and bottom right points of each box in the format $[x_1, y_1, x_2, y_2]$.
Given

$$w = x_2 - x_1$$

and

$$h = y_2 - y_1,$$

the SORT algorithm implementation converts these data into the Kalman filter's variables using the following equations:

$$u = \frac{x_1 + x_2}{2}$$
$$v = \frac{y_1 + y_2}{2}$$
$$s = w \cdot h$$
$$r = \frac{w}{h}$$

A Kalman filter takes the four variables as inputs and estimates the state $\mathbf{x}$ of the box in terms of the raw coordinates and the velocities associated to u, v and s.

$$\mathbf{x} = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}]^T$$

The measured values, though, are only the first four, because there is no way to actually track the velocities accurately.

$$\mathbf{y} = [u, v, s, r]^T$$

The matrices used by the Kalman filter are thus the following:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

The first three rows of the **A** matrix represent the fact that the new location (u and v) and scale s of the bounding box depend on the past values and velocities. The structure of the **C** matrix reflects the fact that the velocities (corresponding to the last three columns) are only inferred and never actually measured.

Bounding box prediction

The predicted state $\hat{\mathbf{x}}(N+1|N) = [\hat{u}, \hat{v}, \hat{s}, \hat{r}]^T$ is used to generate a "predicted" bounding box for the following frame for each of the tracked objects, using these equations:

$$\hat{w} = \sqrt{\hat{s}\hat{r}}$$
$$\hat{h} = \frac{\hat{s}}{\hat{w}}$$
$$\hat{x}_{1/2} = \hat{u} \mp \frac{\hat{w}}{2}$$
$$\hat{y}_{1/2} = \hat{v} \mp \frac{\hat{h}}{2}$$

When frame $N + 1$ is acquired and the new bounding boxes are processed, the Hungarian algorithm[11] is used to associate them to the Kalman filter predictions. The assignment criterion is the IoU (Intersection over Union) factor computed for each {Predicted; Actual} bounding box couple.

Since the assignment algorithm aims at creating the highest possible number of associations, regardless of the absolute value of the cost functions it works with, a minimum IoU value is imposed in order to remove the most unlikely associations from the algorithm. This may obviously lead to a number of successful associations lower than the numbers of predicted and detected bounding boxes.

Every detected box which does not get associated to an existing ID receives a new one and is added to the tracked bounding boxes, while each predicted box which is not associated to a prediction simply updates its state and waits for the following frame. A counter is also updated to record how many times in a row a specific ID has not been paired with any detected bounding box: if and when the counter reaches a specific maximum value, the ID is no longer used and the corresponding predicted bounding box is deleted.

4.2.4 Gaze estimation

Gaze estimation is the task of recognizing the direction of one or more persons' eyesight(s) in a video stream or image. Estimating where a person is looking at may be a helpful factor in determining whether they are interacting with someone or something around them. This knowledge may be applied in many different ways. One of the most widespread uses of gaze estimation is checking whether a person

[11] H. W. Kuhn. «The Hungarian method for the assignment problem». In: *Naval Research Logistics Quarterly* 2.1–2 (Mar. 1955), pp. 83–97. ISSN: 1931-9193. DOI: 10.1002/nav.3800020109. URL: http://dx.doi.org/10.1002/nav.3800020109.

is looking away from the screen during computer-held tests and exams. In this project gaze estimation is used to infer whether a person is looking at objects in the scene or the camera.

Geometrical methods

Methods based on geometry rely on some hypotheses about position and proportions of human faces, usually taking into account eyes and nose or specific points on them. An example of this approach is the one proposed by Marcialis et al.[12], already mentioned in Chapter 2.

Other methods, more precise than the aforementioned one, require more key-points and are thus not usable in this context.

In all cases, some sort of neural network is needed to recognize the facial features to be processed.

Deep learning methods

Gaze estimation (or, more generally, head pose estimation) is a perfect example of problem in which convolutional neural networks may be helpful or even resolutive. An example is the method proposed by Ruiz et al.[13]. These methods, though, are computationally too expensive to guarantee real-time performance on hardware such as the Intel® NUC used in this project, especially if used alongside other networks like the YOLO ones, which are necessary for other tasks.

---

[12]Gian Marcialis et al. «A novel method for head pose estimation based on the "Vitruvian Man"». In: *International Journal of Machine Learning and Cybernetics* 5 (Feb. 2013), pp. 111–124. DOI: 10.1007/s13042-013-0188-y.

[13]Nataniel Ruiz et al. «Fine-Grained Head Pose Estimation Without Keypoints». In: (2017). eprint: arXiv:1710.00925.

Gaze estimation with geometrical inputs

The unsatisfactory precision of the Vitruvian Man estimation and excessive weight of CNN methods made necessary to try a new approach based on the following workflow:

1. Find, or build, a gaze estimation dataset;
2. Use YOLOv8 pose estimation network to extract face keypoints' coordinates;
3. Choose input and output formats compatible with the chosen method;
4. Choose a ML or DL method;
5. Process the data from step 3 passing them as inputs and outputs to the chosen ML or DL model for training and testing.

These steps will be now described in more detail.

Gaze estimation dataset

Gaze360 Finding a gaze estimation dataset usable to train a gaze estimator for generic person placements was not an easy task, because most studies on this kind of task are based on the assumption that the camera and/or the subject is in a known and fixed position, and this constraint reduces the variability of gaze directions.

Nevertheless, datasets with a good degree of variablity in gaze directions exist. One of these is the Gaze360 dataset[14], composed of thousands of pictures of 238 different subjects in various poses. Each picture is labelled with gaze data for each

[14]Petr Kellnhofer et al. «Gaze360: Physically unconstrained gaze estimation in the wild». In: (Oct. 2019). arXiv: 1910.10088 [cs.CV].

person, and the dataset itself provided pre-cropped head images to make data collection easier.

Gaze360 gaze vectors are referred to a specific reference frame in which the x axis points to the left, the y axis to the top of the image and the z axis away from the camera. This made necessary to multiply all vectors by an appropriate rotation matrix

$$R = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

to express the vectors in the same reference frame used by the rest of the system.

Dataset expansion using Blender In order to try to improve the gaze estimation models' performances, another dataset was generated. This was done by using a single Blender 3D model (available online) featuring a fully rigged human figure. Blender's compatibility with Python allowed to write a script that automatically managed the creation of around 100,000 new pictures of the same subject viewed from different points and with the head in different positions.

The union of Gaze360 and the Blender-generated dataset resulted in more than 120,000 unique images that could be used for training and testing of the candidate gaze estimation models.

YOLOv8 for pose estimation

The pose estimation model described in 4.2.2 was fed with all the images from the dataset described above and a database of face keypoints was generated. The first attempts were performed taking into account three keypoints (nose, middle of left eye, middle of right eye), which correspond to keypoints `[0,1,2]` of the pose

estimation model, as shown in Figure 4.2. It was later decided to re-train the pose estimation model as described in 4.2.2 and, after this, four new keypoints became available: the left and right extremities of each eye, corresponding to keypoints `[17,18,19,20]` of the new model.

Data input and output formats

Figure 4.4: First choice of input data

Input data The features that were chosen to perform gaze estimation were some geometrical data derived from the positions of eyes and nose provided by the YOLOv8 pose estimation model. The first attempts were performed with the following data, also shown in the figure above:

- Distance between nose and left eye (`dist_nl`)
- Distance between nose and right eye (`dist_nr`)
- Angle of the segment between nose and left eye (`angle_l`)
- Angle of the segment between nose and right eye (`angle_r`)
- Angle of the segment between the two eyes (`angle_e`)

Lengths are normalized with respect to the distance between the eyes (`dist_e`), while angles are taken with respect to the horizontal direction and rescaled so that the maximum value is 1 rather than 2π.

Figure 4.5: Second choice of input data

After the pose estimation network retraining, these data were extracted from each set of keypoints:

- Distance between eyes (`dist_e`)
- Size of each eye (i.e. length of the segment between the two eye corners) (`size_l` and `size_r`)
- Angle of the segment between the two eyes (`angle_e`)
- Angle of the segment between the two eye corners, for each eye (`angle_l` and `angle_r`)
- Position of each pupil with respect to the segment between the eye's corners (normalized to [0,1])
- Position of the nose with respect to the midpoint of the segment between the two pupils (normalized to $[-1,1]$)

Segment lengths are normalized with respect to the distance of the nose from the segment connecting the pupils (`dist_ne`).

The data that are actually passed to the various methods that were tested are:

- data related to length, normalized;
- trigonometric functions of angle data;
- in some cases, "symmetric" logarithms of the aforementioned inputs (it will be defined later in this chapter).

Output formats The output of a gaze detection model is a 3D vector. It can be expressed in different ways, and the two that were used in this thesis are the following:

- x, y and z coordinates of the gaze vector in a specific reference frame (usually the global one described in 4.2.1), normalized so that the resulting vector has unit norm;
- spherical coordinate angles θ and ϕ, in some cases normalized to be within the [0,1] interval.

The first method has the advantage that its outputs directly represent intuitive and immediately usable values, but it is necessary to keep in mind that in most cases the outputs will not be automatically normalized. The second format, on the other hand, allows to compute one less output value but may not be as immediate and the correlation between inputs and outputs may not be straightforward and, as a consequence, the chosen estimation method may perform more poorly.

Machine Learning models for pose detection

Many different Machine Learning methods were tested:

- Linear regression (LR)
- Polynomial regression
- Decision trees
- Random forests (RF)
- Neural network

All of these methods were described in the previous chapter. There is, though, an uncommon aspect related to neural networks that needs to be further explained.

Multiplications in neural networks Simple neural networks usually only perform algebraic sums of weighted inputs and biases. It may be sufficient in many use cases, but not in this one. The input data have geometrical meanings, since they represent segment lengths and angles (or sines and cosines of angles), thus it is straightforward to imagine that, whatever the mathematical relationship between the eyes and nose's positions is, it somehow depends on sums of products of the aforementioned data. The issue is that neural networks are not designed to calculate products.

An interesting and effective workaround was found by Goel et al.[15], who proposed to include logarithms of the standard inputs as additional inputs, in order to exploit the property that a sum of logarithms is equal to the logarithm of the product of

[15]Bhaavan Goel. «Estimating multiplicative relations in neural networks». In: (2020). DOI: 10.48550/arXiv.2010.15003.

the arguments. In order to avoid that issues given by the values that logarithms assume when their arguments are lower than 1 (i.e. divergence for values close to 0, non-definition of logarithm for negative values), two new functions were introduced.

