POLITECNICO DI TORINO

Master's Degree in Mechatronics Engineering

Master's Degree Thesis

Multi-sensor machine learning based robotic perception applied to human recognition

Supervisors

Prof. Cristiano PREMEBIDA

Prof. Claudio Ettore CASETTI

Candidate

Federico ARDAGNA

April 2023

Abstract

Multi-sensor perception systems, such as machine learning implemented for pedestrian recognition, are frequently employed in safety contexts for autonomous or semi-autonomous vehicles such as mobile robots and self-driving cars. Since the data redundancy is crucial in the field of human safety (or situational awareness involving humans), a multiple sensor system is usually exploited. To do so, the collected data, or information, is then combined in order to obtain a more accurate estimation of the perception system. The robotic platform used in this work is a compact entry-level commercial robot (Jackal UGV by Clearpath) equipped with cameras and a LiDAR mounted onboard the robot. Sensor calibration, and their intrinsic and extrinsic parameters, is part of this project; a particular effort has been done on the calibration of the thermal camera, due to the fact that it cannot detect colors and cannot be set through canonical methodologies. Regarding machine learning, a convolutional neural network (CNN) is firstly trained with a subset of the collected data. Then, the remaining part of the data is used to validate the parameters of the network, testing also the performances in different light conditions. The errors and confusion matrix are then used to evaluate the detecting system and computed thanks to the testing and validation phase of the job. Finally, data from two different cameras are combined using a late-fusion technique and leading to a further improvement in the human detection framework. The final results are promising and can be used for further improvements, for instance the network could be used for real-time detection. In this case, this work can be also seen as a possible starting point for applications related to drive assisted systems, autonomous vehicles, and robotics.

Acknowledgements

Thanks to Professor Premebida and the whole team of the Mechatronics Laboratory in the ISR of the University of Coimbra, for the constant support and for teaching me fundamentals concepts in arguments that were almost new to me. The assistance of the professor started at the beginning of the Erasmus in February 2022 and ended with the last suggestions on the thesis just a few weeks ago. Thank you for your time and patience.

Thanks to Professor Casetti for the help from Turin while I was in Portugal, and for the suggestions and support when I came back to Italy.

Thanks to the University of Coimbra, where the Thesis work was carried out, and a special thanks to the Polytechnic University of Turin, where I spent the rest of the Master's.

Thanks to my family and friends, because without all of you my results would have been unreachable. Thanks for supporting and inspiring me everyday, motivating me to become by best self.

"Se podes olhar, vê. Se podes ver, repara."
"Se puoi vedere, guarda. Se puoi guardare, osserva."
"If you can see, look. If you can look, observe."
José Saramago

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter will present, in a summarized manner, the motivations, main objectives, course of action, technical implementations and contributions to the work carried out in this MSc dissertation in Mechatronic Engineering done at the Instituto de Sistemas e Robótica of the Universidade de Coimbra during my Erasmus experience. The first sections expose the motivations for working on this topic, followed by an introduction to the concepts and methods used. Subsequent sections present the objectives and contributions of this project. Finally, the last section sets out the organizational structure of the thesis.

1.1 Motivation

During the time in which this thesis is written, there is a massive interest in the sector of the autonomous and semi-autonomous mobility. Both public transportation networks and private companies are moving in the direction of a smarter and self-driving future [1]. The main urban areas aim to move forward in becoming smart, and great economic investments are made in the effort to reach this ambitious goal [2]. In this regard, a large list of projects, starting from the past century until our days, that focused their attention to the ITS systems improvement, can be mentioned: NAVLAB (1990s), ARGO (1999), ULTra (2003), CityMobil (2006), DARPA grand challenge, URBAN challenge, 5G-IANA (2021) i4Driving (2022). This dissertation's work is collocated in the context of intelligent systems,

including mobile robots, and intelligent mobility, in which the technologies and the innovations related to the ITS systems are of considerable concern [3], [4].

Since great part of the existing systems in mobile robots and autonomous vehicles share a large number of architecture modules (perception, navigation, trajectory planning, communication, control, localization, obstacle detection, collision alert, object classification, human detection, etc.), it is very common that the research and the experimentation starts indoor, or in controlled environments, with mobile platforms. Projects in the ambit of the systems of perception applied to mobile robotics constitute an area of significance in the last years and are already applied in the field of cars and smart vehicles in real environments [5]. In the latter domain, the main parameters to be supervised are safety related. This is the reason why multi-sensor fusion approaches represent a valuable solution in the field of autonomous driving vehicles. One of the most widely employed sensor in such applications, alongside with cameras, is the LIDAR, which is highly useful in the detection of the surrounding object and eventually able to prevent collisions. In the particular thesis scope, its data is used in combination with different optical sensors, *i.e.* cameras. The redundancy given by the multiple sensor system confers to the vehicle a greater reliability and, as a consequence, the possibility to be applied to real and urban contexts.

In this data-driven world, the application of Machine Learning (ML) algorithms has become highly efficient and almost necessary. Efficient solutions to the problems that involve a large amount of data seems to be difficult and cannot - easily - be found anymore using traditional (non-deep) algorithms. The approach of ML-oriented problem solving, which is based on an inductive strategy philosophy, suits perfectly for this work ambitions; indeed, the adoption of artificial intelligence is nowadays widely spread and the most common way to deal with autonomous driving vehicle's control [6].

1.2 Objectives of the dissertation

The main objective of this dissertation is to give the reader a broad and comprehensive panoramic of the issues of computer vision and machine learning, as applied to the case study in multi-sensory perception for mobile robots.

In particular, the goals are to:

- assemble a sensory suite, using aluminum profile elements, that “fits" the position of the cameras to be mounted on the mobile robotic platform and to design tailored plastic cases to protect the sensors and fix them stably on the profile;
- calibrate the available cameras and sensors correctly, calculating with acceptable accuracy the intrinsic and extrinsic parameters and their relative orientation;
- collect large and varied data sets to feed a convolutional neural network (CNN) for training;
- use convolutional neural networks (CNN) based methods for applying deep learning techniques to human detection based on the collected data;
- explore the fusion data possibilities from different sensors to improve perception and object detection performance.

1.3 Work performed and contributions

The entire work was carried out at the Institute of Systems and Robotics (ISR), located at Polo 2 of the University of Coimbra (PT). In particular, the calibration of the cameras and the assembly of the full apparatus took place at the Mechatronics Laboratory located at floor 0 of the aforementioned institute. The recordings and data collection took place outside of the laboratory, specifically in the corridors of floors 0, 1, 2 and 4. The University’s contribution in providing me with all the tools and sensors needed to carry out this thesis was also crucial and worth mentioning. The performed work was firstly to assemble the apparatus by means of a structure composed of aluminum profiles, made available by the University for the Mechatronics Laboratory. After that, some pieces of plastic material, which were previously designed thanks to the 3D-modeling software SolidWorks, were molded with the function of protecting and housing the sensors. The next and crucial step was to calibrate the sensors. The process took a lot of time and different tries in order to obtain the most accurate estimation of the cameras parameters.

In this precise phase, the presence of the LIDAR sensor was particularly crucial. Once the whole apparatus was assembled and functional the next step was data collection: the mobile platform was remotely guided to explore 4 different floors of the building, while RGB and thermal cameras collected data. Then the data were processed, so that they could be fed into the convolutional neural network (CNN) and the results were collected. Finally, data from the two different optical sensors were fused so that differences and improvements could be analyzed using data fusion techniques from multi-sensor perception systems.

1.4 Organization of the thesis

The dissertation is divided in three main parts. The first two chapters (**Introduction** and **Related work**) and the last one (**Discussion and conclusion**) are mainly discursive and aim to present the work, the scientific context in which the work is located, the discussion about the results and the future possible scenarios. The third chapter (**Computer vision basics and camera calibration**) firstly gives the reader an overview about the computer vision topic and, after that, it technically describes the calibration work done with the four sensors. The fourth chapter (**Dataset, experiments and results**) explains how the data were collected, describes what the work was about in a practical way and depicts the results. In this chapter there is also an important panoramic about the deep learning technique used in the present work. At the end of this thesis the reader can also find three different appendices: A) where the used sensors are briefly described, B) in which there is the full apparatus setup explanation and C) for a deeper comprehension about the point cloud-frame extraction.

Chapter 2

Related work

2.1 Multi-sensor perception systems

The state of art in the multi-sensor perception systems argument is, at the current time, deeply varied and comprehensive. In this field, the main applications in automotive and mobile robots concern human and object detection (and eventually avoidance). Vision modules are able to detect nearby moving objects like pedestrians, bicyclists and vehicles. Modern systems can utilize the visual recognition information to improve a tracking model selection, data association, and movement classification. An example of this can be found in the work by *Cho et al.* [7]. The results are proved to be very robust and reliable, exploiting the fused data coming from two sensors (LIDAR and camera). In this way, it is possible to take as a reference also the paper by *Burlet and Dalla Fontana* [8].

Other works focus their attention to the human recognition, putting an on-point target in the pedestrian detection goal. These are set in an urban environment and simulate the real daily application with mobile robots. One of these is the paper that can be found in the bibliography at index [9], the work by *Premebida et al.* An interesting comparison between two different data fusion architectures is examined: a centralized and a decentralized one; the results highlight that the latter one led to a more valuable performance. Another interesting work is [10], in which the authors *Yan et al.* developed a functioning online transfer learning based on a multisensor system, to the service of a mobile robot with the aim of human detection. The ML

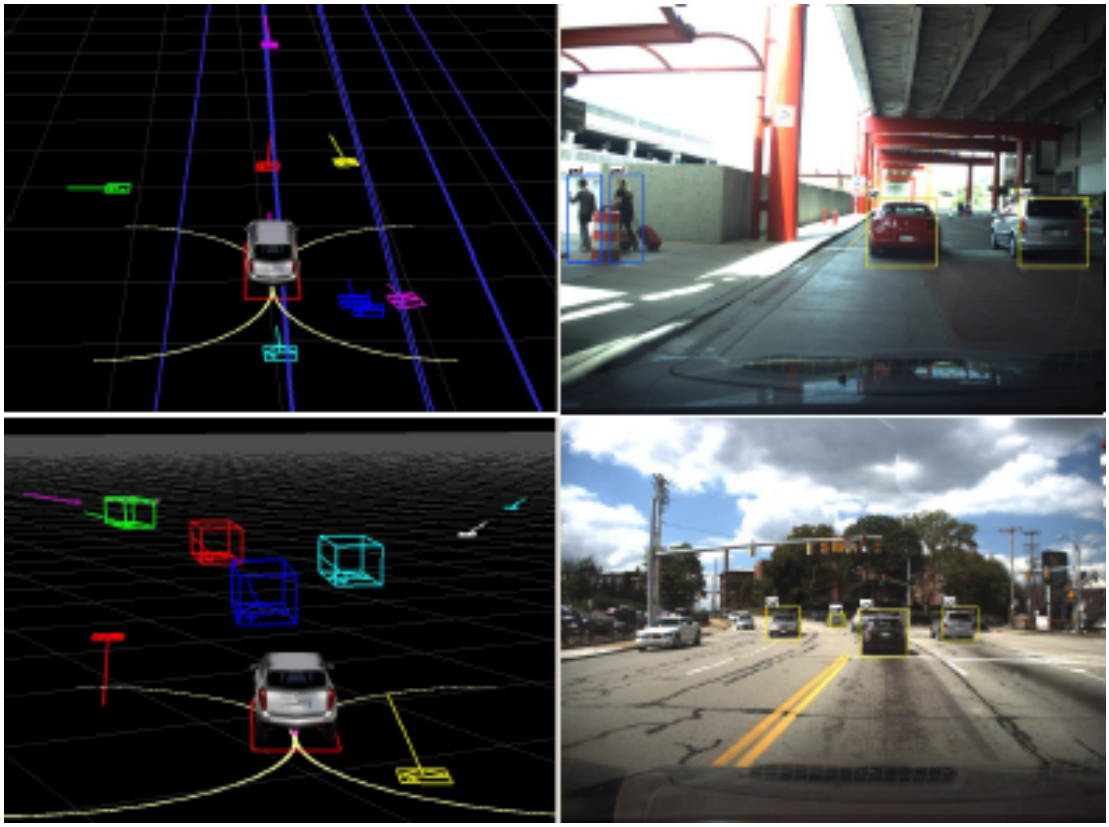


Figure 2.1: Images taken from the article "*A multi-sensor fusion system for moving object detection and tracking in urban driving environments*" by *Cho et al.* [7]

algorithm is trained on a basis of "trajectory probability". The probability is used to determine whether new detection belongs to a human trajectory or not. The tests are conducted in a real-word environment and led to great results.

In [11], the authors *Monteiro et al.* have studied the classification problem applied to different kind of dynamic obstacles: pedestrians and cars. In this case, data of a LRF (Laser Range Finder) are combined with data coming from a monocular camera. Again, the multi-sensor apparatus and the fusion of the sensors data led to a working system for detecting, tracking and classifying objects in outdoor environment.

To have a reference inherent also to automotive applications, the reader can take a look at the work by *Herpel et al.* [12]. In this study, fusion paradigms are

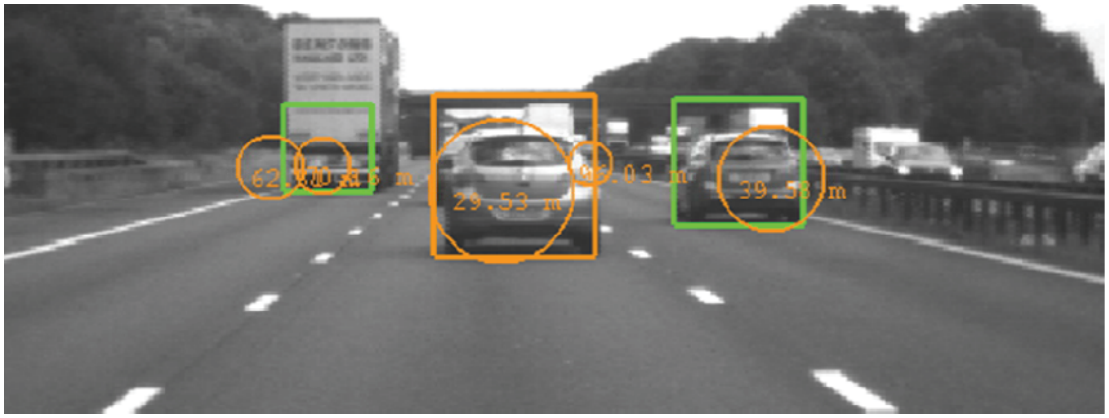
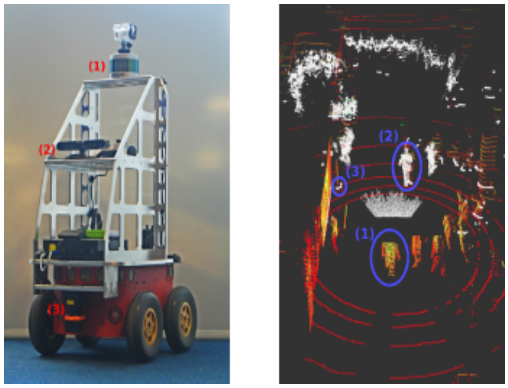


Figure 2.2: Picture depicted in the work by *Burlet and Dalla Fontana*: *"Robust and efficient multi-object detection and tracking for vehicle perception systems using radar and camera sensor fusion"* [8]



(a) Mobile platform and sensor acquisition from the work by *Yan et al.*: *"Multisensor online transfer learning for 3d lidar-based human detection with a mobile robot"*

(b) Detections taken from the work *"Tracking and classification of dynamic obstacles using laser range finder and vision"* by Monteiro et al.

Figure 2.3: Images used as a reference for two different kind of human and object detection systems

used in reliable context perception for ADAS. The simulations results for different settings of traffic scenarios, sensors set and fusion paradigms show advantages of the low-level fusion paradigm in both overall tracking accuracy and false positive detections.

Finally, another interesting work is the one by *Premebida et al.* [13], that shows how to obtain depth maps starting from a 3D-LiDAR input. Depth maps are widely

used in the field of object detection and could result to be very useful for data fusion purposes. The results are verified exploiting the KITTI database, showing very positive performances of the approach introduced in the work.

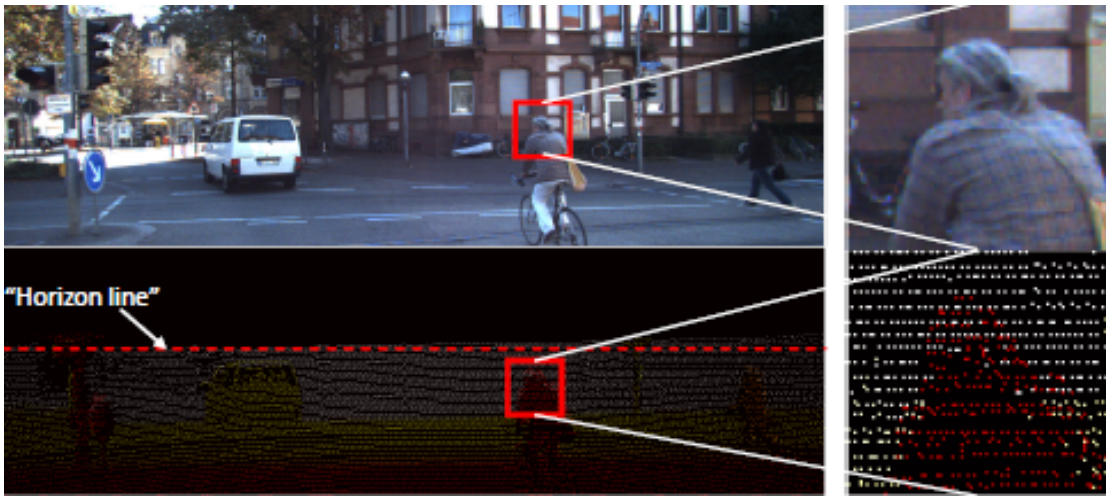
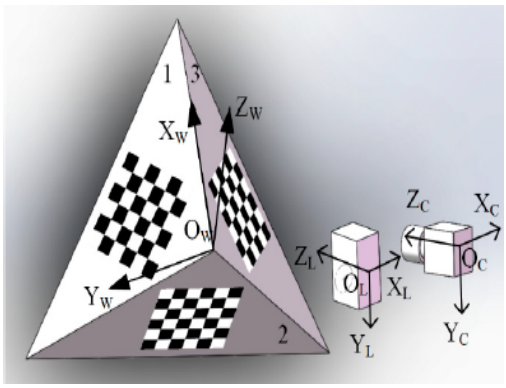


Figure 2.4: Example of depth map from 3d-lidar data, work by *Premebida et al.*: *High-resolution lidar-based depth mapping using Bilateral filter* [13].

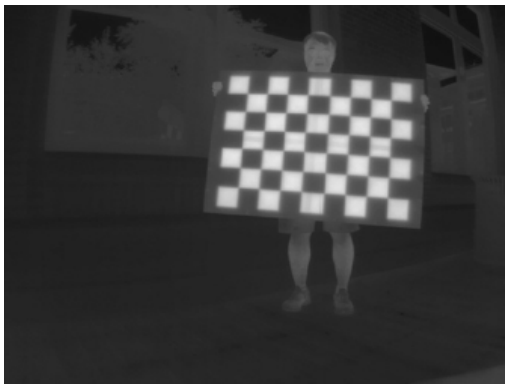
2.2 Sensors calibration

Of the main references in the field of geometric calibration of optical sensors is the work by Zhengyou Zhang [14]. This brilliant paper provides the scientific community with a slim and easy method that returns a closed form solution with a nonlinear refinement based on the maximum likelihood criterion. Almost all algorithms incorporate this method's practical applicability for camera calibration in MATLAB and Python environments, like the one used in this thesis [15].

The state of the art in hybrid (LiDAR-camera) calibration systems is referred in an article from 2021 that can be found in the bibliography at index [16]. It reports an example of calibration using a pyramidal checkerboard pattern. This latter work is taken as reference for the work developed in this thesis; in particular because the lidar model differs only in the number of channels (64 *vs* 16) from the equipment used in this thesis.



(a) Working principle schematic of the work by *Bu et al.*: *"Calibration of camera and flash Lidar system with a triangular pyramid target" [16]*



(b) Image taken from the thermal camera calibration process in the work by *Liu et al.*: *"Multiple methods of geometric calibration of thermal camera and a method of extracting thermal calibration feature points"* [17]

Figure 2.5: Two of the works taken as a reference for camera calibration.

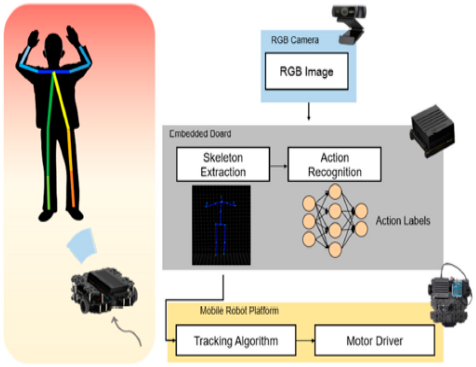
One of the few papers that treats the argument of FLIR camera (*i.e.* long-wave infrared) calibration is the one written by Ruixuan Liu, Hengrui Zhang and Sebastian Scherer in 2018 [17] and can be considered as the only one which offers three easy-to-follow and practical methods to calibrate a thermal camera and that overcome the issue of the non-optical sensor geometric calibration.

2.3 Machine learning applied to human detection using mobile robotic platform

In the field of experimentation, it is common to use a mobile robot and mount on it a multi-sensor perception system. The presented thesis follows this method and takes as a reference different works in this sense. The first one is the article *"A Survey of Deep Learning Techniques for Mobile Robot Applications"* [18], where *Shabbir and Anwer* present a discussion of the applications, gains, and obstacles to deep learning in comparison to physical robotic systems. In the end the research-survey will show the shortcomings and solutions to mitigate them in addition to discussion of the future trends. Furthermore, *Lewandowski et al.* in the work *"A Fast and Robust 3D Person Detector and Posture Estimator for Mobile Robotic*

Applications" [19] depict a solution for human detection and posture classification in the field of mobile platforms. In that project the robot is physically placed in a challenging environment from the detection point of view, which is a supermarket. For the purposes of the mentioned work, it has been used the depth sensor Kinect2.

(a) Frame extraction from the work by *Lewandowski et al.* showing the mobile platform perception system at work, detecting two people and their different pose [19].



(b) Summary schematic of the system depicted in the work by *Lee and An*: *"Real-Time Human Action Recognition with a Low-Cost RGB Camera and Mobile Robot Platform"* [20].

Figure 2.6: Pictures from works based on a mobile platform multi-sensor perception system.

In the work by *Spiess et al.* [21] a combination of RGB camera and depth sensor is used to detect human people silhouettes. The results of the project can be found in Figure 2.7 and show how the sensor data augmentation process leads to a remarkable improvement in terms of accuracy.

Lighting Cond.: / CNN trained on:	Day	Evening	Night
RGB	99.0 %	97.99 %	50.09 %
RGB+Aug	98.99 %	98.98 %	92.22 %
Depth	98.99 %	98.99 %	98.99 %
Depth+Aug	99 %	99 %	99 %

Figure 2.7: Results in different weather and light conditions taken from the work *"People Detection with Depth Silhouettes and Convolutional Neural Networks on a Mobile Robot"* [21].

In the work published by *Lee and Ahn* [20], the authors have used some frameworks to implement a skeleton-like structure extraction from RGB camera images. This is done to improve the computational expense for the human detection and applied to a mobile platform perception system. The work represents an innovative way to approach the human detection problem and the results are particularly promising and remarkable. A schematic of the working system is depicted in Figure 2.6b.

Chapter 3

Calibration of the sensors

3.1 Introduction to camera calibration

To correct lens distortion, quantify the size of an object in world units, and determine the location of a camera in the picture, optical sensors have to be calibrated. Some of the modern applications in existing computer vision applications demand geometry calibration because they require an accurate translation map from 3-dimensional object points to 2-dimensional picture points in order to function effectively. The process of camera calibration is aimed to estimate a large number of parameters of a physical camera. They can be computed through the comparison of some key frames captured by the optical sensor, knowing the real dimension of the objects depicted in those pictures. Usually the objects used to calibrate cameras are checkerboards, since they have very sharp black and white edges, and a regular shape.

Measurements on the images taken from the cameras, with reference to another camera or a LiDAR, which are the main part of the data collection section of this project, can be done after a successful calibration; moreover, since this work is based on a multi-sensor system, it is necessary that each of the sensors gets calibrated after being fixed on the aluminium support attached to the mobile robot platform. This is due to the fact that the parameters to be extrapolated from the calibration include the ones that define the orientation of the camera with reference to a given system of coordinates; so, this orientation of the cameras must

be fixed before the calibration and never changed during the whole usage of the sensors - otherwise the system has to be calibrated again. Consequently, if any change is made to the mechanical setting of the system, then a re-calibration will be necessary. In addiction to this, each camera presents some peculiar internal parameters that are due to:

- differences in lens position (focal length, optical center, aperture)
- optical elements in the light path such as protective filters for the sensor
- signal transfer and A/D conversion

These factors impact on the image and must be taken into account in the reconstruction of the 3D object starting from the picture of it. To do so, during the camera calibration it is also important to extract a set of parameters that are related to the internal structure of each sensor (the intrinsic parameters).

Therefore, it is possible to claim that after a successful calibration both the exact orientation and position of each camera in the so called *world reference system* and the internal (intrinsic and extrinsic) parameters of the cameras will be known.

3.2 Forward imaging model

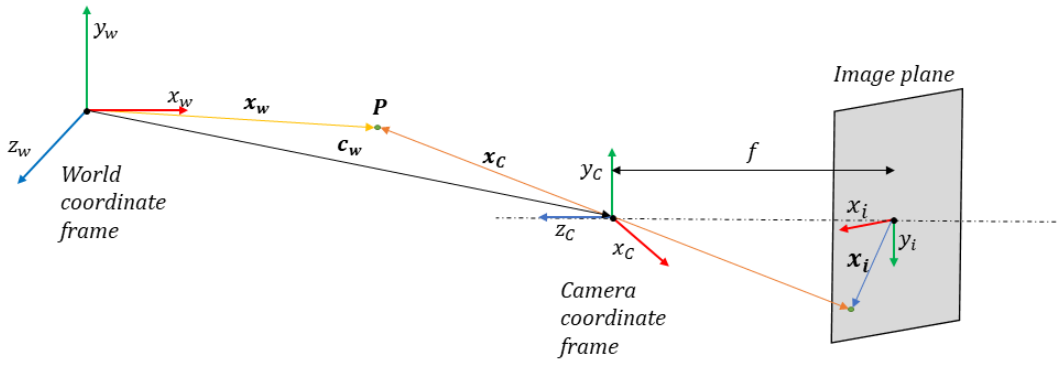


Figure 3.1: Representation of world, camera and image coordinates.