Symmetric logarithm "Symmetric logarithm" is a function that allows to extend the logarithmic function to negative numbers and preserve the sign of operands.

$$\text{symlog}(x) = \text{sgn}(x) \log(1 + |x|) = \begin{cases} \log(1 + x) & \text{for } x \geq 0 \\ -\log(1 - x) & \text{for } x < 0 \end{cases}$$

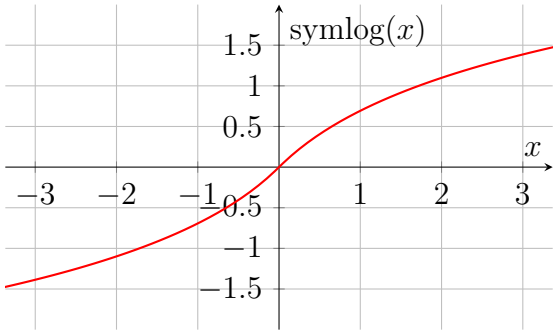


Figure 4.6: Plot of symmetric logarithm function

Symmetric exponential In order to reconvert the results of sums of logarithms (corresponding to products of inputs) into "ordinary" values, the "symmetric exponential" was used as activation function. This function is defined as

$$\text{symexp}(x) = \begin{cases} e^x - 1 & \text{for } x \geq 0 \\ -e^{-x} + 1 & \text{for } x < 0 \end{cases}$$

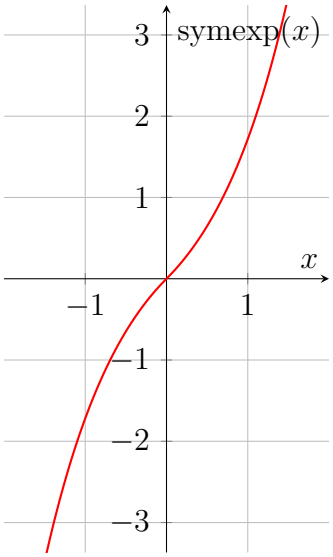


Figure 4.7: Plot of symmetric exponential function

These two functions allow to deal with both positive and negative values without any issue, preserve the sign of the original data and are continuous and differentiable in any point.

Some of the tested networks' layers are made of some ordinary neurons that only receive the ordinary data as input and by some "multiplication" neurons that operate using the "symmetric logarithms" of ordinary inputs and use the "symmetric exponential" as activation function. From the second hidden layer on, the neurons are fully connected.

Neural network structure The neural networks designed for pose estimation in this project are structured like this:

- a first layer made of two sets of 20 neurons. The neuron of the first set are fully connected to the "linear" inputs and use the "tanh" activation function, while the second set is made of neurons that receive "logarithmic" inputs (i.e. the "symmetric logarithms" of the "linear" inputs) and use the "symmetric

exponential" as activation function;

- a certain number of intermediate layers alternated like this: one fully connected layer with ReLU activation function, and another one that takes the "symmetric logarithms" of the previous layer's outputs and uses "symmetric exponential" as activation function, and so on;
- an output layer with three neurons with a "normalization" activation function, which ensures that the Euclidean norm of the output vector is 1.

The best results were obtained by using 5 intermediate layers (3 with ReLU activation, 2 with "symmetric exponential" activation) made of 20 neurons each.

Data processing

Training of models, except for neural networks, was performed using 90% of the dataset for training and 10% for testing. Dataset splitting was random. The cost function used in most cases was MSE. The neural network, on the other hand, aimed at minimizing another metric, the "scalar product loss" (described below).

Neural network training was performed using 10-fold cross validation and the Adam optimizer[16].

Scalar product loss The desired output of a gaze estimation system is a vector, or, better, the direction of a vector, since its magnitude is not relevant. This implies that the most used cost functions, such as the mean square error or the maximum absolute error, may be misleading: most multi-output Machine Learning models cannot impose constraints on multiple variables at a time and simply try to

[16]Diederik P Kingma and Jimmy Ba. «Adam: A method for stochastic optimization». In: (2014). DOI: 10.48550/arXiv.1412.6980.

approximate each simple variable (i.e. each Cartesian component or each spherical coordinate angle) on its own, and this may lead to having vectors with large MSE or MAE errors but with direction very similar to the ground truth one.

A better metric to check how similar the directions of two vectors are is cosine similarity, i.e. the cosine of the angle between the two vectors $\mathbf{u}$ and $\mathbf{v}$. It can be easily computed by reversing the scalar product formula:

$$\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\| \, \|\mathbf{v}\| \, \cos\alpha$$

$$\cos\alpha = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \, \|\mathbf{v}\|} = \frac{\sum_{i=1}^{3} u_i v_i}{\sqrt{\sum_{i=1}^{3} u_i^2 \cdot \sum_{i=1}^{3} v_i^2}}$$

The issue with this metric is that it grows if the vectors are close to each other in terms of direction, while Machine Learning models usually require a cost function to be minimized. One way to express cosine similarity as a cost function is this:

$$L_{\cos}(\alpha) = 1 - \cos\alpha$$

This expression shrinks when α decreases and is always non-negative, so it is a good choice for a cost function.

The metric chosen to compare the results of the methods proposed is given by the average between the median value of α (expressed in degrees) and its mean on the same testing dataset. The median value is taken into account because it helps to find how many outliers are produced by each model. If two models have the same mean loss, the one with the lower median loss is preferable, because it means that it performs fairly well more often than the one with a higher median

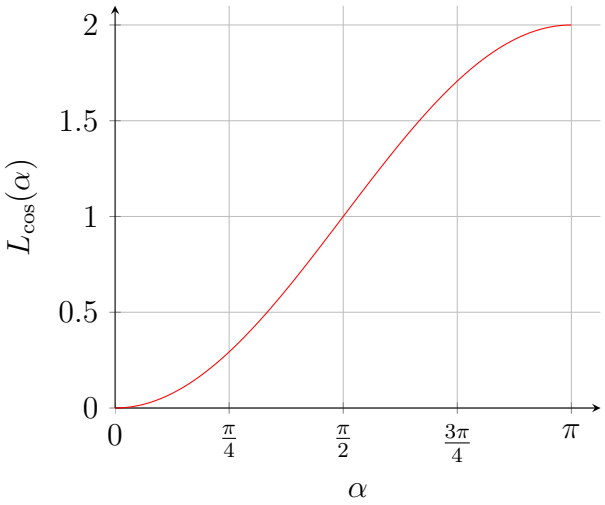


Figure 4.8: Plot of scalar product loss function

loss, even though its errors tend to be larger.

Training and testing results overview

Model	Mean Error	Median Error	Overall Error	Notes
Linear regression	15.375	10.668	13.021	
Polynomial regression	13.306	8.195	10.750	Maximum degree = 3
Decision tree	23.042	21.923	22.483	Maximum depth = 10
Random forest	13.312	8.098	10.705	10 trees Half features per node
Neural network	12.666	7.231	9.948	12 ordinary inputs 12 logarithmic inputs Neurons per layer: 40, 20, 20, 20, 20, 20, 3 See 4.2.4 for details

Table 4.1: Recap of the tested ML gaze estimation methods

As a consequence of this results, it was decided to use the neural network for gaze estimation.

4.2.5 Gaze evaluation

Gaze vectors are used to check whether persons are looking at something in particular. This information contributes to the computation of each person's distraction score.

Staring at an object

It is asserted that a person is looking at an object if their gaze line (i.e. the straight line parallel to the gaze vector starting from the midpoint of the segment between the eyes) crosses the object's 3D bounding box. This condition is expressed as a system of equations.

Each bounding box is made of 6 rectangular sides orthogonal to the main reference frame axes. Given the top front left (x_1, y_1, z_1) and bottom back right (x_2, y_2, z_2) points of the box, the six sides are defined by the following systems:

Front	**Top**	**Left**
$\begin{cases} x_1 \leq x \leq x_2 \\ y_1 \leq y \leq y_2 \\ z = z_1 \end{cases}$	$\begin{cases} x_1 \leq x \leq x_2 \\ y = y_1 \\ z_1 \leq z \leq z_2 \end{cases}$	$\begin{cases} x = x_1 \\ y_1 \leq y \leq y_2 \\ z_1 \leq z \leq z_2 \end{cases}$

Back	**Bottom**	**Right**
$\begin{cases} x_1 \leq x \leq x_2 \\ y_1 \leq y \leq y_2 \\ z = z_2 \end{cases}$	$\begin{cases} x_1 \leq x \leq x_2 \\ y = y_2 \\ z_1 \leq z \leq z_2 \end{cases}$	$\begin{cases} x = x_2 \\ y_1 \leq y \leq y_2 \\ z_1 \leq z \leq z_2 \end{cases}$

The gaze vector can be expressed as $[x_G, y_G, z_G]$ and its starting point as (x_E, y_E, z_E). Thus, the gaze line equations are:

$$\begin{cases} x = x_E + x_G t \\ y = y_E + y_G t \\ z = z_E + z_G t \end{cases}$$

for any $t \geq 0$.

In order to check whether the gaze vector crosses one of the sides of the bounding box, the fixed value of that side (for example z_1 for the front face) is substituted in the respective gaze line equation to obtain the value of t corresponding to the intersection of the gaze vector with the plane containing the side. For example, for the front side:

$$z_1 = z_E + z_G t^* \Rightarrow t^* = \frac{z_1 - z_E}{z_G}$$

After that, t^* is used to compute $x^* = x_E + x_G t^*$ and $y^* = y_E + y_G t^*$, the coordinates of the intersection between the gaze vector and the plane on which the side lies. If $x_1 \leq x^* \leq x_2$ and $y_1 \leq y^* \leq y_2$, then it is proved that the gaze vector crosses the side, and consequently the object's bounding box. Obviously, if z_G is 0, the calculation is not performed and it can be affirmed that the gaze line doesn't cross the front nor the back side of the bounding box.

The same procedure is applied to the other sides, and it is sufficient that the conditions are satisfied for any of the six sides to conclude that the gaze line crosses the 3D bounding box and, for the scopes of this project, the person is looking at the object.