The model to be described in this section is used to map points from the 3D real world dimension to the 2D image plane. The world coordinate system is shown

in the Fig.3.1 and marked with the subscript w. The attention will be focused on a single point of the real world for simplicity (*i.e.* point $\mathbf{P}$). The camera is defined by the camera coordinate frame denoted by the subscript c. The z_c axis of the frame is aligned with the optical axis of the camera. The focal length f is the distance between the effective central projection and the image plane of the camera. Lastly, the image coordinate frame is defined by the subscript i, and is a 2D coordinate system located in the image plane.

Knowing the position and orientation of the camera coordinate frame with respect to (w.r.t.) the world coordinate frame, it is possible to write an expression that converts the point $\mathbf{P}$ in the world coordinate frame to its projection $\mathbf{x}_i$ in the image plane. The complete mapping is called forward imaging model.

This process will be decomposed in two sub-problems: to map from the world coordinate frame to the camera coordinate frame and to map from the camera coordinate frame to the image plane.

3.2.1 From the world coordinate frame to the camera coordinate frame

The aim of this subsection is to describe the steps of transforming the $\mathbf{x}_w$ coordinates of point P to the $\mathbf{x}_c$ coordinates in the camera reference system through a coordinates transformation:

$$\mathbf{x}_w = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} \qquad \longrightarrow \qquad \mathbf{x}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}.$$

To reach this objective it is indispensable to know the position and the orientation of the camera w.r.t. the world coordinate frame. The position is given by the vector $\mathbf{c_w}$ in Fig. 3.1. The orientation is given by a rotational 3×3 matrix $\mathbf{R}$. Those two elements (which include 12 parameters) are the unknowns in our problem and constitute the **external parameters** of the camera.

Knowing them, the camera-centric location of the point $\mathbf{P}$ in the world coordinate frame is:

$$\mathbf{x_c} = R(\mathbf{x_w} - \mathbf{c_w}) = R\mathbf{x_w} - R\mathbf{c_w} = R\mathbf{x_w} + \mathbf{t} \qquad \mathbf{t} = -R\mathbf{c_w}$$

which can also be written in matrix form:

$$\mathbf{x_c} = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

and, finally in homogeneous coordinates form:

$$\tilde{\mathbf{x}}_\mathbf{c} = \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

where the 4×4 matrix is called **extrinsic matrix** $\mathbf{M}_{ext}$. For the definition of homogeneous coordinates the reader can refer to Subsection 3.2.2. Finally the following equation sums up the conclusion that was achieved:

$$\tilde{\mathbf{x}}_\mathbf{c} = \mathbf{M_{ext}}\tilde{\mathbf{x}}_\mathbf{w} = \begin{bmatrix} \mathbf{R}_{3\times3} & \mathbf{t} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} \tilde{\mathbf{x}}_\mathbf{w}$$

3.2.2 From the camera coordinate frame to the image coordinate frame

The aim of this subsection is to show how we can transform from the $\mathbf{x}_c$ coordinates of point P in the camera reference frame to the $\mathbf{x}_i$ image coordinates.

$$\mathbf{x}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \qquad \longrightarrow \qquad \mathbf{x}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

Once $\mathbf{x}_c$ is computed, it is possible to apply the perspective projection to end up with the 2D coordinates in the image plane. The projection equations are reported in (3.1) and (3.2).

$$\frac{x_i}{f} = \frac{x_c}{z_c} \qquad \longrightarrow \qquad x_i = f\frac{x_c}{z_c} \tag{3.1}$$

$$\frac{y_i}{f} = \frac{y_c}{z_c} \qquad \longrightarrow \qquad y_i = f\frac{y_c}{z_c} \tag{3.2}$$

A further step is necessary to end up with the sensor output. In fact, the point in the camera image has coordinates in pixels, not in millimeters. So, the actual image coordinates in pixels will be given by:

$$u = m_x x_i = m_x f\frac{x_c}{z_c}$$

$$v = m_y y_i = m_y f\frac{y_c}{z_c}$$

where m_x and m_y are the pixel densities along the x and y axis defined in pixels/mm. Those parameters are unknown and their computation is a part of the calibration problem. Following the completion of this part, it is feasible to state that the coordinates in pixel of the projected point $\mathbf{P}$ in the image plane are $\mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix}$. The assumption that the origin of the axis coincides with the optical center of the image cannot be given for granted and is not true in general. The point in which the optical axis pierces the image plane is called *principal point* and its coordinates o_x and o_y are unknown. Differently from that point, usually the origin is placed on the top-left corner of the sensor image (for indexing reasons). Said that, the new equations to compute the point $\mathbf{P}$ pixel coordinates in the image plane are the followings:

$$u = m_x f\frac{x_c}{z_c} + o_x$$

$$v = m_y f\frac{y_c}{z_c} + o_y$$

Usually the two unknowns m_x and f are combined together in a single parameter f_x, just as m_y and f in f_y, being called focal lengths in pixels along the x and y directions. The equations become

$$u = f_x\frac{x_c}{z_c} + o_x \tag{3.3}$$

$$v = f_y\frac{y_c}{z_c} + o_y \tag{3.4}$$

where the unknowns are (f_x, f_y, o_x, o_y) and are called **intrinsic parameters** of the camera, representing the internal geometry of the camera.

Since the previous equations are nonlinear (the term z_c is at the denominator) it is usual to linearize them exploiting the *homogeneous coordinates*. The homogeneous representation of a 2D point $\mathbf{u}$=(u,v) is a 3D point $\tilde{\mathbf{u}}$=($\tilde{u}$,$\tilde{v}$,$\tilde{w}$) where the third coordinate is fictitious such that $u = \frac{\tilde{u}}{\tilde{w}}$ and $v = \frac{\tilde{v}}{\tilde{w}}$. So all the points that belong to the line starting from the origin and crossing the point $\mathbf{u}$=(u,v,1) in the 3D space can be considered as homogeneous representation of the same 2D point $\mathbf{u}$=(u,v). This leads to:

$$\mathbf{u} \equiv \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{w}u \\ \tilde{w}v \\ \tilde{w} \end{bmatrix} \equiv \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \tilde{\mathbf{u}}$$

for every $\tilde{w} \neq 0$. If we choose a convenient value for $\tilde{w}$ these considerations can be used to linearize Eq. (3.3) and Eq. (3.4):

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} \equiv \begin{bmatrix} \tilde{w}u \\ \tilde{w}v \\ \tilde{w} \end{bmatrix} \equiv \begin{bmatrix} z_c u \\ z_c v \\ z_c \end{bmatrix} = \begin{bmatrix} f_x x_c + z_c o_x \\ f_y y_c + z_c o_y \\ z_c \end{bmatrix}$$

Finally it is possible to convert this in a multiplication between a 3×4 matrix which includes the intrinsic values and the homogeneous coordinates of the point in camera frame, leading to:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} z_c u \\ z_c v \\ z_c \end{bmatrix} = \begin{bmatrix} f_x x_c + z_c o_x \\ f_y y_c + z_c o_y \\ z_c \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

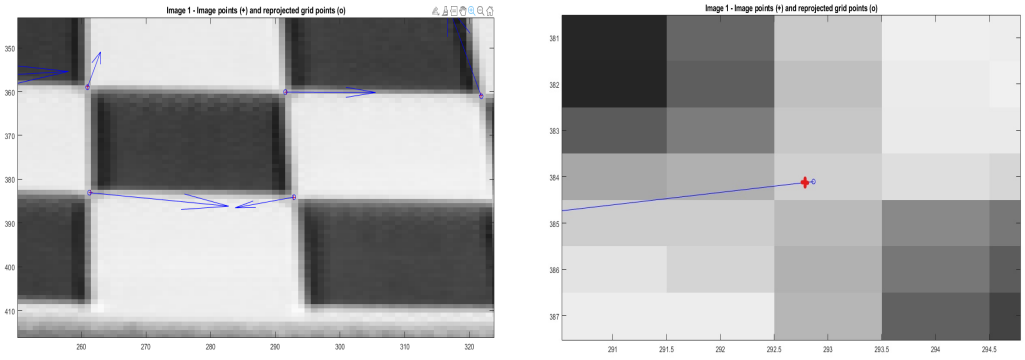
The matrix is called **intrinsic matrix** $\mathbf{M}_{int}$ and includes all of the intrinsic parameters of the camera. After all these considerations it is possible to conclude this subsection with the following equation:

$$\tilde{\mathbf{u}} = \mathbf{M_{int}} \tilde{\mathbf{x}}_\mathbf{c}$$

3.2.3 Errors in camera calibration

The main source of accuracy loss in single camera calibration comes from the so-called *reprojection error*, which is the distance between a pattern keypoint detected in a calibration image and a corresponding world point projected into the same image. This error depends on the quality of the camera calibration and the quality of the marked point on the images. It should have a value smaller than 1 pixel but, in some circumstances, values that are a little bigger than 1 pixel can be considered acceptable.

The mean re-projection error is a number that indicates the quality of the calibration. In the following images there is a graphical representation of it.



(a) **(b)**

Figure 3.2: Original and re-projected points visualization.

In Fig. 3.2 the image points are marked with a cross and the reprojected ones with a circle. If they are very far one from each other, the calibration was not very good and in some cases needs to be done again, maybe choosing better images and improving the pose and the distance from the camera of the checkerboard. In the calibration images taken as example in the Fig. 3.2 the process was very accurate and lead to a mean reprojection error of 0.1636 pixels. It's a very impressive result picked from the MATLAB documentation. The results presented in this work are close to the one seen in this example except for the thermal camera for reasons that will be explained in Chap. 3.5. In the majority of the cameras, including the ones used in this work, the focal length and the shutter time can be set. These

parameters contribute to the success of the calibration, thus in case of problems with too significant reprojection error it is suggested to set them better in order to get higher image quality.

3.3 RGB camera calibration

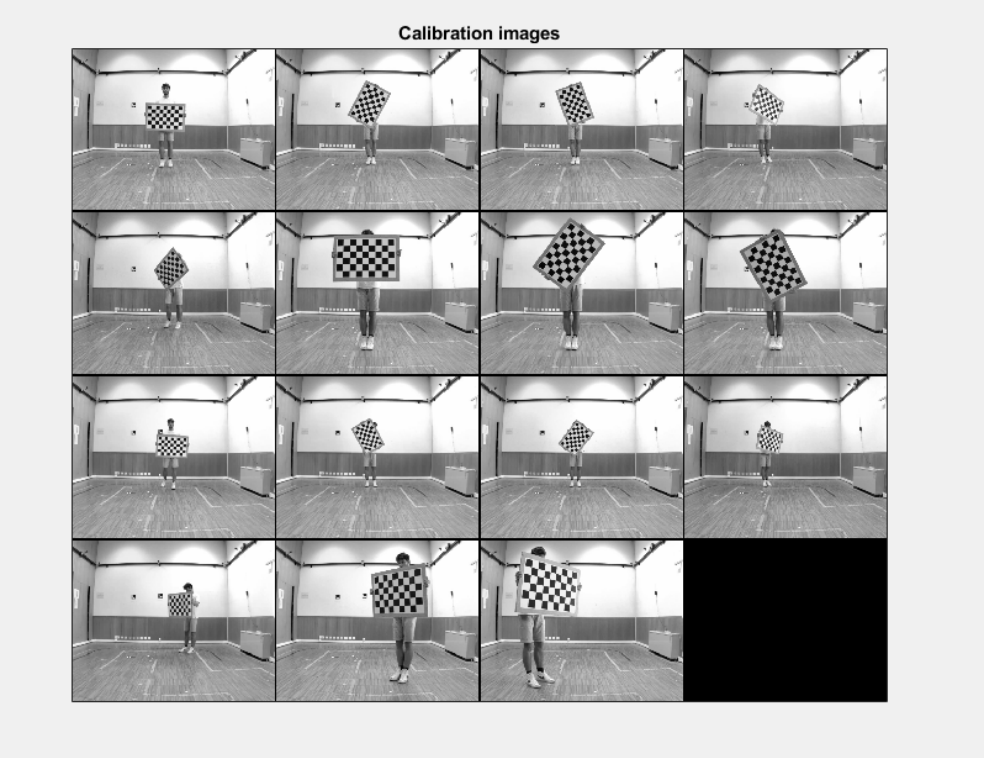


Figure 3.3: Set of collected frames (the images are visualized in black and white).

After correctly collected the dataset made of 15 relevant frames (Fig.3.3), the calibration can start based on the MATLAB function *estimateCameraParameters* in the following way:

```
% Create a set of calibration images.
images = imageDatastore(fullfile('C:\Users\feder\Desktop\Dissertation
    \01-Calibration\calib rgb\img'));
imageFileNames = images.Files;

% Detect calibration pattern.
```

```
[imagePoints, boardSize] = detectCheckerboardPoints(imageFileNames);

% Generate world coordinates of the corners of the squares.
squareSize = 81.4; % millimeters
worldPoints = generateCheckerboardPoints(boardSize, squareSize);

% Calibrate the camera.
I = readimage(images, 1);
imageSize = [size(I, 1), size(I, 2)];
[params, ~, estimationErrors] = estimateCameraParameters(imagePoints,
    worldPoints, 'ImageSize', imageSize);

%extr visualization
figure(1);
showExtrinsics(params, 'CameraCentric');
figure(2);
showExtrinsics(params, 'PatternCentric');
figure(3);
showReprojectionErrors(params);

%save
save('calib_data');
```

The results are saved in the "calib_data.mat" file and the extrinsic parameters are visualized in two different modes, which are camera-centred (Fig.3.4) and board-centred. Moreover, a chart showing the reprojection error for each image will be also visualized. Thanks to this, some images can be suppressed and not used in order to lower the average error (see Fig.3.5).

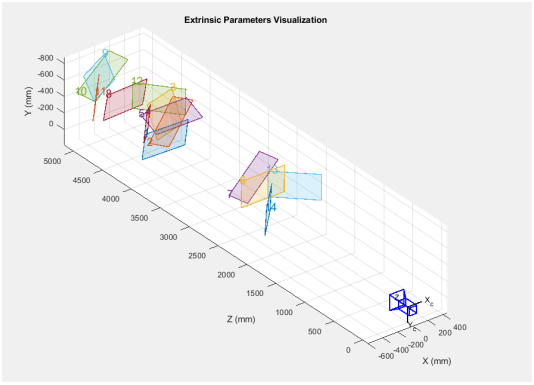


Figure 3.4: Camera-centered visualization of extrinsic parameters

Focal length	1180.2
	1180.3
Principal point	633.01
	511.59
Image size	1024
	1280

Table 3.1: Intrinsic parameters of RGB camera

The intrinsic parameters which are of the focal length, principal point and image size will be reported in the Table 3.1. Another way to visualize the reprojection

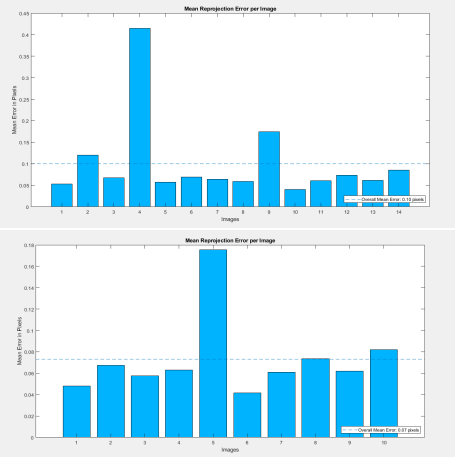


Figure 3.5: Reprojection errors before and after images suppression

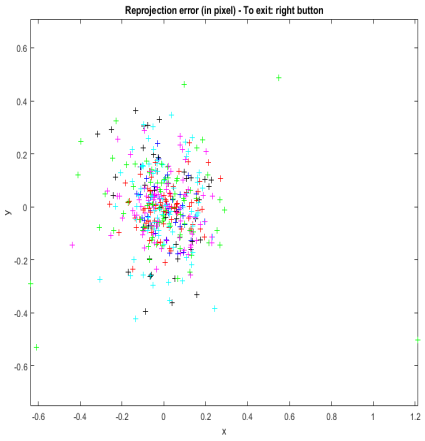


Figure 3.6: Scatter plot reprojection errors

errors is using a scatter plot like the one shown in the Fig.3.6, in which every different color represents a different image and every single point represents a reprojected point. To do this, the previous MATLAB command has been used to set an additional parameter: *showReprojectionErrors(params,'ScatterPlot')* . Thanks to the suppression of the worst images for the calibration a final value of average reprojection error of 0.0772 was obtained and can be considered acceptable.

3.4 NIR camera calibration

Since RGB and NIR cameras are very similar the procedure is identical and the expected results should be comparable in terms of errors. The script used is the same as in the Chapter 3.3 and the results are reported here. The NIR calibration dataset can be found in Fig. 3.7.

The dataset has a reduced number of images with respect to the previous case, thus for our purpose it is not necessary to have a very accurate calibration, and considering the results from the RGB camera calibration, it is possible to settle for a rougher calibration. This choice was made also to improve the computing time.

For this reason there was no need to suppress any image, and the results confirm this assertion, returning a still acceptable final value of mean reprojection error equal to 0.1411 px, as can be seen in Fig. 3.8 and in Fig. 3.9.

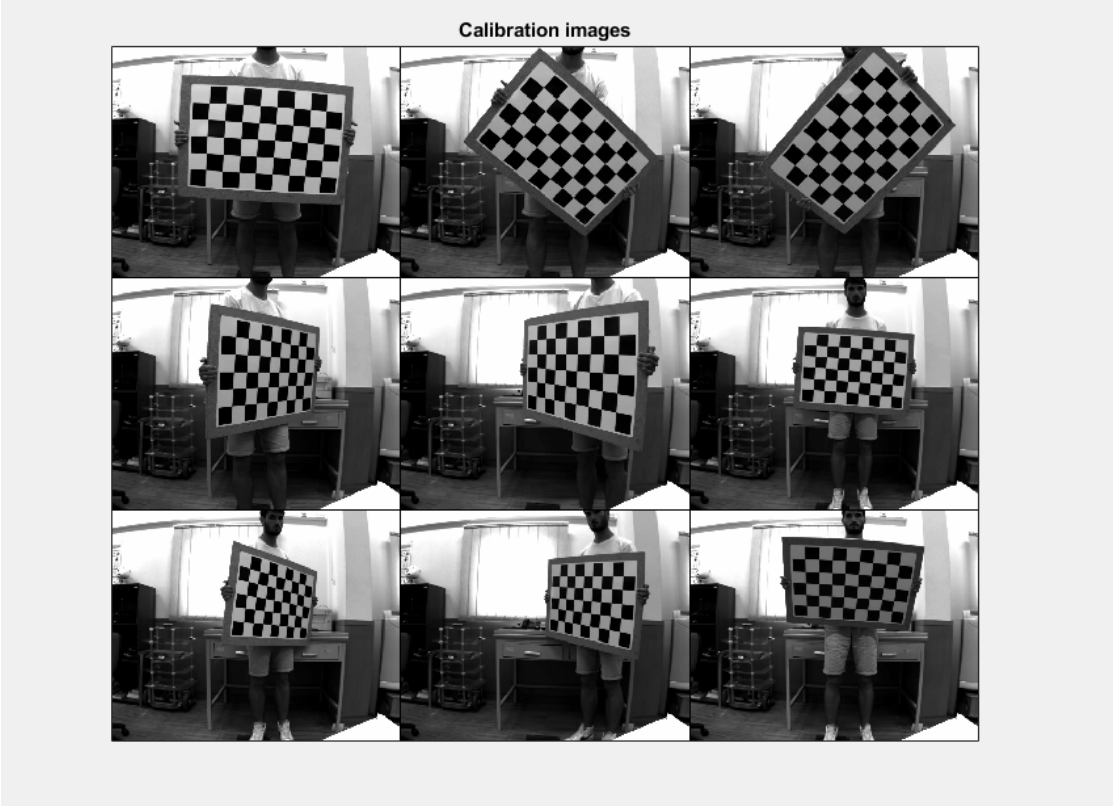


Figure 3.7: Dataset of 9 frames used for the NIR camera calibration.

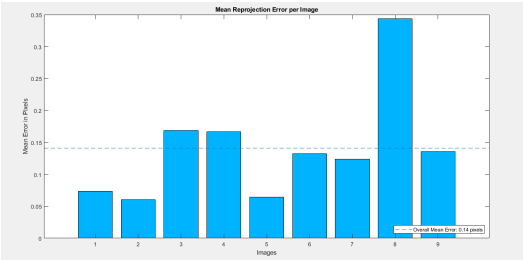


Figure 3.8: Reprojection errors

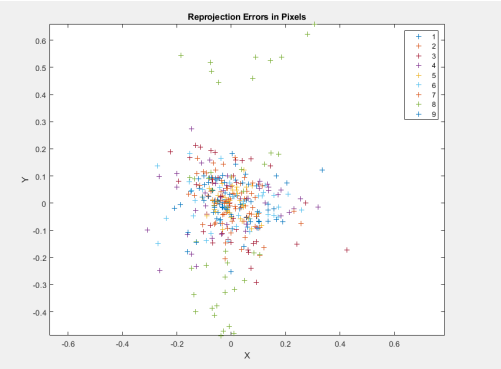


Figure 3.9: Scatter plot reprojection errors

The intrinsic parameters are reported in Table 3.2, while the extrinsic ones are visualized in Fig.3.10.

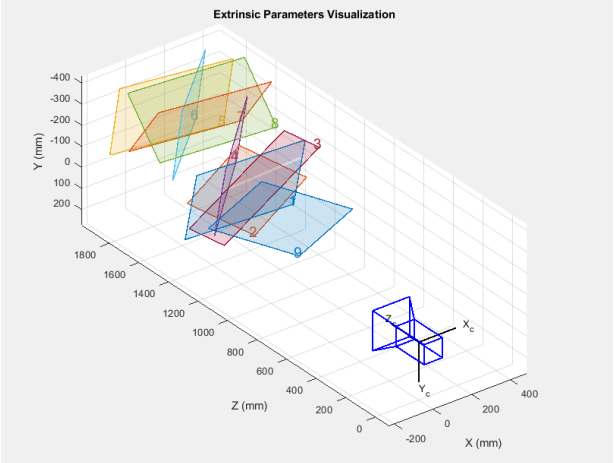


Figure 3.10: Camera-centered visualization of extrinsic parameters

Focal length	1178.5
	1179.7
Principal point	628.65
	532.69
Image size	1024
	1280

Table 3.2: Intrinsic parameters of NIR camera

3.5 Thermal camera calibration

In the calibration of the thermal camera there is an additional challenge. This is due to the fact that the edges on the checkerboard are detected only by the color cameras (since the difference between the squares is, in fact, related to the gray-level or color intensity). It is therefore necessary to find a way to make the FLIR-camera able to "see" the edges on the checkerboard.

One way to reach this goal is to exploit the different emissivity of the colors white and black on the checkerboard. The board was exposed to the direct sunlight of the lunchtime of the city of Coimbra during spring. An hour was more than enough to reach a proper heating of the panel. In this way, the black squares of the checkerboard absorbed more energy than the white ones, resulting in a different infra-red light emission that can be sensed by the FLIR.

Of course, the expected image of the checkerboard will be less clear and the edges between the squares less defined and sharp. This will be translated in a poorer calibration that needs to be set in a more precise way than in the other two calibrations analysed.

To do this, a calibration app [15] running on MATLAB was brought into play. This allows the user to manually select the corners of the checkerboard and to manually set parameters like the width and height of the window size of the corner

finder in the toolbox algorithm.

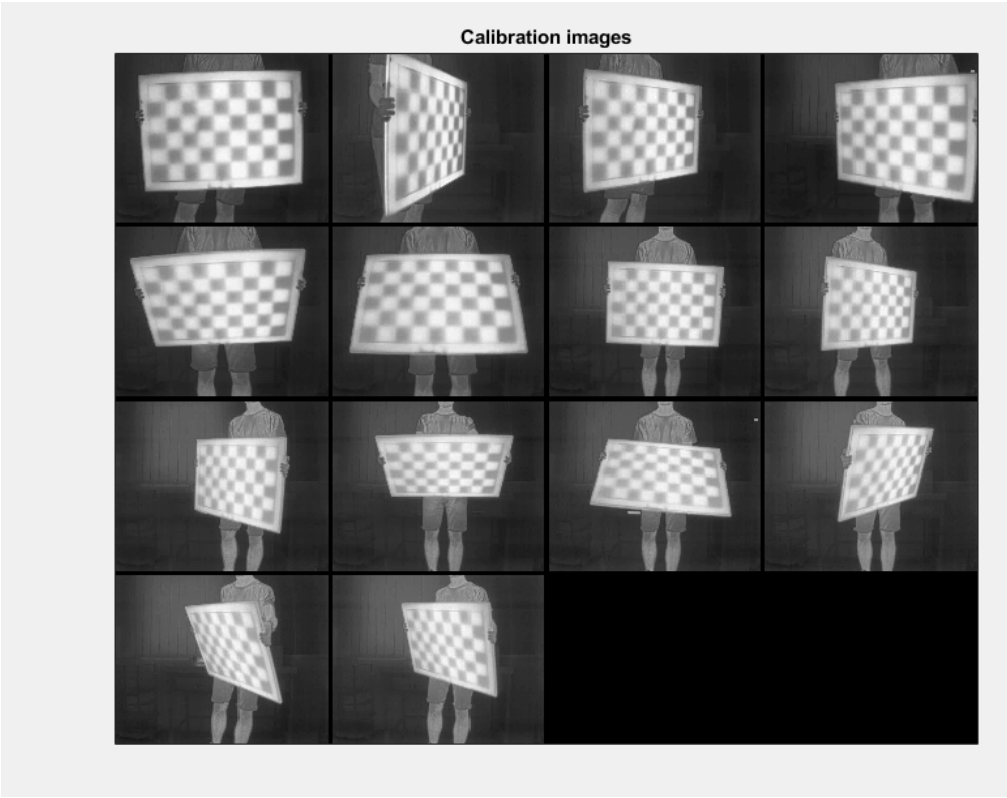


Figure 3.11: Dataset of 14 frames extracted for the thermal camera calibration.

This time, it was used an initial set of 14 images that can be visualized in the Fig.3.11. In the Figure 3.12, the corner points are projected by the algorithm, based on the number of squares of the checkerboard and the position of the four extreme corners, which was manually selected.