Staring at the camera

The criterion used to determine whether a person is staring at the camera is equivalent to the one used for the front side of a 3D bounding box, where $x_1 = -500$, $x_2 = 500$, $y_1 = -500$, $y_2 = 500$ and obviously $z_1 = 0$. In practice, the condition to be satisfied is that the person's gaze is headed towards a point within 50 cm (horizontally or vertically) from the camera.

$$t_0 = \frac{z_1 - z_E}{z_G} = \frac{0 - z_E}{z_G} = -\frac{z_E}{z_G}$$

$$x_0 = x_E + x_G t_0$$

$$y_0 = y_E + y_G t_0$$

Consequently, the equations needed to evaluate whether the person is staring at the camera are:

$$\begin{cases} -500 \leq x_0 \leq 500 \\ -500 \leq y_0 \leq 500 \end{cases}$$

4.3 System workflow

The system's flow of operations is the following:

1. get a new camera frame, including both RGB and depth data;

2. use the YOLOv8 instance segmentation and pose estimation neural networks to identify objects and persons in the current frame;

3. extract the data used for gaze estimation for all persons and calculate their gaze directions using the gaze estimation neural network;

4. update the SORT subsystem with the bounding boxes and confidence scores of the persons detected in the current frame;

5. update the pose skeleton, bounding box and gaze data related to each person recognized by the SORT subsystem;

6. update the data of the persons not present in the frame;

7. delete from the persons' database those who have not been detected for a certain number of frames;

8. associate eligible detected objects to persons and check whether each person is looking at the robot;

9. compute the attention score of each person;

10. (optional) generate visual and console outputs.

Each step will be now explained in more detail.

4.3.1 RGBD frame acquisition

Frame acquisition is managed by an object of a dedicated class called `RealSenseCameraStreamer`, which makes use of many functions contained in the library `pyrealsense2` developed by Intel for Realsense cameras' management. Frames can be acquired in two ways:

- by reading a `.bag` file containing a previous recording (used especially for development and debugging);

- real-time acquisition.

Frame size is set to 640×480 pixels for both color and depth.

The frame acquisition method acquires the frame only when the RGB and depth frames are aligned, i.e. each RGB "pixel" corresponds to a depth value and vice versa.

The computation of the pixels' 3D positions in space is performed by the `rs2_deproject_pixel_to_point(cam_intrinsics, [px_x,px_y], z)` method of the `pyrealsense2` library, which takes as inputs the pixel coordinates `[px_x, px_y]`, the depth `z` of the given pixel and some intrinsic data used to take into account optical parameters of the camera.

4.3.2 Instance segmentation and pose estimation

After the frame is acquired, its RGB component is passed to the pose estimation and instance segmentation YOLOv8 neural networks to detect objects of interest and persons in the frame.

Instance segmentation

The YOLOv8 models aimed at recognizing objects are trained on the COCO dataset, which include labelling for 80 generic object classes. Most of them, though, are not useful in this project, thus only a small subset of them, listed in Table 4.2, is taken into consideration:

Class name	Default class identifier	Custom class identifier
TV[1]	62	0
Laptop	63	1
Mouse	64	2
Keyboard	66	3
Cell phone	67	4
Book	73	5

[1] Both TVs and monitors are detected.

Table 4.2: Object classes selected for instance segmentation

Segmentation was preferred to object detection because knowing which pixels are actually part of the detected object is fundamental to compute its 3D bounding box, as explained in Algorithm 1 later in this chapter. However, this comes with the cost of a slower inference process.

The output of the neural network is a Python object containing, for each person:

- 2D bounding box (in both XYXY and XYWH formats);
- pixel mask;
- detection confidence index, from 0 to 1;
- class identifier.

Pose estimation

Person detection and pose estimation are performed using the dedicated YOLOv8 neural network, retrained on the COCO-WholeBody data to feature more keypoints, as explained in 4.2.2.

The output of the pose estimation network includes, for each detected item:

- 2D bounding box (in both XYXY and XYWH formats);
- detection confidence index, from 0 to 1;
- XY coordinates and confidence score for each of the 21 keypoints. If any keypoint is not detected, its coordinates are set to `[0, 0]`.

All this information is stored in an instance of a class called `PoseProcessor`, which is also used to perform gaze estimation.

4.3.3 Data production and gaze estimation

The keypoints obtained in the previous step are passed to the method `keypointsToData_21kp` of the `PoseProcessor` class. This method extracts the data listed in 4.2.4 from the face keypoints of each person and converts them into a feature matrix suitable as an input for the gaze estimation neural network.

The gaze estimation neural network, in turn, evaluates the data and generates a 3D gaze vector for each of the persons whose data were extracted. This operation is performed by the `updateNNGaze` method of the `PoseProcessor` class. The computed gaze vectors are then stored in a matrix in which each row represent the gaze vector of a different person, with the exception of those with one or more undetected face keypoints: in that case the gaze vector is assigned to be `[-2, -2, -2]` to indicate that it was impossible to evaluate it.

4.3.4 SORT update

In order to update the SORT subsystem, all 2D bounding boxes of persons with a sufficiently high confidence score (i.e, higher than 0.4) are passed to the `update` method of the `Sort` Python class, which computes which boxes are more likely to

be associated to persons already detected in previous frames (i.e persons already owning an ID). Each of the aforementioned boxes is then given the ID of the best fitting SORT box.

The boxes that were not associated to any ID, on the other hand, are added to the tracking system as new objects and receive new IDs.

If no person is detected in the current frame, of course, no SORT update is executed.

More details about the tracking process can be found in 4.2.3.

4.3.5 Persons database update

The persons' data are stored in a Python dictionary called `persons` in which the keys are the IDs assigned by the SORT subsystem and the values are instances of a custom class called `Person`.

The `Object` class and its derivations

The classes `InanimateObject` (which will be described in more detail later in this chapter) and `Person` derive from the same superclass called `Object`. This class's attributes are listed in Table 4.3.

Attribute name	**Description**
`cat`	YOLOv8 class identifier
`ID`	Unique ID
`className`	YOLOv8 class name
`birthFrame`	Frame in which the object was first detected
`memory`	Number of frames after which coordinates acquired in the current frame are deleted; default is 25
`p1`, `p2`, `dims`	Lists containing coordinates of the top left (`p1`) and bottom right(`p2`) points and the width and height (`dims`) of the 2D bounding box in the latest frames
`p1_3d`, `p2_3d`, `dims_3d`	3-element lists representing, respectively, the top front left and bottom back right points and the width, height and depth of the 3D bounding box in the latest frame
`timeout`	Number of frames since the last detection
`distance`	Latest computed distance of the object from the camera along the global Z axis
`cam_intrinsics`	Intrinsic parameters of the camera, provided by a function in the `pyrealsense2` library and needed to convert RGBD pixels into 3D coordinates using the method mentioned in 4.3.1.
`camHeightFromGround`	Vertical distance of the camera from the ground; if it is unknown, the value is `None`

Table 4.3: `Object` class attributes

In addition to these attributes, that are common to the two subclasses of `Object`, the `Person` class contains some more, listed in Table 4.4.

Attribute name	Description
`relatedObjects`	Dictionary containing object class identifiers as keys and lists of data related to objects as values. Each list contains distance from the object, a boolean that indicates whether the person is looking at the object, a boolean that indicates if the object has been detected in the current frame.
`skeleton`	Boolean; it is set to `True` if some of the person's keypoints were detected in the current frame
`poseSkeleton`	Matrix containing the person's 2D pose keypoints
`poseSkeleton3d`	Matrix containing the person's 3D pose keypoints
`skeletonValidPoints`	List containing the indices of all the keypoints actually detected
`bbox3d`	Boolean; it is set to `True` if the 3D bounding box is well defined
`midpoint_3d`	List containing the X,Y,Z coordinates of the middle of the person's 3D bounding box
`nnGaze`	Gaze vector computed by the gaze estimation neural network; the latest valid gaze vector is stored
`nnGazeTimeout`	Number of frames since the latest successful gaze estimation was performed
`midEyePoint`	3D coordinates of the midpoint of the segment connecting the two pupils. This point is needed to draw the gaze line

Table 4.4: `Person` class exclusive attributes

Person update operations

The following operations are performed on each `Person` present in the frame:

New `Person` object creation If the person's ID is not present in the persons' database, a new instance of the class `Person` is created. All the following operations are included in the class's contructor.

Timeout update Since the person is present in the latest frame (i.e. 0 frames ago), its `timeout` is set to 0.

2D bounding box update The person's bounding box is updated by adding the new values of `p1`, `p2` and `dims` to the respective lists. If the lists have more than 25 (i.e. the value of `memory`) elements, the oldest ones are deleted.

Gaze vector update The person's `nnGaze` vector is updated with the computed gaze vector, if available. Otherwise, the `nnGazeTimeout` variable is increased by 1.

Pose keypoints and 3D bounding box update If at least one pose keypoint was detected by the pose estimation network, the person's `poseSkeleton` and, if possible, `poseSkeleton3d` are updated with the new values, and the `skeleton` variable is set to `True`. The 3D bounding box is updated as well, by computing the new `p1_3d`, `p2_3d`, `dims_3d` values and setting `bbox_3d` to `True`. If the gaze vector was computed, also the `midEyePoint` coordinates are updated, otherwise it is set to `None`.

The 3D bounding box is computed with the procedure described in Algorithm 1. As it is possible to notice, the Y coordinate is managed differently from the other two. The reason is that it is possible to estimate how much vertical space is occupied by a human body even if not all of its keypoints are detected.