After a first calibration, the mean error returned by the algorithm results to be equal to [1.51, 1.56] (using this method the error is decomposed in error along the x and y direction), which cannot be considered acceptable ($norm([1.51, 1.56]) = 2.16$).

In order to reduce this error the calibration is more finely set and the following steps have been taken:

- the window size of the corner finder are set to $[wintx, winty] = [3, 3]$ from an initial value of [5, 5];
- the images with the major reprojection errors are suppressed resulting in a set

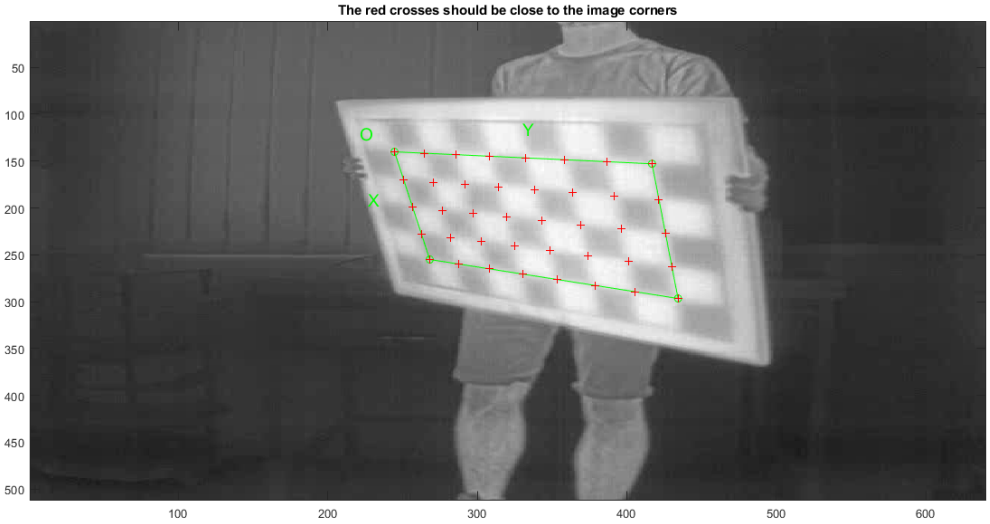


Figure 3.12: Corner points

of 10 used images as following: $img_{active} = [1, 2, 4, 5, 6, 9, 10, 11, 12, 14]$.

The calibration is still not very accurate as in the previous cases, like is shown in the extracted points in the Fig.3.13, but the reprojection error is lowered down to [0.72, 0.76], which has a norm value of $norm(e) = 1.04$ that can be considered acceptable. This can be noticed also in Fig. 3.14, showing that also in a different image the reprojected points are in average not precise as in the calibration of the other cameras. In the Table 3.3 the difference in mean reprojection errors among the three cameras can be evaluated.

RGB	0.07
NIR	0.14
FLIR	1.04

Table 3.3: Summary of the mean reprojection errors (pixels)

For a supplementary investigation the reader can also take the scatter plot shown in Fig. 3.15 as a reference. About the extrinsic parameters, it is convenient to visualize them as camera-centered as usual. They are pointed out in Fig. 3.16. Finally, the intrinsic parameters are reported in the Table 3.4.

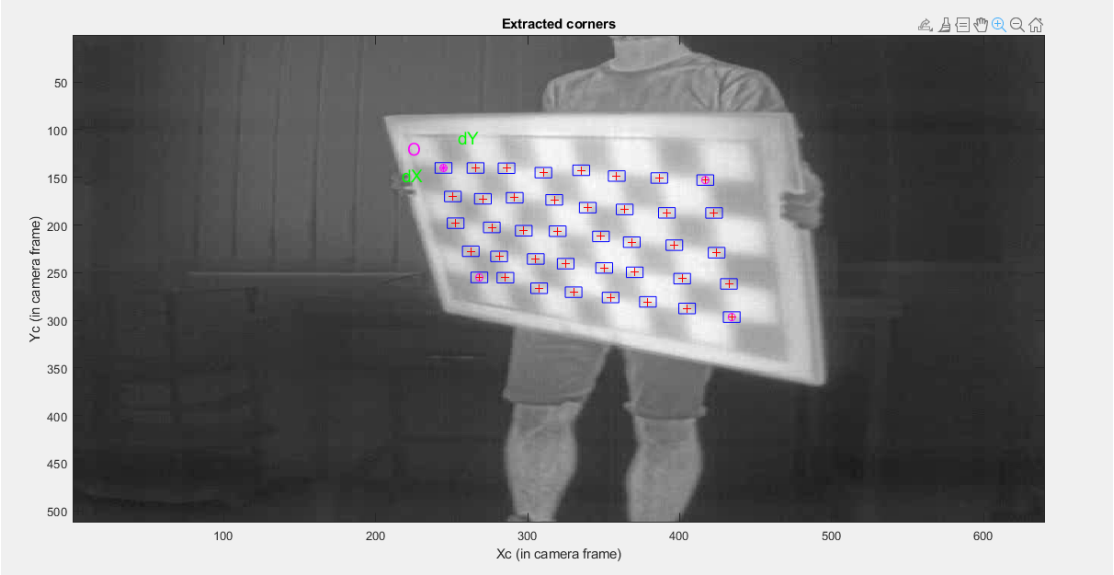


Figure 3.13: Extracted points

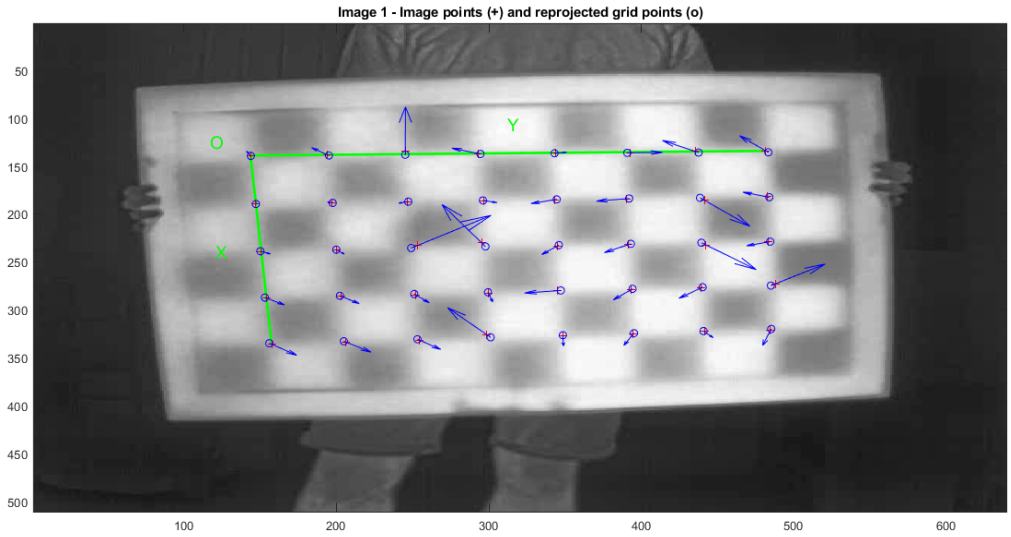


Figure 3.14: Reprojected points in image 001.jpg

3.6 LiDAR-RGB camera calibration

About the Velodyne®LIDAR VLP-16 calibration, the process is different with respect to the other camera sensors. The internal parameters are already calibrated

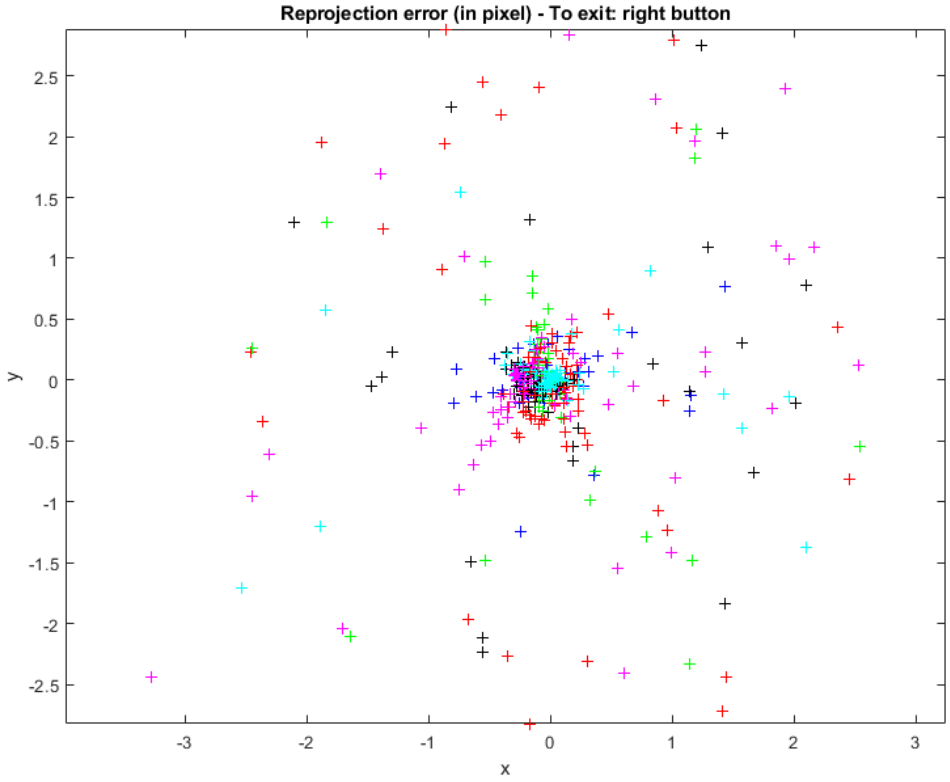


Figure 3.15: Scatter plot for reprojection error in thermal camera calibration

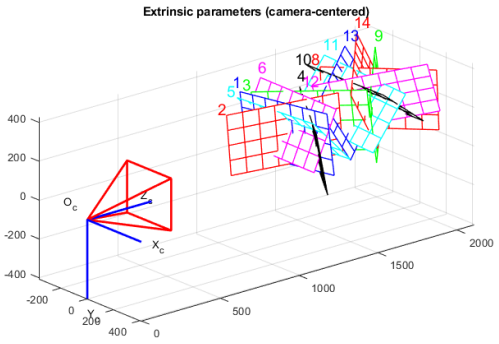


Figure 3.16: Camera centered visualization of thermal camera external parameters

Focal length	749.4
	762.84
Principal point	384.84
	282.69
Image size	512
	640

Table 3.4: Intrinsic parameters of FLIR thermal camera

by the manufacturer in advance. The fundamental part is then to link the 3-D Lidar points and the 2-D camera images. In summary, the aim of this part of the calibration section is to obtain a rigid transformation matrix T which maps from

the reference coordinate system of one of the cameras (i.e. RGB camera) to the coordinate system of the Lidar sensor. In order to do this MATLAB provides an useful function included in the Lidar toolbox:

```
[tform,errors] = estimateLidarCameraTransform(ptCloudPlanes,
    imageCorners);
```

where the input parameters are:

1. **ptCloudPlanes** | *P-by-1 array of pointCloud objects* | each pointCloud object must contain points that represent a checkerboard;
2. **imageCorners** | *4-by-3-by-P array* | each row of a channel is in the form [x y z] of a checkerboard corner extracted from the corresponding camera image;

and the outputs are a rigid 3D map and error data that will be later analyzed.

3.6.1 Extraction of checkerboard from point Clouds

The first step is to extrapolate relevant frames starting from a whole VeloView® recording (VeloView® is the software used to visualize and record the output of the lidar scan). To do this, a little script was developed and can be found in the appendix.

To detect the checkerboard starting from a point cloud will be exploited the following function implemented in MATLAB:

```
ptCloudPlanes = detectRectangularPlanePoints(ptCloudIn,
    planeDimensions,...)
```

In particular, it is used in the following script:

```
calibfolder = 'C:\Users\feder\Desktop\Dissertation\01-Calibration\
    calib rgb-lidar';
pcdfolder = fullfile(calibfolder, '4\pcd');
ptCloud = pcread(fullfile(pcdfolder,'11.pcd'));

pcshow(ptCloud)
title('Input Point Cloud')
```

```
xlim([-5 10])
ylim([-5 10])

sqsize=81.4;%mm
padding=[53 57 60 53];%mm
boardSize = [6*sqsize+padding(2)+padding(4) 9*sqsize+padding(1)+
    padding(3)];
mindist=0.2;
tol=0.1;
ROI=[-2.3 2 -3.3 6 -1 1];

lidarCheckerboardPlane = detectRectangularPlanePoints(ptCloud,...
    boardSize,'RemoveGround',true,'DimensionTolerance',tol,...
    'ROI',ROI,'MinDistance',mindist);

%visualization part
hRect = figure;
panel = uipanel('Parent',hRect,'BackgroundColor',[0 0 0]);
ax = axes('Parent',panel,'Color',[0 0 0]);
pcshow(lidarCheckerboardPlane,'Parent',ax)
title('Rectangular Plane Points')
figure
pcshowpair(ptCloud,lidarCheckerboardPlane)
title('Detected Rectangular Plane')
xlim([-5 10])
ylim([-5 10])
```

The parameters have been set in this way:

- **boardsize**: differently from the other cameras calibration, here it is necessary to compute the dimensions of the whole checkerboard, including the padding if present; in the checkerboard used in this work (see Fig. 3.17) the measured padding dimensions are [53 57 60 53] mm;
- **mindist**: clustering threshold for two adjacent points, high in case of low resolution lidars; in this case was set to 0.2;
- **tol**: tolerance for uncertainty in rectangular plane dimensions; set to 0.1;
- **ROI**: region of interest; is used to save computational memory; the algotithm will look for the checkerboard only in the selected area expressed in the form $[x_{min} \quad x_{max} \quad y_{min} \quad y_{max} \quad z_{min} \quad z_{max}] = [-2.3 \quad 2 \quad -3.3 \quad 6 \quad -1 \quad 1]$

The final part of the script is used to visualize the rectangular plane as can be seen in the Fig. 3.18a and 3.18b. This procedure is then repeated for the whole

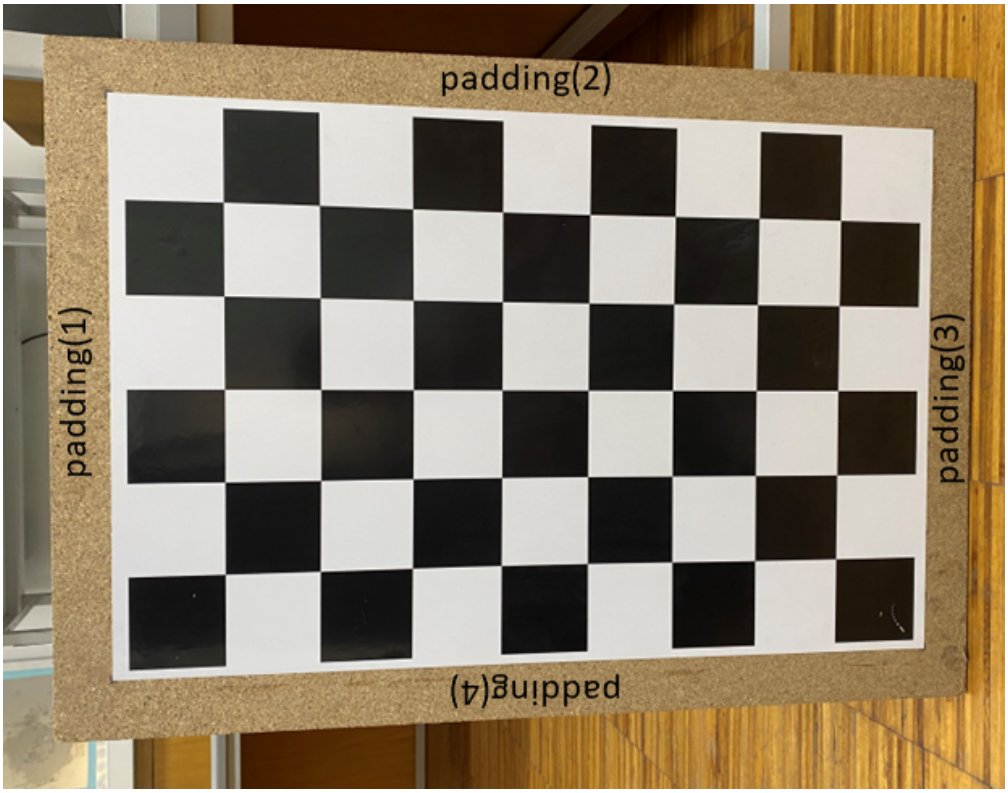
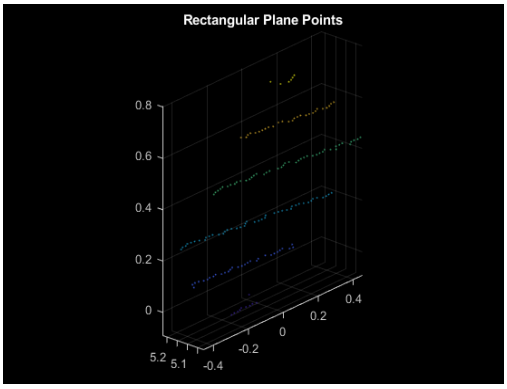
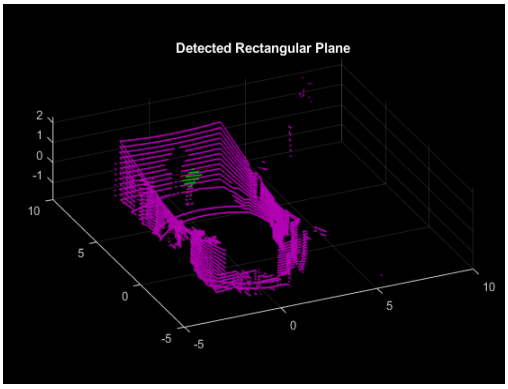


Figure 3.17: Checkerboard and padding vector parameters



(a) Rectangular plane points highlighted in the point cloud



(b) Detected rectangular plane

Figure 3.18: Checkerboard visualization for pointcloud "11.pcd" of the dataset

dataset of pointclouds in order to have an array of detected rectangular planes.

3.6.2 Extraction of checkerboard corners from the camera images

The next ingredient for the Lidar calibration is an array of the coordinates in the 3D world of the four corners of the checkerboard, for each of the images. To get this the following function will be exploited:

```
imageCorners3d = estimateCheckerboardCorners3d(I,cameraIntrinsic,
    checkerSize,...)
```

which needs as input parameters:

- **I**: image file
- **cameraIntrinsic**: intrinsic parameters (we know them from the prior RGB camera calibration
- **checkerSize**: size of the checker ($81.4mm$)
- **Padding**: this is a necessary 'Name-Value' argument, since the checkerboard of this work has a non-zero padding value([53 57 60 53], see Fig. 3.17)

and outputs as a matrix the coordinates of the four corners. The following script does all the work:

```
image = imread('C:\Users\feder\Desktop\Dissertation\01-Calibration\
    calib rgb-lidar\4\img\11.png');

%intrinsic evaluation
fc=[1180.2 1180.3];
cc=[633.0 511.6];
n=[1280 1024];
intrinsic = cameraIntrinsics(fc,cc,n);

%params evaluation
sqsize = 81.4;%mm

%corners extraction
boardCorners = estimateCheckerboardCorners3d(image,intrinsic,sqsize,'
    Padding',[53 57 60 53]);

```

```
%visualization
imPts = projectLidarPointsOnImage(boardCorners,intrinsic,rigid3d());
J = undistortImage(image,intrinsic);
imshow(J)
hold on
plot(imPts(:,1),imPts(:,2),'.r','MarkerSize',30)
title('Detected Checkerboard Corners')
hold off
```

and visualizes the corners on the image in the end as shown in the Fig.3.19. Similarly

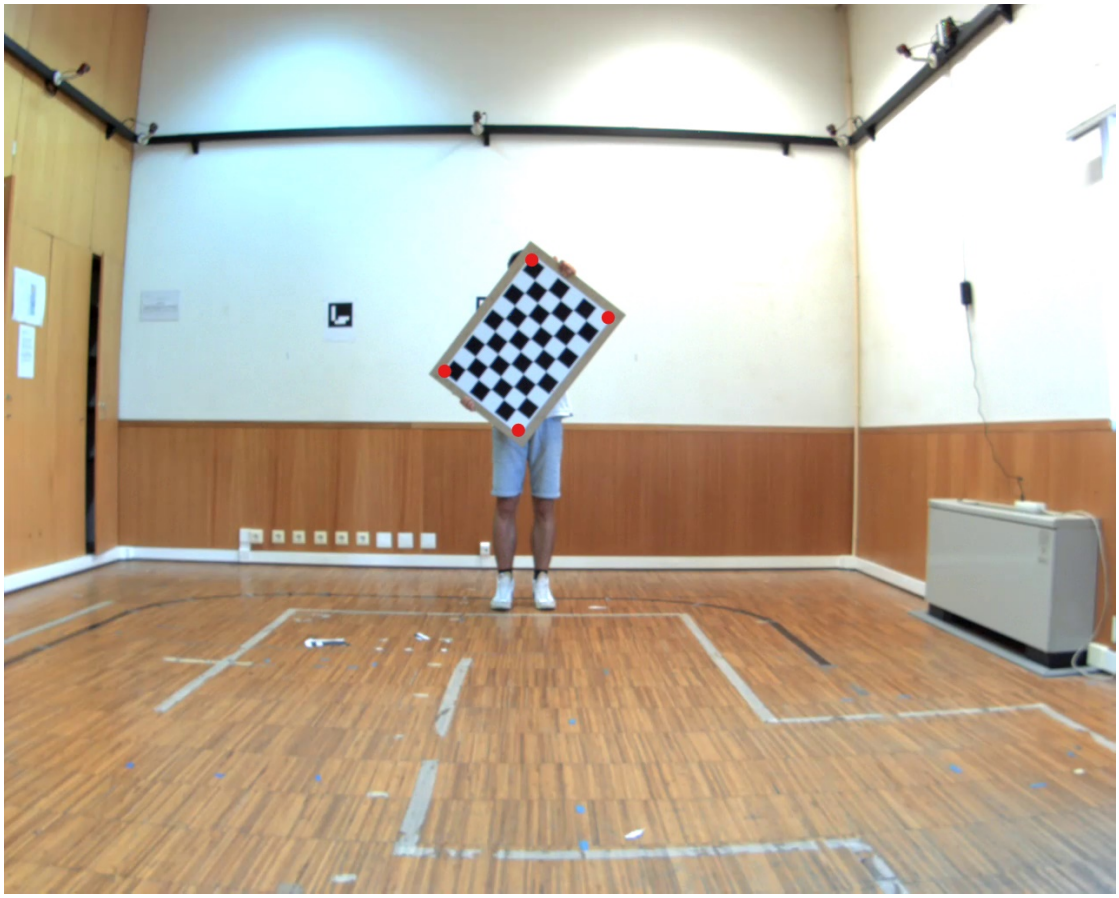


Figure 3.19: Corner extracted and visualized on the image

to the rectangular plane extraction, these matrices will be gathered in a single array with one channel for each image.

3.6.3 Estimation of rigid transformation lidar-camera and results

After the collection of the arrays **ptCloudPlanes** and **imageCorners** it is now possible to use the function *estimateLidarCameraTransform* which returns the following transformation matrix

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{t} & 1 \end{bmatrix} = \begin{bmatrix} 0.9999 & 0.0096 & 0.0089 & 0 \\ -0.0088 & -0.0110 & 0.9999 & 0 \\ 0.0097 & -0.9999 & -0.0110 & 0 \\ -0.0872 & -0.1186 & -0.0187 & 1 \end{bmatrix}$$

The other output of the function is a structure containing three different errors:

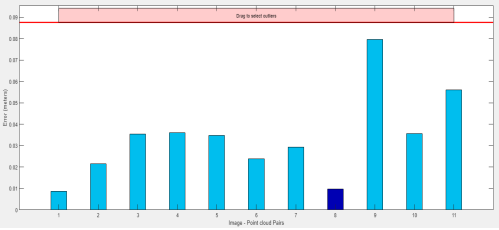
- **Translation error**: the difference between the centroid coordinates of checkerboard planes in the point clouds and those in the images
- **Rotation error**: the difference between the normal angles defined by the checkerboard planes in the point clouds (lidar frame) and those in the images (camera frame)
- **Reprojection error**: the difference between the projected (transformed) centroid coordinates of the checkerboard planes from the point clouds and those in the images

and all of these can be visualized in the Fig.3.20.

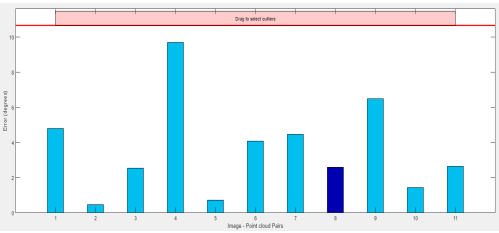
From this visualization it is easy to see and exclude the three image-point cloud couples which are characterized by the worst translational, rotational and reprojection error. This leads to a new dataset of 8 frames with the errors shown in Fig.3.21.

3.6.4 Lidar-RGB camera calibration validation

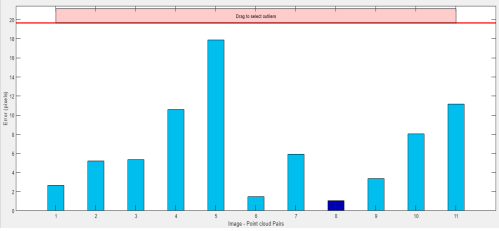
Before passing to the next step of the work it is important to validate the obtained results in the lidar-camera calibration, to check if the computed transformation matrix works properly. To do this, all the theoretical knowledge depicted in Chap. 3.2 will be applied to the images. The point cloud-image couples used in Chap. 3.6



(a) Translation error

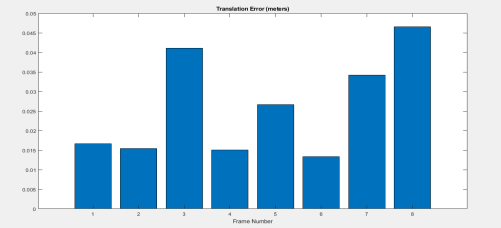


(b) Rotation error

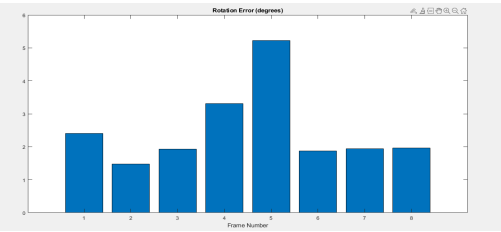


(c) Reprojection error

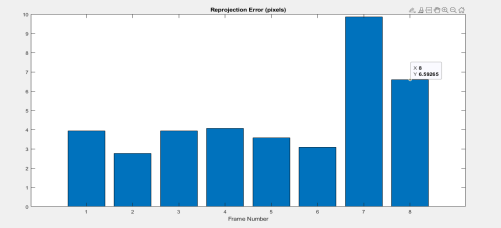
Figure 3.20: First dataset error visualization



(a) Translation error



(b) Rotation error



(c) Reprojection error

Figure 3.21: Final dataset error visualization

will be taken as a reference. Firstly, a reference point was chosen for each frame. The coordinates of the selected points in the Lidar world frame were extracted and saved in a .txt file that is reported in the Table 3.5, and can be used by the reader to test the following scripts. These are what have been previously called $\mathbf{x_P}$ coordinates.