Algorithm 1 Person 3D bounding box computation

procedure BBOX3D(poseSkeleton3d,skeletonValidPoints,distanceMtx, camHeightFromGround)

▷ poseSkeleton3d: 21×3 matrix in which each row contains X,Y,Z global coordinates of a specific keypoint (or all zeroes if the keypoint was not detected)

▷ skeletonValidPoints: list of the indices of the valid keypoints

▷ distanceMtx: depth matrix of the current frame

▷ camHeightFromGround: vertical distance of the camera from the ground

if distanceMtx is None **or** skeletonValidPoints is empty **then**

bbox3d ← False ▷ No computation, bounding box not generated

else

x_min, y_min, z_min ← min(poseSkeleton3d[skeletonValidPoints]) column-wise

x_max, y_max, z_max ← max(poseSkeleton3d[skeletonValidPoints]) column-wise

end if

x_mid ← $\frac{1}{2}$(x_min+x_max)

z_mid ← $\frac{1}{2}$(z_min+z_max)

x_size ← 1.1 (z_max-x_min)

z_size ← 1.1 (z_max-z_min)

▷ The 1.1 factor is needed to keep into account the fact that the keypoints do not correspond to the actual body extremities, so the bounding box is usually larger than the one that encloses the keypoints

x_max, x_min ← x_mid ± $\frac{\texttt{x\_size}}{2}$

z_max, z_min ← z_mid ± $\frac{\texttt{z\_size}}{2}$

additional_vals_up ← vector of heights representing difference between supposed total height and supposed distance of each keypoint from the ground

ymin_index ← index of row of poseSkeleton3d containing y_min y_max

y_min ← y_min − additional_vals_up[ymin_index] ▷ The box is made higher to keep into account the fact that the person's height is greater than its uppest keypoint's position

if camHeightFromGround is None **then**

additional_vals_down ← 2000 − additional_vals_up ▷ The person's height is considered equal to 2 m (2000 mm) to keep a good margin and include most cases

ymax_index ← index of row of poseSkeleton3d containing y_max

y_max ← y_max + additional_vals_down[ymax_index]

else

y_max ← camHeightFromGround

end if

y_mid ← $\frac{1}{2}$(y_min+y_max)

bbox3d ← True

end procedure

Calculation of distance from the camera The distance of each person from the camera is computed with the following procedure:

1. If one or more of the torso keypoints (i.e. shoulders and hips, corresponding to keypoints `[5,6,11,12]`) are present, the distance is set equal to their mean distance from the camera;

2. if no torso keypoint was detected, but other keypoints were, the distance is set equal to the mean distance of all computed keypoints;

3. if there is the 2D bounding box but no keypoint is detected, the distance is set equal to the depth value of the center of the bounding box.

4.3.6 Non-present persons' update

The following operations are performed on each of the persons present in the database but not in the current frame:

- the `timeout` and `nnGazeTimeout` variables are increased by 1;

- placeholder values are added to lists `p1`, `p2` and `dims`, to mark that no bounding box for this person was detected in this frame; in addition to this, the oldest data are deleted in order to keep the lists not longer than 25 elements;

- the `Skeleton` boolean is set to `False`, the `poeSkeleton` and `poseSkeleton3d` matrices are filled with zeroes, and the `skeletonValidPoints` list is emptied.

4.3.7 Inactive persons' deletion

After the operations done in the previous two steps, all persons' `timeout` variables are up to date. In order to avoid excessive data accumulation and computational

overload, when a person's `timeout` reaches a certain threshold value that person it deleted from the persons dataset. Its ID will not be reassigned.

4.3.8 Association of objects to persons

In order to compute availability scores, it is necessary to evaluate whether each person is interacting with one or more objects that may represent sources of distraction for them. The following operations are performed:

Preliminary step

Before the current frame's object processing, the booleans that signal the presence in the current frame of each object associated to persons are all set to `False`. If any of the objects is still present its presence boolean will be set to `True` later in the process.

`InanimateObject` instances' creation

An instance of the `InanimateObject` class is created for each of the objects detected with a confidence higher than a certain threshold that depends on the object class. `InanimateObject`, exactly like `Person`, derives from the `Object` superclass described in 4.3.5. Its only exclusive attribute is the boolean variable `mask`, used to mark whether the segmentation pixel mask related to the object exists.

The `InanimateObject` class constructor works performs the following tasks:

Attributes assignment The attributes `cat`, `ID`, `className`, `birthFrame`, `p1`, `p2`, `dims`, `timeout`, inherited from the `Object` superclass,are populated with the data extracted from the neural network's results.

If instance segmentation was used as detection method (and this is always the case in this project) the `mask` boolean is set to `True`. This boolean is needed to check whether the 3D bounding box can be generated.

3D bounding box calculation If `mask` is `True`, the objects' pixel masks can be used to compute the positions and sizes of their 3D bounding boxes. The following procedure is used for each object:

1. A binary mask the same size of the depth frame is created. The pixel belonging to the objects have value 1, the others are set to 0.

2. The mask is applied to the depth frame through element-wise multiplication. As a result, the only pixels with depth values greater than zero are the ones belonging to the object.

3. The indices of all non-zero points of the new depth matrix are stored as an N$\times$2 matrix in which each row corresponds to row and column of a non-zero pixel.

4. The 3D coordinates of each of the pixels stored before are computed using the `rs2_deproject_pixel_to_point` method mentioned in 4.3.1.

5. The set of 3D points is filtered retaining only the points (x_i, y_i, z_i) that satisfy the following inequality:

$$z_i - z_{median} < 600$$

   This filter excludes the points that are too far away from the median depth of the object, because there is a high chance that they have been incorrectly assigned to the object even though they are not part of it and, if they were

retained, they would lead to generating a 3D bounding box way larger than expected in the Z direction.

6. The 3D bounding box is generated by picking the minimum and maximum value of each coordinate among the remaining points.

Assignment of objects to persons

The general principle followed to assign objects to persons is that each object is assigned to the "conceptually nearest" person. "Conceptual distance" is expressed by a numerical value that may be equal or lower than the geographical distance from between the person's and object's 3D bounding boxes.

Objects belonging to some categories(TV/monitor, laptop, cell phone, book) may be assigned to more than one person at a time since one of the ways in which interaction is established with these objects is by simply looking at them, thus even more than one person at a time may be interacting with them.

Objects may also be not assigned to any person, if they are too far away from anyone and no person is interacting with them.

The `distance` parameter of each object from a person does not only take into account geometrical distance, but also some factors that may suggest that the person is interacting with it. In these cases the `distance` value is set to a value between 0.0001 and 0.0003 (which are lower than the minimum geometrical distance detectable by the system) and is used to represent the relatioship between the object and the person and to set a hierarchy between the different kinds of interaction. In order to make sure that interaction is always evaluated with priority with respect to geographical proximity, geometrical distances from pose keypoints are set to 0.001 (which is greater than any "conceptual distance" expressing the presence of

an interaction) if their actual value is lower.

The calculation of the "conceptual distance" takes into account the person's gaze line and the position and distance from the object of some pose keypoints.

TV or monitor The only meaningful way to interact with a TV or monitor is by watching it. For this reason, if the gaze line of the person intersects the 3D bounding box of the TV or monitor, the object's distance from the person is set to 0.0001. In all other cases, the "conceptual distance" equals the geometrical one from the closest pose keypoint.

Laptop When a person is using a laptop, they usually look at it and have one or both hands close to it. This is translated in terms of "conceptual distance" by assigning one of the following values:

- 0.0001 if the person is looking at the laptop (i.e. their gaze line crosses the laptop's bounding box) *and* one or both of their wrists (keypoints 9 and 10) are within 15 cm from the laptop;
- 0.0002 if the person is looking at the laptop but their wrists are either undetected or farther than 15 cm from the laptop;
- 0.0003 if the minimum distance of the laptop from one of the wrist keypoints is lower than 15 cm *and* the person is not looking at the laptop;
- equal to the distance from the closest of the detected keypoints in the other cases.

Book A person is considered to be interacting with a book if they are looking at it and either they have it in hand or it is not too far from the person's eyes. This is translated in terms of "conceptual distance" by assigning the following values:

- 0.0001 if the person is looking at the book (i.e. their gaze line intersects the book's bounding box), the closest of the two eyes is no farther than 50 cm from the book and one of the wrists is within 15 cm from the book;
- 0.0002 if the person is looking at the book and it is no farther than 50 cm from the closest of the two eyes, but there is no wrist within 15 cm from the book;
- equal to the minimum distance from one of the wrist keypoints if at least one of them was detectable *and* the person is not looking at the laptop;
- equal to the distance from the closest of all the detected pose keypoints in the other cases.

Mouse or keyboard The only way of interacting with a mouse or keyboard is by touching it. This is why the "conceptual distance" of these objects from a person is equal to:

- the minimum distance of one of the wrist keypoints, if at least one of them is available;
- the distance from the closest of all the detected keypoints, if both wrists were not detected.

Cell phone There are two main ways in which humans interact with phone cells: by watching the screen and by holding it close to an ear to listen to something. A person that is listening (or even talking) to the phone is usually more distracted than one that is just watching it or holding it in hand. This aspect is reflected in the evaluation of the "conceptual distance" calculation, which is set equal to:

- 0.0001 if the cell phone and one of the ear keypoints (i.e. keypoints 2 and 3 of the pose detection neural network output) are closer than 5 cm;

- 0.0002 if the person is watching the cell phone;

- 0.0003 if the person is holding the cell phone in hand (i.e. if the cell phone is within 15 cm from one of the wrist keypoints) without watching it and without having it close to an ear;

- the minimum distance of one of the wrist keypoints, if at least one of them is available and the previous conditions are not satisfied;

- the distance from the closest of all the detected keypoints, in all cases not listed above.

Gaze towards camera evaluation

If a person is watching the camera, a fictitous object of category 100 is added to their `relatedObjects`.

In order to understand whether this is happening, the gaze line of the person (if present) has to be evaluated. The procedure is explained in 4.2.5.

Removal of old objects

After the objects in the current frame have been assigned to persons, objects that have not been assigned to specific persons for a certain number of frames (10 by default) are deleted from the `relatedObjects` dictionaries of those persons. This is done in order to reduce memory usage and computational costs.

4.3.9 Persons' scores computation

Once the object management is completed, it is possible to use the data related to objects associated to each person to compute each person's interactivity score.

Score as a moving average of partial scores

The overall score is a moving weighted average of the scores computed in the latest n frames, according to this formula:

$$S[k] = \frac{\sum_{i=0}^{n-1} w_i s[k-i]}{\sum_{i=0}^{n-1} w_i}$$

$S[k]$ is the overall score for frame k (the current frame), w_i are predermined weights, and $s[k-i]$ are the partial scores computed in frames $k-i$ (i.e. i frames before the current one) with $i = 0,1,\ldots,n-1$. In other words, a linear combination of the partial scores of the latest n frames is computed. The weighting factors selected are $[1, 0.75, 0.5, 0.3, 0.1]$. This means that the contribution of a frame to the score decreases over time and totally vanishes after $n = 5$ other frames are captured.

If an ID was generated at frame m with $k < m + (5-1)$ (i.e. the ID is less than 5 frames old), the score obviously only takes into account frames from m to k.

It may happen that a person is present in some of the previous $n-1$ frames but not in the current one. In that case, the person's partial scores are preserved, but the person is not taken into account as the best candidate for interaction.

Partial score calculation

Each frame's partial score is a real number in the [0,1] range. A higher score means that it is more likely that the person can interact successfully with the robot. Lower scores are assigned to persons which look more distracted or busy. Each distracting factor is translated into a multiplicative factor for the score, with values lower than 1. This ensures that the score always stays within the [0,1] range. Here the contributions of each object and circumstance to the current frame's partial score are listed:

Groups of persons If a person is close to N others (i.e. the distance from each of them is lower than 1 meter), the former's score is multiplied by 0.9^{N-1}. This contribution is due to the fact that it is likely that persons close to each other require each other's attention.

Interaction with objects As it was explained earlier in this chapter, objects associated to each person have a parameter called "conceptual distance" that is used to describe the way the person is interacting with them or the physical distance from the person.

- If the person is watching a TV/monitor, book, laptop or cell phone, their score is multiplied by 0.6.
- If a person has the phone close to an ear (i.e. the phone's conceptual distance is 0.0001), their score is multiplied by 0.5, while if the person is only holding the phone in hand (i.e. the phone's conceptual distance is 0.0003), the multiplier is 0.75.
- If the person is using a laptop (i.e. the laptop's conceptual distance is 0.0003

or lower) or a keyboard or mouse (i.e. their distance from the person is lower than 15 cm), the score is multiplied by a factor spanning between 0.5 and 0.7 (0.5 if the conceptual distance from a laptop is 0.0001, 0.6 if it is 0.0002, 0.7 if it is 0.0003 or the condition for keyboard or mouse applies).

- if the person is not using an object (i.e. the aforementioned conditions do not apply) but is within a certain distance from the object itself (20 cm for cell phones, 1 m for the other classes), the score is multiplied by 0.85.

Interaction with the camera If a person is watching the camera, all multipliers related to groups and objects used are neglected. In other words, the only multiplier that is taken into account in this case is the one related to the distance from the camera, which will be described in the next paragraph.

Distance from the robot Distance of each person from the robot is taken into account because each person's capability to interact with the robot is higher if the person is closer. The formula that associates a person's distance from the camera in millimeters x to its distance multiplier d is

$$d(x) = \max\left\{0,\ 1 - \left(\frac{0.5^{\frac{1}{3}} \cdot x \cdot c}{5000}\right)^3\right\}$$

The parameter c is equal to 1 for persons who are not watching the camera and 0.625 for persons watching the camera. This difference is used to allow the robot to recognize persons that are quite far away but are looking at it.

The score decreases with the increase of the distance from the camera. It halves at 5 m from the camera and becomes 0 at ≈ 6.3 m for people not watching the camera.

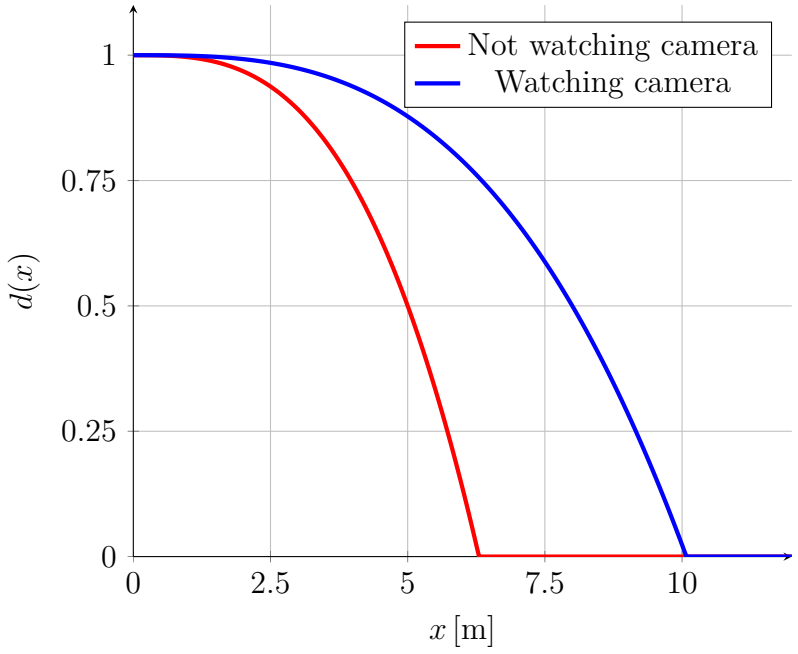


Figure 4.9: Plot of score multiplier due to distance from the camera.

4.3.10 Graphical outputs

Three kinds of graphical output can be provided by the sistem: the original frame with some additions or modifications to show the data produced or extracted, a 3D representation of the persons and objects detected, and charts representing the scores.

In addition to them, some messages are always printed in the Python console when some events (i.e. detection of a new person, computation of the best score, deletion of a person from the database) occur.

Plots on the original frame

YOLOv8 neural networks use built-in functions to plot bounding boxes, pose keypoints and segmentation masks.

In order to make objects recognizable from each other, each object class is

assoociated to a color, which is used to plot the bounding box around the object and (in the case of instance segmentation) to highlight the segmentation masks.

Pose keypoints and gaze Pose keypoints, on the other hand, have different colors even though they belong to the same person. Each color represents, roughly, a different part of the human body: as an example, all face keypoints are light green, while arms and shoulders are blue. Some keypoints are connected to each other in a way that helps identifying them at first glance.

In addition to pose keypoints, it is possible to plot the gaze directions as well, if present. They are represented by blue segments starting from the nose keypoints.

Person bounding boxes All bounding boxes generated have a label in which some data are present. The built-in data available are the name of the object class and the confidence score of the detection. In order to make visible the SORT results, though, two modifications were made:

- the persons' bounding boxes color, instead of being always red, varies depending on each person's ID, and is gray for the person with the highest score;
- the persons' labels contain the SORT ID and the distraction score alongside the "object class" (which is, obviously, 'person' for everyone), while the detection confidence level is not shown.

Object instance segmentation Plotting of object bounding boxes and pixel masks is managed by the YOLOv8 default drawing functions, that assign a color to each object class and use it for both bounding boxes and pixel masks. Pixel masks, of course, are semi-transparent in order not to totally hide the objects themselves.

Depth visualization The depth frame can be converted in a color image using two functions of the `cv2` library, that translates depth values to colors. The color map chosen is `jet` and is shown below.

Figure 4.10: `cv2` `Jet` color map

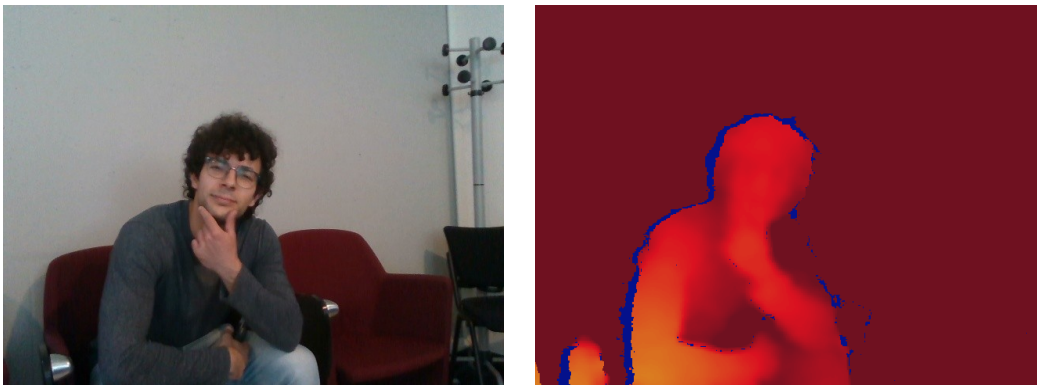


Figure 4.11: RGB and depth frame

3D objects and persons plotting with `matplotlib.pyplot`

The `pyplot` Python library, part of `matplotlib`, allows to generate 3D representations of the objects and persons detected in the frame.

Pose keypoints are plotted using the same colors as in the pose estimation outputs, while for the 'skeletons' the same colors as the SORT bounding boxes are used. The detected gaze lines are plotted in yellow.

The other objects, on the other hand, are represented through their 3D bounding boxes.

The camera tolerance area (the one used to determine whether someone is

staring at the robot, as explained in 4.3.8) is also plotted, and its surface becomes green ifany person in the scene is staring at it.

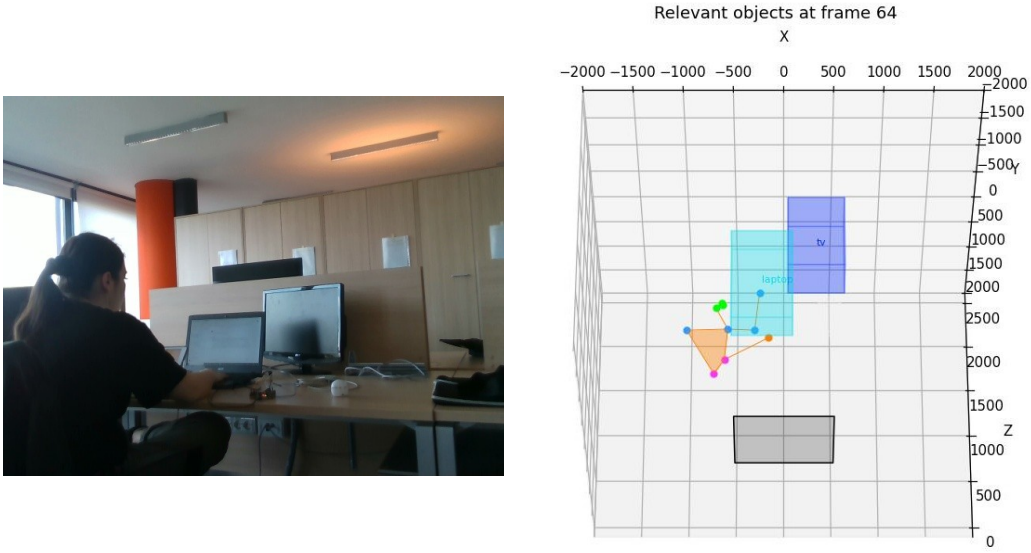


Figure 4.12: Example of 3D scene reconstruction

Scores visualization using `matplotlib.pyplot`

The scores are visualized in histograms in a 2-plot window: the top plot contains the scores of every person who appeared in the latest 5 frames, while the bottom one is a pile histogram in which each contribution to the score reduction is shown for each person who appeared in the latest frame.