The remaining job is to transform these points in image points exploiting the transformation matrix T from the Chap. 3.6 and the RGB camera intrinsic parameters from the Chap. 3.3. There are some differences with the formulae presented in Chap. 3.2 due to the fact that the calibration algorithms used in this work output the transpose matrices of the theoretical ones. So, in order to use these, the points coordinates will be computed as row vectors instead of column

Figure 3.22: Image point estimated coordinates are marked by a red circle (top-right corner)

	x_P	y_P	z_P
1	0.397	3.943	0.347
2	0.374	3.891	0.619
3	0.122	3.958	0.770
4	0.065	4.365	0.382
5	0.397	3.618	0.318
6	0.405	2.333	0.547
7	0.404	2.416	0.566
8	0.378	2.536	-0.045
9	0.357	5.036	0.442
10	0.280	5.091	0.267
11	0.466	5.013	0.440
12	0.058	5.208	0.456
13	0.569	4.181	0.369
14	0.642	1.967	0.478
15	0.101	2.328	0.453

Table 3.5: Reference points coordinates (one point for each frame)

and the multiplication with the matrices will be a right-hand side multiplication. The code used in MATLAB is the following:

```
calibfolder=('C:\Users\feder\Desktop\Dissertation\01-Calibration\
    calib_rgb-lidar\4\');
txtfolder=(fullfile(calibfolder,'txt\'));
imgfolder=(fullfile(calibfolder,'img\'));

Points=readmatrix([txtfolder,'points.txt']);
nframes=length(Points(:,1));

load(fullfile(calibfolder,'results.mat'));
load(fullfile(imgfolder,'calib_data.mat'),'params');

M_int=params.IntrinsicMatrix;
T=tform.T;

for i=1:1:nframes
```

```
15      x_point=[Points(i,:) 1]; %homogeneous coordinate
16      x_cam=x_point*T; x_cam=x_cam(1:3);
17      x_image=x_cam*M_int; x_image=x_image/x_image(3); %division by the
        scale factor wtilde
18      x_image=x_image(1:2);
19
20      path=sprintf('%s%02d.png',imgfolder,i);
21      figure(i)
22      imshow(path)
23      hold on
24      plot(x_image(1),x_image(2),'or');
25      axis on
26  end
```

In the end of the code there is a part that visualizes the original images and the corresponding projected reference point with coordinates $\mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix}$.

The results can be visualized in two examples reported in the Fig. 3.22. In both of them the reference point is the top-right corner, and it is possible to affirm that the estimation is accurate enough to proceed to the next part of the work.

3.7 FLIR-RGB camera calibration

It can be useful to understand the relative position of the two sensors used in this work, so this section will be focused on a quickly exploration of a method to find the orientation and the translation vector of the RGB camera with respect to the FLIR camera and vice-versa. The starting point is the general frame transformation formula that is the following:

$$\mathbf{x_c} = \mathrm{R_c}\mathbf{x_w} + \mathrm{t_c} \tag{3.5}$$

which converts the coordinates of a point in the world reference system $\mathbf{x_w}$ into the coordinates of the same point with reference to a different reference system (in our case of a generic camera) $\mathbf{x_c}$ using the rotation matrix R_c and the translation vector t_c. From (3.5) it is possible to do the following computations for our specific

cameras reference systems:

$$\begin{cases} \mathbf{x_{th}} = \mathrm{R}_{\mathrm{c_{th}}}\mathbf{x_w} + t_{c_{th}} \Longrightarrow \mathrm{R}_{\mathrm{c_{th}}}^{-1}(\mathbf{x_{c_{th}}} - t_{c_{th}}) = \mathbf{x_w} \\ \mathbf{x_{rgb}} = \mathrm{R}_{\mathrm{c_{rgb}}}\mathbf{x_w} + t_{c_{rgb}} \end{cases} \tag{3.6}$$

with $\mathbf{x_{th}}, \mathbf{x_{rgb}}$ the coordinates in the cameras' frames, $\mathrm{R}_{\mathrm{c_{th}}}, \mathrm{R}_{\mathrm{c_{rgb}}}$ the rotation matrices (considered here always invertible), $t_{c_{th}}, t_{c_{rgb}}$ the translation vectors. Remember that R_{c_i} and t_{c_i} are known from the camera calibration and are the external parameters of the two cameras. Now it is trivial to express the coordinates of a camera with respect with the ones of the other:

$$\mathbf{x_{rgb}} = \mathrm{R}_{\mathrm{c_{rgb}}}\mathrm{R}_{\mathrm{c_{th}}}^{-1}(\mathbf{x_{th}} - t_{c_{th}}) + t_{c_{rgb}} = \mathrm{R}_{\mathrm{c_{rgb}}}\mathrm{R}_{\mathrm{c_{th}}}^{-1}\mathbf{x_{th}} - \mathrm{R}_{\mathrm{c_{rgb}}}\mathrm{R}_{\mathrm{c_{th}}}^{-1}t_{c_{th}} + t_{c_{rgb}} \tag{3.7}$$

and, with the notation

$$\mathrm{R}_{\mathrm{th2rgb}} = \mathrm{R}_{\mathrm{c_{rgb}}}\mathrm{R}_{\mathrm{c_{th}}}^{-1}$$

and

$$t_{th2rgb} = -\mathrm{R}_{\mathrm{c_{rgb}}}\mathrm{R}_{\mathrm{c_{th}}}^{-1}t_{c_{th}} + t_{c_{rgb}}$$

equation (3.8) is obtained, which results to be a direct connection between the two reference systems of the two cameras.

$$\mathbf{x_{rgb}} = \mathrm{R}_{\mathrm{th2rgb}}\mathbf{x_{th}} + t_{th2rgb} \tag{3.8}$$

3.7.1 FLIR-RGB camera calibration validation

In order to validate the results obtained in the previous subsection and the ones obtained in the FLIR calibration, a procedure similar to the one seen in the subsection 3.6.4 will be performed. In particular, will be added a further step to that algorithm:

1. transform the world coordinates of a point (thanks to the lidar scan) to the coordinates of the point with respect to the RGB-camera thanks to the external parameters of the camera;

2. transform the RGB-camera point coordinates to the 2D image coordinates

thanks to the internal parameters of the camera and project them on the image to see if the point is at the corner of the checkerboard;

3. transform the RGB-camera point coordinates to the FLIR-camera point coordinates thanks to the equation (3.8) and its inverse;

4. transform the FLIR-camera point coordinates to the 2D image coordinates thanks to the internal parameters of the camera and project them on the image to see if the point is at the corner of the checkerboard.

Notice that only step 3) and 4) are new with respect to the RGB camera calibration validation.

```
clear
close all
clc

%folders configuration
calibfolder=('C:\Users\feder\Desktop\Dissertation\03-Calibration\
    calib-final');
txtfolder=(fullfile(calibfolder,'txt\'));
thimgfolder=(fullfile(calibfolder,'thermal\png\'));
rgbimgfolder=(fullfile(calibfolder,'rgb\png\'))

%reading of the relevant points from a text file
Points=readmatrix([txtfolder,'points.txt']);
nframes=length(Points(:,1));

%load internal and external parameters of the two cameras
load(fullfile(calibfolder,"rgbparams.mat"));
tform_rgb=tform; R_rgb=tform.R(:,:,1); t_rgb=tform.t(:,1);
M_int_rgb=params.InternalMatrix;
load(fullfile(calibfolder,"flirparams.mat"));
tform_th=tform; R_th=tform.R(:,:,1); t_th=tform.t(:,1);
M_int_th=params.InternalMatrix;

%compute the rotation and translation parameters
R_th2rgb=R_rgb*inv(R_th);
t_th2rgb=-R_rgb*inv(R_th)*t_th+t_rgb;
th2rgb=rigid3d(R_th2rgb,t_th2rgb);
rgb2th=inv(th2rgb);

for i=1:1:nframes
    x_point=[Points(i,:) 1]; %homogeneous coordinate
    x_rgb=x_point*tform_rgb.T;
```

```
32      x_rgb=x_rgb(1:3);
33      x_th=x_rgb*rgb2th.T;
34      x_th=x_th(1:3);
35      x_im_th=x_th*M_int_th; x_im_th=x_im_th/x_im_th(3);%division by
        the scale factor wtilde
36      x_im_th=x_im_th(1:2);
37      x_im_rgb=x_rgb*M_int_rgb; x_im_rgb=x_im_rgb/x_im_rgb(3);
38      x_im_rgb=x_im_rgb(1:2);
39
40      path_rgb=sprintf('%s%03d.png',rgbimgfolder,i);
41      path_th=sprintf('%s%03d.png',thimgfolder,i);
42      figure(i)
43      subplot(2,1,1)
44      imshow(path_th)
45      hold on
46      plot(x_im_th(1),x_im_th(2),'or');
47      axis on
48      subplot(2,1,2)
49      imshow(path_rgb)
50      hold on
51      plot(x_im_rgb(1),x_im_rgb(2),'or');
52      axis on
53  end
```

This returns the images shown in Fig. 3.23.

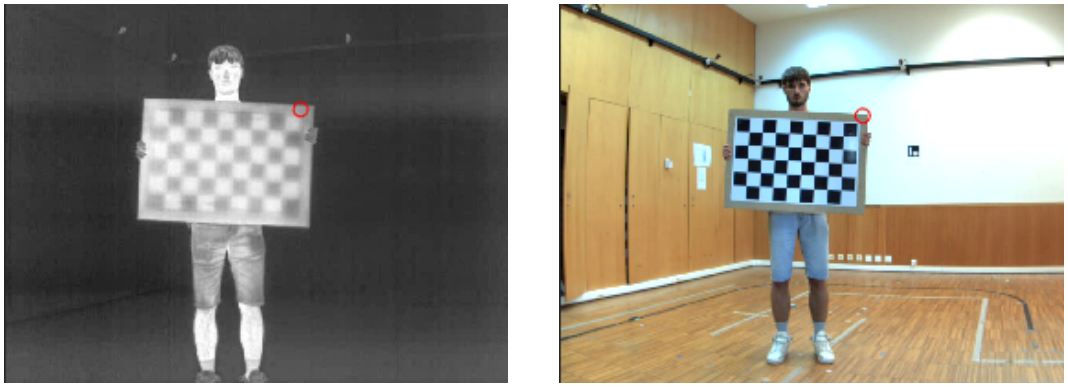


Figure 3.23: The correspondence between the red dot in the left image and in the right one shows how the thermal camera parameters and the relative orientation with respect to the RGB one are valid and usable.

Chapter 4

Dataset, experiments and results

4.1 Machine Learning

> The field of study that gives computers the ability to learn without being explicitly programmed.
>
> Arthur Samuel, IBM

Over the past decades, interest around Machine Learning (ML) related topics has risen in a considerable way. It is discussed in a wide variety of fields, from computer science classes to business conferences. Machine learning is essentially the use of algorithms to extract information from unstructured data and represent it in a model. This model is used to draw conclusions about additional data that we have not yet modeled.

To give a high-level insight of a ML formulation, in the domain of linear algebra, the main interest is in solving linear equations in the form:

$$Ax = b$$

where A is a matrix of our set of input row vectors, b is the column vector of labels for each vector in the A matrix and x is called *parameter/feature vector*. In a very

simplistic manner a machine learning algorithm has his main goal in minimizing the error of this equation through **optimization**.

In optimization, the objective is to iteratively change the values in the x column vector until a good set of values is discovered, that produces results that are as accurate as possible. After, the loss function determines the error (depending on the actual outcome, as illustrated previously as the b column vector), each weight in the weight matrix will be modified. This is called *parameter optimization*. It can be seen like a scientific method:

1. *formulate a hypothesis*: choose a set of weights;
2. *test against reality*: use the weights in the problem;
3. *refine hypothesis and repeat*: iterate the process to minimize the error.

4.1.1 How to evaluate a model

In order to understand how well a model gives the correct classification and to measure the value of a prediction, generally the following parameters (*i.e.* performance *metrics*) are employed.

Confusion matrix

The *confusion matrix* is a table where the predictions and the actual outcomes for a classifier are compared.

	P' (Predicted)	**N' (Predicted)**
P (Actual)	True Positive	False Negative
N (Actual)	False Positive	True Negative

Table 4.1: Definition of confusion matrix.