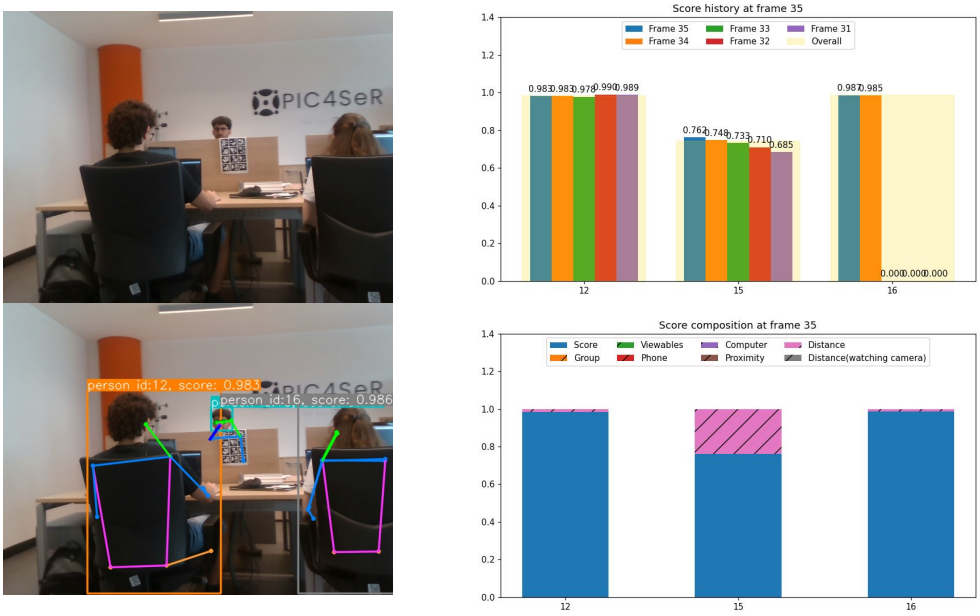


Figure 4.13: Example of score graphs plotting

Chapter 5

System implementation

5.1 Hardware devices used

5.1.1 Intel® NUC

NUC (abbreviation for Next Unit of Computing) is a family of compact computers developed by Intel and then by ASUS. They are characterized by their light weight and smaller physical size with respect to other kinds of computers with similar features and are thus feasible for usage on medium-sized mobile robots requiring computational power comparable to laptops. The model on which development

and testing were performed features an 11th generation Intel® Core™ i5 processor and a 16 GB RAM. There is no dedicted graphics card. The computer runs the Linux Ubuntu 20.04 LTS operating system and can be connected to almost any monitor using one of the two HDMI ports on its back. Four USB and two USB type C ports are present.

5.1.2 Intel® RealSense™ d435

RealSense™ is a family of depth cameras produced by Intel. They allow to record the depth of the scene alongside the RGB representation using technologies that vary depending on the used model. In this project the RealSense™ d435 model was used. It performs depth detection by using stereoscopic technology and allows to detect the distance of pixels with good precision, especially in the distance range $0.3 \div 3$ m. A dedicated application (`realsense-viewer`) is available for basic functional testing, but cannot be easily integrated with Python scripts, so the `pyrealsense2` Python library, which allows to integrate recording and loading functions within the main program, was used in the whole project.

5.2 Software libraries used

5.2.1 NumPy

NumPy is a Python library used for many types of calculations. The main feature used in this project is the native management of multi-dimensional arrays with a fairly simple syntax. Most data used and generated by the system are stored as NumPy arrays.

5.2.2 SORT

SORT[17] is an object tracking algorithm for video streams developed by Alex Bewley et al. that allows to perform real-time multiple object bounding box tracking and identification using a Kalman filter. The algorithm is implemented in the project scripts by importing the classes and methods included in a specific Python file developed by the algorithm producers.

More details can be found in 4.2.3.

5.2.3 OpenCV

OpenCV is a graphic library for computer vision available for many different languages including Python. In this project it is used to visualize the frames captured by the camera (both the RGB and depth data) and plot bounding boxes, labels, skeletons and keypoints on them.

[17] Alex Bewley et al. «Simple online and realtime tracking». In: *2016 IEEE International Conference on Image Processing (ICIP)*. 2016, pp. 3464–3468. DOI: 10.1109/ICIP.2016.7533003.

5.2.4 Ultralytics YOLOv8

YOLO is a family of neural networks used for computer vision tasks, including object and pose detection, image classification and segmentation, developed and maintained by Ultralytics[18].

In this project the segmentation and pose detection networks of the YOLOv8 series were used. In order to do this the `YOLO` sublibrary from the `ultralyitcs` Python library was imported in the project scripts. More details can be found in 4.2.2.

5.2.5 TensorFlow and Keras

TensorFlow is an open source library used for various machine learning tasks, able to optimize multi-dimensional array operations, especially when using GPUs as computing units. Keras is an open source high-level library for neural network development, training and testing. It can use TensorFlow as a backend.

These two libraries were used to implement, train and test the gaze estimation neural network described in 4.2.4.

5.2.6 `matplotlib`

The Python `matplotlib` library, and more specifically its sublibrary `pyplot`, is used by the `Plotter3D` class to plot the 3D representation of the evaluated data and by the `BarGraphs` class to plot bar graphs representing the evolution and composition of persons' scores. More details can be found in 4.3.10.

[18]Glenn Jocher et al. *YOLO by Ultralytics*. Version 8.0.0. Jan. 2023. URL: `https://github.com/ultralytics/ultralytics`.

Chapter 6

System testing

The attention score estimation system presented in the previous chapter was tested using the hardware described in Chapter 5.

The experimental setup consisted of:

- the Realsense camera connected to a computer used to manage and monitor the recordings;
- a hand cart used to move the camera and the computer.

Testing was not performed in real time, but the `bag` files recorded allowed to replay video streams like if they were being produced in real time.

Tests were conducted in four different scenarios:

- Elevator room, to simulate the necessity of asking a person to press the elevator call button;
- crowded waiting room, to simulate a generic interaction request in a crowded environment;

- hallway near a door, to simulate the necessity of asking a person to open the door;
- office open space, to simulate a generic interaction request in an uncrowded environment.

6.1 Samples from tests

Samples from each of the videos will be shown and analyzed below.

6.1.1 Test 1: elevator room

The test was performed in the elevator room of an office building. 6 persons were present: 3 of them were close to each other, and some of them were using their cell phones in certain moments.

In this frame it is possible to observe the effects of grouping, usage of cell phone and distance on the scores.

All 5 persons in the scene are detected and associated to IDs. Persons with IDs 56 and 62 (the leftmost ones) were first detected in this frame, person 61 (near the elevator door) was also detected in the previous frame, person 60 (to the right) has been present for 3 frames, while person 50 (the rightmost one in the 3-person group) has been detected in 4 of the latest 5 frames.

Person 61's gaze was also detected, and from the 3D visualization it is possible to infer that she is looking towards a point above the camera's tolerance area. She is not influenced by any relevant object, thus her score in the current frame is only affected by her distance from the camera.

Persons 50, 56 and 62 are geographically close to each other, thus the grouping

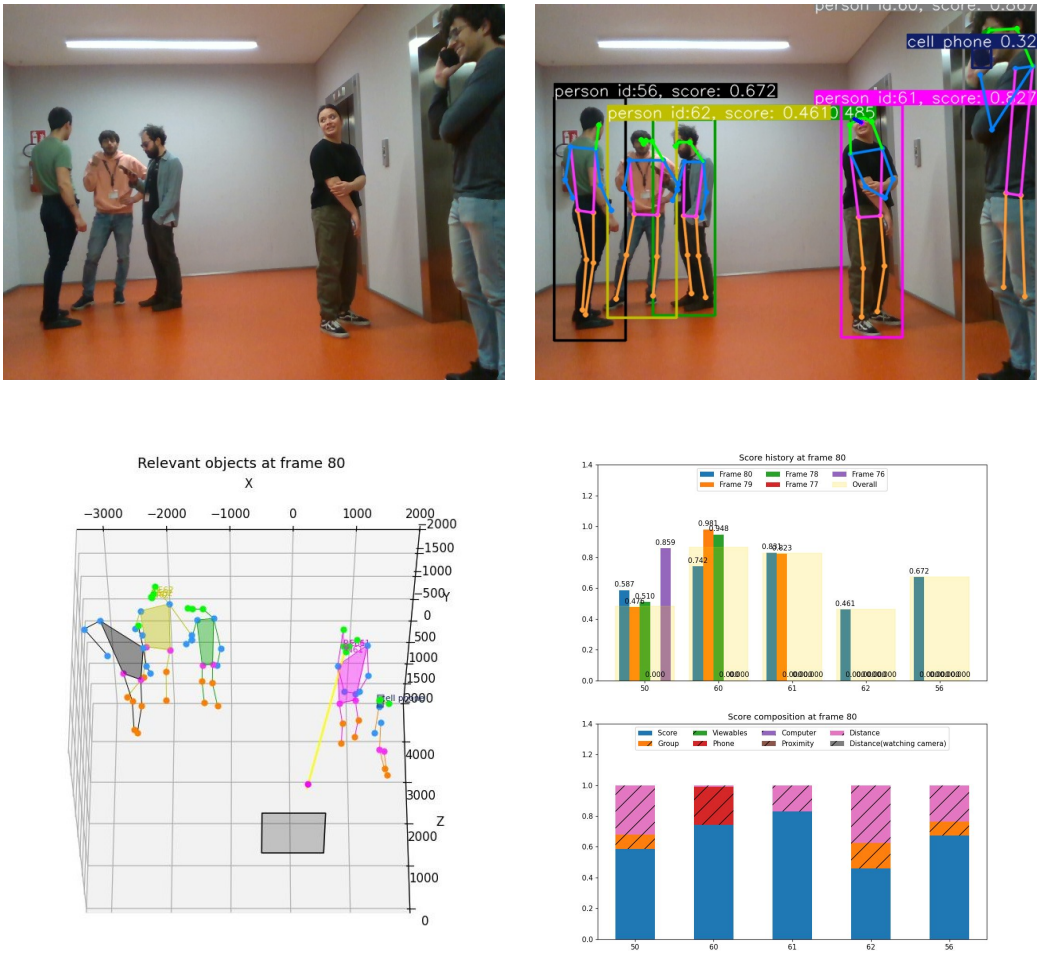


Figure 6.1: Frame taken from test 1

penalty takes effect, as shown by the orange section of the bottom bar graph. It is possible to see that this penalty has a higher impact on person 62, because he is considered close enough to both his neighbors, while persons 50 and 56 are only affected by the closeness to person 62.

Person 60 is not entirely visible in this frame, but he is detected nonetheless. He is clearly the person closest to the camera, and his distance penalty is almost negligible, but the cell phone in his hand and close to the head was detected, thus

the cell phone penalty (in red in the bottom bar graph) takes effect and brings his current frame score to be lower than the one of person 61.

If only the current frame had been taken into account for evaluation, person 61 would be selected as the most suitable for interaction with the robot, but since also some previous frames are taken into account (as explained in 4.3.9) and person 60's scores in the previous frames were high, the end result is that the person selected by the system in this frame is person 60, despite the distraction given by the cell phone in this frame.

6.1.2 Test 2: crowded waiting room

This tests was conducted in a wide area at the end of a corridor, that may resemble the waiting room of an office or a studio. 5 persons were present. One of them was using a laptop, while two others had their cell phones in hand.

In the frame shown below it is possible to observe the effects of grouping and usage of a "viewable" object.

Persons 2,4 and 5 are sitting next to each other and have overlapping bounding boxes; still, the pose estimation network allows to recognize each of them. Since they are close to each other, they are considered to be grouped, and the penalty given by grouping is equal for all 3 of them because each one is close enough to the other two to be considered grouped with both of them.

It can be observed that person 2's facial keypoints are located in the background, outside of the person's face: as an effect, they are considered to be farther from the camera than they actually are and this brings person 2 to be wrongly considered as farther from the camera than person 4, as it is possible to infer from the current frame's scores (0.798 for person 2, 0.799 for person 4).

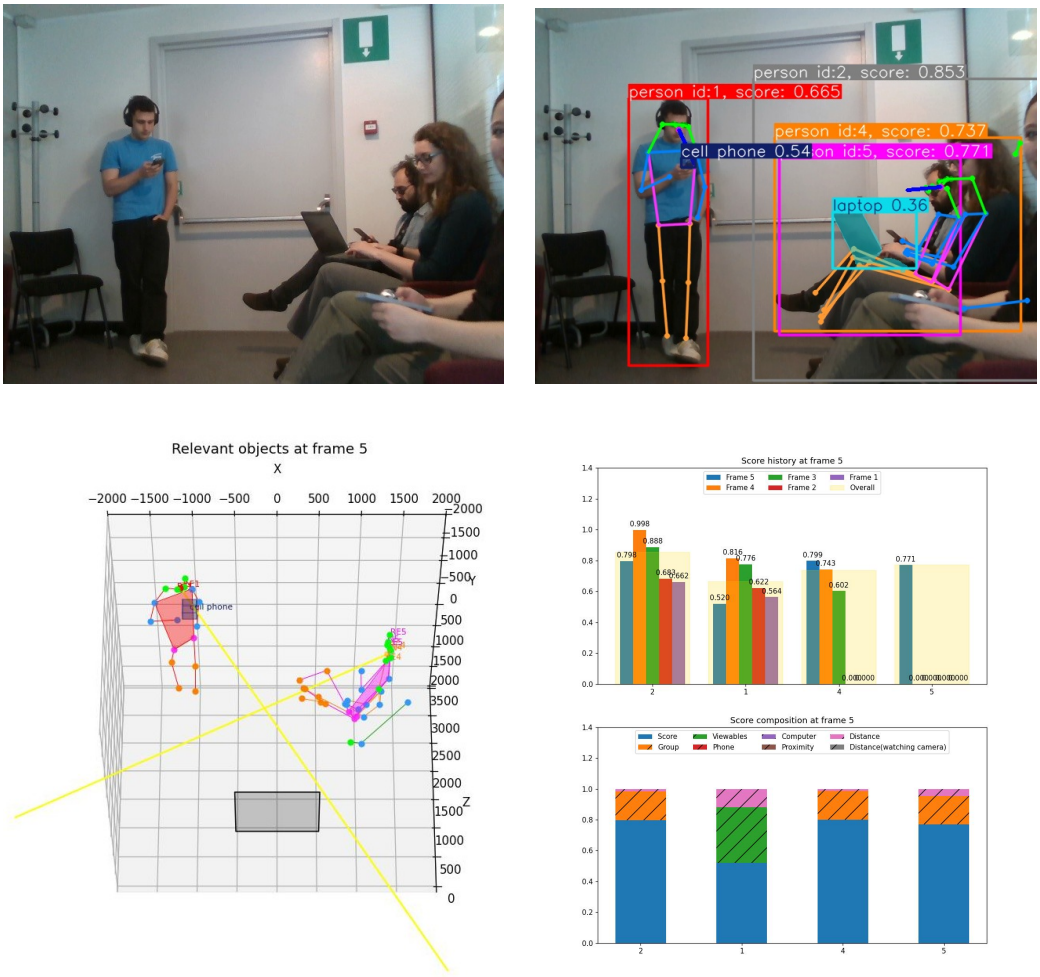


Figure 6.2: Frame taken from test 2

The laptop being used by person 4 is detected, but it is not considered in the scene evaluation because of its low confidence value. Since many false positive detections of TVs/monitors and laptops were observed in other tests, their confidence threshold was set higher than the one used for the other object classes (0.4 for TVs/monitors and laptops, 0.2 for other classes). This, as it is possible to see, unfortunately also cuts off some valid detections.

The cell phones that persons 2 and 5 have in hand are also, unfortunately, not

detected. This may be due, as in the previous example, to their small size in the picture and orientation.

Person 1's distance penalty is lower than the other persons' grouping penalties, but its score is drastically lowered by the fact that his gaze is headed towards the cell phone he has in hand (more precisely, his gaze line crosses the cell phone's 3D bounding box), and this generates a huge penalty (shown in green because the cell phone in this context is considered as a "viewable object", like a monitor, a laptop's screen or a book).

Overall, person 2 is considered as the best candidate for interaction due to her higher scores in the previous frames (in which the keypoints placement error did not occur).

6.1.3 Test 3: hallway

This test was carried out in a hallway, close to a closed door. There was a group of three people talking, and a couple of others moved through the hallway during the test.

In this frame it is possible to observe the effects of high distances from the camera.

All scores are highly affected by the distance from the camera, because all four persons are quite far from the robot. As explained in 4.3.9, the distance penalty grows cubically with respect to the distance between the person and the camera, and persons farther than a certain distance get a partial score of 0.

The three persons in the foreground are also affected by the grouping penalty, but the fact that they are much closer to the camera than person 54 makes sure that person 53, which is the closest of the three in the foreground, is considered the best candidate for the required interaction (in this cases, opening the door to

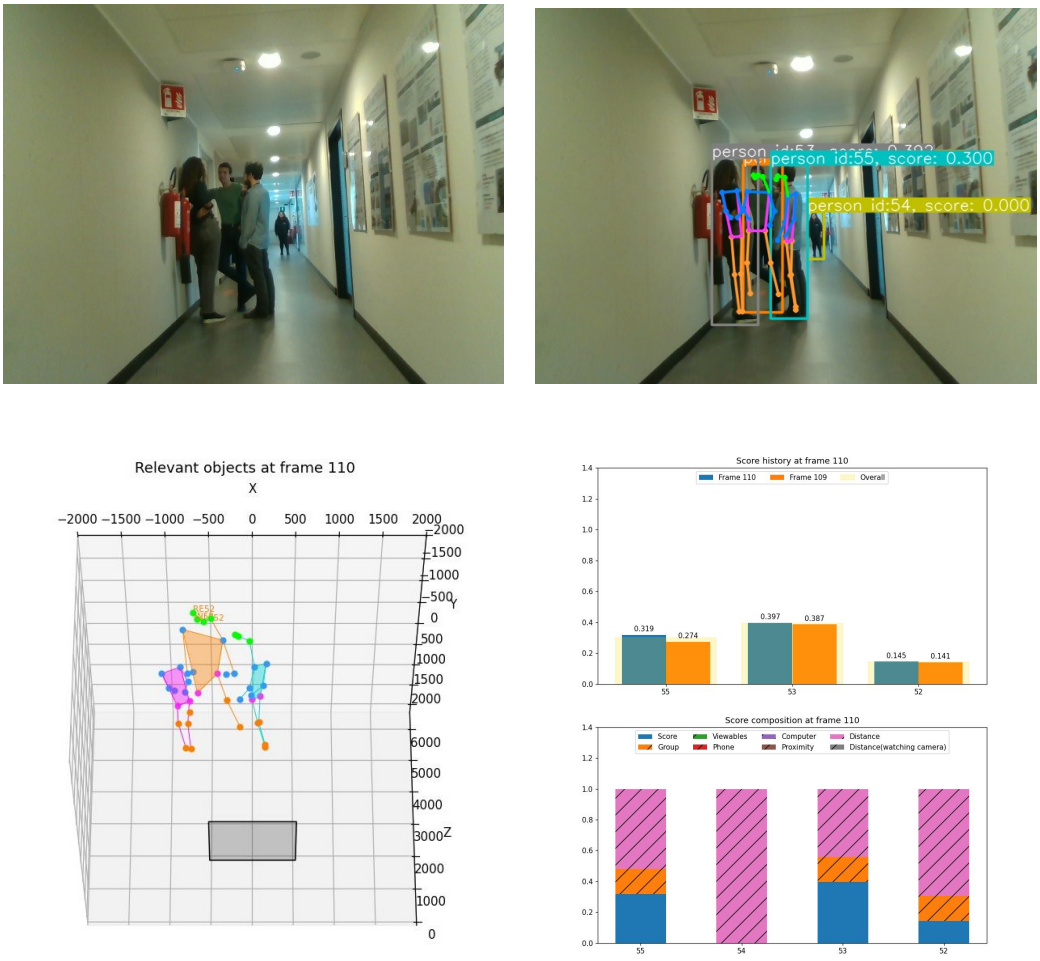


Figure 6.3: Frame taken from test 3

the right).

Person 54 does not appear in the 3D reconstruction because the pose estimation model could not detect its keypoints, probably as an effect of the small apparent size in the frame.

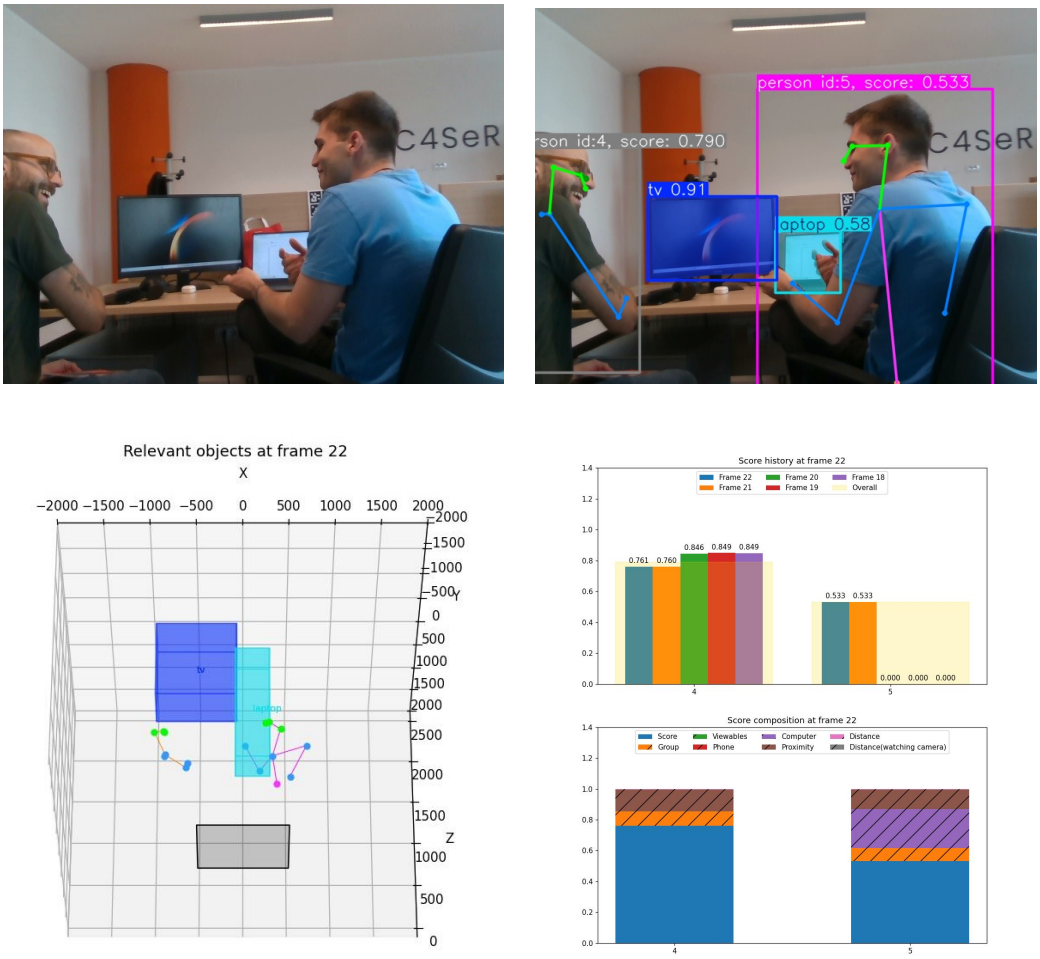


Figure 6.4: Frame taken from test 4

6.1.4 Test 4: office

This test was carried out in a scarcely populated office room. 3 people were present: two were close to each other, while the third one was far from them. Two persons were using laptops and many monitors were present all over the office.

In this frame it is possible to observe the effects of interaction with a computer.

The two persons' distances from the camera are short, thus the effect of distance

on the score is almost invisible but present. If it only depended on the distance from the camera, the best candidate for interaction would be person 5, which is visibly closer to the robot, but it is easy to see that two other factors influence the scores.

The TV highlighted in blue is not being watched by any of the two persons, but is also reasonably close to both, so a "proximity penalty" (as described in 4.3.9) is applied to both persons.

Also, the laptop highlighted in cyan is close to person 5, and in particular to his hands. This is a clear hint that person 5 is using it, thus a relevant "computer penalty" is applied to him.

As a result, the person considered the best for an interaction is person 4, even though he is farther away from the robot than person 5.

6.2 Results

The tests carried out on the system showed that:

- The system, as a whole, is able to run in real time on reasonably powerful computers like the Intel® NUC used for the tests. This may allow its usage on many different types of service robots. The frame rate is stably higher than 1 frame per second, often getting close to 2: the key limiting factors are the times required for neural network inferences and SORT tracking operations.
- The pose estimation neural network works fairly well in most cases, but may face difficulties when the detected persons are far away or just a small portion of the body is shown. In some cases it also happened that a single person was detected twice in the same frame. The re-training needed for the addition of eye corner keypoints did not affect the overall performance of the network.

- The instance segmentation neural network was improved by the specific training on the few classes needed. It is usually able to recognize most objects as it should, with the exception of cell phones in some poses. In some cases "false" positive detections of laptops and TVs/monitors occurred.

- The gaze estimation system works fairly well and allows to recognize with a low error rate the gaze direction in most cases in which both eyes and the nose of a person are visible.

- The SORT subsystem works fairly well when the camera or the subjects do not move too fast. There may be some "ID transfers" (i.e. an ID passes from a person to another one) in cases in which there are persons close to each other and either they or the camera move.

- The score computation system performance depends mainly on the neural networks' accuracy levels. The weights given to the various potential sources of distraction or disturbance allowed, in most cases, to recognize the best candidate for human-robot interaction.

Chapter 7

Conclusion and future work

While the neural networks already perform fairly well, re-training the instance segmentation network on a different dataset with more object classes related to the specific environment in which the robot operates (in this case an office environment) may furtherly improve the performances and allow to recognize a higher number of objects with which humans can interact and, as a consequence, identify different human activities and generate more accurate interaction availability scores.

The tests conducted on the system showed that it behaves as expected in most circumstances, and most of the issues encountered are due to the hardware and software limitations imposed by the experimental setup.

Summing up, it is reasonable to assume that the interaction availability evaluation system presented in this thesis may contribute to improvement of human-robot interactions in indoor environments, and the methods and algorithms used for scene reconstruction and gaze detection may also be useful for other Computer Vision tasks.

Bibliography

[1] Bhaskar Chakraborty, Ognjen Rudovic, and Jordi Gonzalez. «View-invariant human-body detection with extension to human action recognition using component-wise HMM of body parts». In: *2008 8th IEEE International Conference on Automatic Face & Gesture Recognition*. 2008, pp. 1–6. DOI: 10.1109/AFGR.2008.4813302 (cit. on p. 3).

[2] Kishore K. Reddy and Mubarak Shah. «Recognizing 50 human action categories of web videos». In: *Machine Vision and Applications* 24.5 (July 2013), pp. 971–981. ISSN: 1432-1769. DOI: 10.1007/s00138-012-0450-4 (cit. on p. 4).

[3] Li Xia, Bin Sheng, Wen Wu, Lizhuang Ma, and Ping Li. «Accurate gaze tracking from single camera using gabor corner detector». In: *Multimedia Tools and Applications* 75.1 (Jan. 2016). DOI: 10.1007/s11042-014-2288-4 (cit. on p. 5).

[4] Gian Marcialis, Fabio Roli, and Gianluca Fadda. «A novel method for head pose estimation based on the "Vitruvian Man"». In: *International Journal of Machine Learning and Cybernetics* 5 (Feb. 2013), pp. 111–124. DOI: 10.1007/s13042-013-0188-y (cit. on pp. 6, 37).

[5] Leo Breiman. «Random Forests». In: *Machine Learning* 45.1 (Oct. 2001), pp. 5–32. ISSN: 1573-0565. DOI: 10.1023/A:1010933404324 (cit. on p. 16).

[6] Mingxing Tan and Quoc V Le. «EfficientNet: Rethinking model scaling for convolutional Neural Networks». In: (May 2019). DOI: 10.48550/arXiv.1905.11946 (cit. on p. 29).

[7] Golnaz Ghiasi, Tsung-Yi Lin, Ruoming Pang, and Quoc V Le. «NAS-FPN: Learning scalable feature pyramid architecture for object detection». In: (2019). DOI: 10.48550/arXiv.1904.07392 (cit. on p. 29).

[8] Tsung-Yi Lin et al. «Microsoft COCO: Common objects in context». In: (2014). DOI: 10.48550/arXiv.1405.0312 (cit. on p. 29).

[9] Sheng Jin, Lumin Xu, Jin Xu, Can Wang, Wentao Liu, Chen Qian, Wanli Ouyang, and Ping Luo. «Whole-Body Human Pose Estimation in the Wild». In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2020 (cit. on p. 31).

[10] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. «Simple online and realtime tracking». In: *2016 IEEE International Conference on Image Processing (ICIP)*. 2016, pp. 3464–3468. DOI: 10.1109/ICIP.2016.7533003 (cit. on pp. 33, 78).

[11] H. W. Kuhn. «The Hungarian method for the assignment problem». In: *Naval Research Logistics Quarterly* 2.1–2 (Mar. 1955), pp. 83–97. ISSN: 1931-9193. DOI: 10.1002/nav.3800020109. URL: http://dx.doi.org/10.1002/nav.3800020109 (cit. on p. 36).

[12] Nataniel Ruiz, Eunji Chong, and James M. Rehg. «Fine-Grained Head Pose Estimation Without Keypoints». In: (2017). eprint: arXiv:1710.00925 (cit. on p. 37).

[13] Petr Kellnhofer, Adria Recasens, Simon Stent, Wojciech Matusik, and Antonio Torralba. «Gaze360: Physically unconstrained gaze estimation in the wild». In: (Oct. 2019). arXiv: 1910.10088 [cs.CV] (cit. on p. 38).

[14] Bhaavan Goel. «Estimating multiplicative relations in neural networks». In: (2020). DOI: 10.48550/arXiv.2010.15003 (cit. on p. 43).

[15] Diederik P Kingma and Jimmy Ba. «Adam: A method for stochastic optimization». In: (2014). DOI: 10.48550/arXiv.1412.6980 (cit. on p. 46).

[16] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *YOLO by Ultralytics*. Version 8.0.0. Jan. 2023. URL: https://github.com/ultralytics/ultralytics (cit. on p. 79).