We measure these values in the following way:

- True positives [**TP**]: positive prediction, label was positive
- False positives (*a.k.a. type I errors*) [**FP**]: positive prediction, label was negative

- True negatives [**TN**]: negative prediction, label was negative

- False negatives (*a.k.a. type II errors*) [**FN**]: negative prediction, label was positive

Sensitivity or Recall

Also called *true positive rate*, this parameter quantifies how well the model avoids false negatives:

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

Specificity

In the contrary, this parameter quantifies how well the model avoids false positives:

$$Specificity = \frac{TN}{TN + FP}$$

Accuracy

In the most general definition, accuracy is the degree of closeness of measurements of a quantity to that quantity's real value:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision

Also called *positive prediction value*, it differs from the accuracy parameters since it only compares the positive predictions:

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

It is possible to have a model which is accurate but not precise, and vice-versa. We consider a model good if it is both.

F1

The F1 score is the harmonic mean of both the precision and sensitivity measures into a single score:

$$F1 = \frac{2TP}{2TP + FP + FN}$$

In binary classifications it is common to use this parameter to evaluate the model's accuracy. Its value goes from 0 to 1 and we consider acceptable those values which are close to the 1.

4.1.2 Deep Learning

> A type of machine learning based on artificial neural networks in which multiple layers of processing are used to extract progressively higher level features from data.
>
> Definition from Oxford Languages

Deep Learning has evolved gradually through the years, which made it difficult to define properly. In the present state of art, it is possible to state that a DL algorithm is a neural network with a large number of parameters and layers in one of the following four fundamentals architectures:

- unsupervised pretrained networks
- convolutional neural networks (CNNs)
- recurrent neural networks
- recursive neural networks

One of the many advantages deep learning provides over conventional machine learning algorithms is 'automatic' feature extraction. This means that the network can determine which properties of a dataset can be used as indicators to accurately label that data. That is the reason why deep learning is usually employed in the fields of: computer vision, natural language processing, sound recognition, face recognition, self-driving cars.

As a further proof of the deep learning algorithms' potentiality, it will be reported here an example of modern application regarding the Covid-19 pandemic and how the AI helped us in the fight against the virus.

Shorten et al. reported in the article [22] how deep learning algorithms have been employed in different applications during the recent years' pandemic:

- **Natural Language Processing**: it has been crucial to analyse written data like Covid-19 clinical reports, but also to understand and categorize public questions (*i.e.* coming from media and social networks) in order to figure out what the public is concerned about with respect to the pandemic;
- **Literature Mining**: the most popular open literature dataset *CORD-19* contains over 128,000 papers. From this datum it's easy to deduct that a simple human-driven search could result difficult and could be useful a little help coming from ML algorithms and neural networks. The tools developed are the following and all of them use deep learning techniques: CO-search, Covidex, SLEDGE, CAiRE-COVID;
- **Misinformation Detection**: the SARS-CoV-2 and COVID-19 infodemic's information dissemination has been chaotic. Numerous studies have developed categorization models to flag tweets that might include false information. One of the most effective algorithms [23] labels tweets according to 7 question labels (contains a verifiable factual claim? is likely to contain false information? is of interest to the general public? is potentially harmful to a person, a company, a product, or society? requires verification by a fact-checker? poses a specific kind of harm to society? requires the attention of a government entity?). The output labels are *misinformative*, *informative* or *irrelevant*.

For sake of completeness, here will be reported the other applications of DL during the pandemic that won't be discussed in this work. The reader can find a further exploration in the article [22] about:

- Medical Images Analysis;
- Ambient Intelligence;
- Vision-based Robotics;

- Precision Diagnostics;
- Protein Structure Prediction;
- Drug Repurposing;
- Epidemiology.

Convolutional Neural Networks (CNN)

To give an overview of the potentiality of the CNNs it is enough to start from the birth them. Every year, during the ImageNet challenge, teams of researchers compete in the attempt to classify images into one of the possible 200 classes given a training dataset of about 450,000 images. No surprise, this event is considered the "Olympics of Computer Vision" [24]. The goal of the competition is to encourage development in the field of Computer Vision algorithms to compete with the accuracy of human vision itself (about 95%).

In 2011, the winner of the ImageNet benchmark managed to achieve a result of 25.7% as error rate, making a mistake on one out of four images. It was quite far from humans' recognition capacity, but much better than random guessing. Then in the next year, Alex Krizhevsky from Geoffrey Hinton's lab at the University of Toronto has been a game changer: pioneering a deep learning architecture known as *convolutional neural network* easily won the competition, with just some months of work, with an error rate of approximately 16%. His network, called AlexNet, put Deep Learning on the map for computer vision and significantly changed the sector.

To describe properly a convolutional neural network the definition will be divided in two parts: **convolution** and **neural networks**. Let's start from the latter.

Neural Network is a computational model that takes inspiration from the animals' brain. In the great part of mammals' encephalon the vision (just like all of the mind's tasks) works thanks to specific cells called **neurons** (Fig. 4.1) which are connected with each other through a dense net. They exist to communicate with one another and pass electro-chemical impulses by means of *synapses*, as long as the impulse is strong enough to activate the release of ions across the synaptic slit. It's important to highlight here that the impulse must overpass a certain threshold

in order to stimulate the chemical release. Each of the connections between neurons are dynamically strengthened or weakened depending on how often they are used. It's the strength of a connection (*synapse*) which determines its weight in the input that will reach the nucleus of the neuron. After being filtered by the strength of the connections, the inputs are summed in the cell's body and then changed in the *axon* of the neuron to be delivered to another cell.

This process can be translated into computers' algorithms with an artificial model shown in Fig. 4.2. As for the biological neuron, our artificial one takes as inputs x_1, x_2, x_n and multiplies them by a specific weight w_1, w_2, w_n. The sum of the results is the so-called *logit* of the neuron: $z = \sum_{i=0}^{n} w_i x_i + b_i$, where b_i represents a bias term used to move the function from the origin. The logit is then passed through a function ϕ (non-linearity function) to become the output and to be transmitted to the next neuron $y = \phi(z)$.

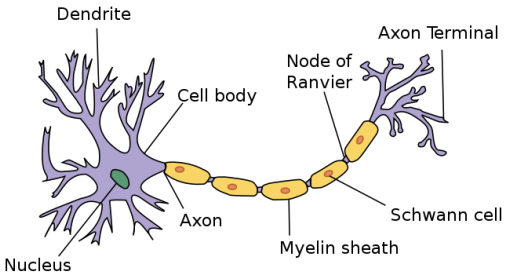


Figure 4.1: Outline of a mammal's neuron structure (image from Wikipedia [25])

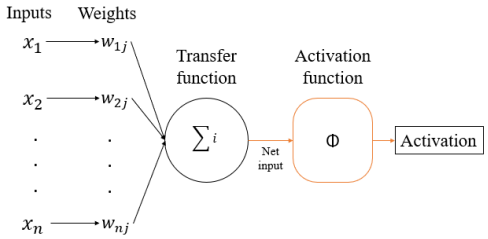


Figure 4.2: Artificial neuron working principle

As the reader can imagine, although they are very powerful, single nodes are not nearly expressive enough to solve complex problems, like for example differentiate handwritten digits. There is a reason in our brain there are millions of neurons: in animals' brain, they are organized in layers; if we take as a reference the human cerebral cortex, it is made up of 6 different layers [26], and sensory information flows from one layer to the other until becomes as an output a conceptual understanding. Taking again as inspiration the biology, the **neural networks** are built in the same way.

As depicted in Fig. 4.3 the usual structure of a neural network is divided in three kind of layers: input layer, hidden layers and output layer. The input layer

pulls the input data and passes them to the nodes that can be found in the first hidden layer. Usually a network has many hidden layers and each layer has a different number of nodes. Each node of a layer is hooked up with the nodes of the following layer and this connection is weighted by different coefficients for each layer and for each node. For sake of example, the coefficient $w_{i,j}^{(k)}$ connects the *j-th* neuron of the *k-th* layer to the *i-th* neuron of the previous layer (see Fig. 4.4).

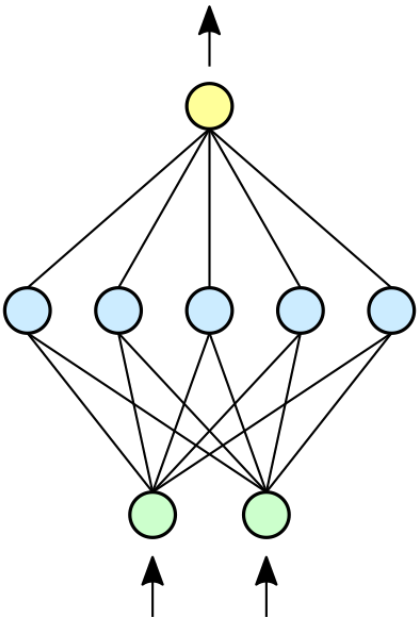


Figure 4.3: Outline of a very simple neural network: an input layer formed by 2 nodes (green), a single hidden layer formed by five nodes (light blue) and an output layer formed by a single node (yellow). Image from Wikipedia [27]

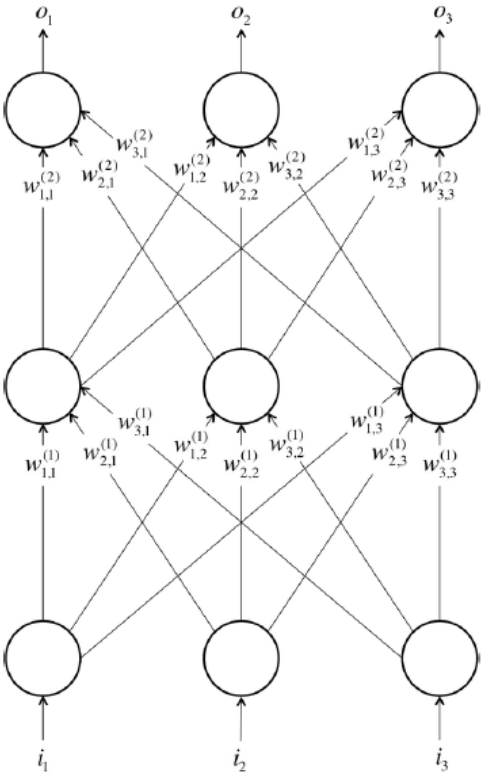


Figure 4.4: Another example in which are highlighted the weights coefficients $w_{i,j}^{(k)}$

Most neurons are defined by the ϕ function they apply to the *logit* $y = \phi(z)$. Usually this function is a non-linear one: this is because linear-functions' neurons run into very serious limitations because linearity reduces the complexity of the network. In fact, it can be proved that a network made of only linear nodes can be expressed as a network with no hidden layers. Which is inconvenient since all the

important features are learnt from the model in the hidden layers. For this reason, we need to introduce neurons that employ some nonlinearities in order to have a model which is able to learn complex features.

There are three common neurons that use nonlinearities and are usually applied in convolutional neural networks. The first one to be depicted is the **sigmoid**. Its function is $\phi(z) = \frac{1}{1+e^{-z}}$ and can be seen on the left of Fig.4.5. When the logit is very small, the output of the neuron node is close to 0, and when it is very large the output is close to 1. In between these extremes, the function assumes a "S shape". Since this function is not zero-centered, the **tanh** (hyperbolic tangent) is usually preferred to the previous one. The characterizing function is $\phi(z) = tanh(z)$ (in the middle of Fig. 4.5) and goes from -1 to 1. Its S shape is also sharper, so the output tends faster to the extreme values. Last and most used are the **ReLU**, which use a particular function (*Restricted Linear Unit*, it can be found on the right in Fig. 4.5) that can be expressed as follows:

$$\phi = \begin{cases} 0 & \text{if } z \leq 0 \\ z & \text{if } z > 0 \end{cases}$$

or, in a more compact way $\phi = max(0, z)$.

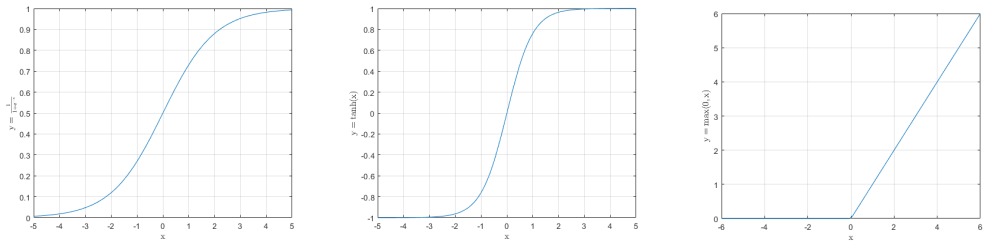


Figure 4.5: From left to right: sigmoid function, tanh function and ReLU function

Now, to go on with the description of a CNN, it is necessary to refer again to the structure of the human brain. This is because it has been used as an inspiration to build the artificial model. In particular, it has been discovered in the late first half of the 20-th century how parts of the visual cortex are used by our brain to detect edges. Later, scientists also found out that the cortex was divided in layers, and that each layer builds on the features detected in the previous layers; so the

information coming from the eyes is interpreted firstly as lines, then as contours, then as shapes and finally as objects.

Coming back to the computer science domain, in order to obtain a simple detail starting from a very complex image it is commonly used the technique of **convolution**. It consists of an iterated application of a filter (or *kernel*) to a whole matrix, to get a so-called *feature map*. An image can be interpreted as a matrix, or as a combination of matrices (depending on the quantity on channels of that image, for example RGB images have 3 channels). From this, it is trivial to deduct that using the **convolution**, an image can be decomposed into simpler and simpler objects in order to extract basic features. Moreover, in real application filters also have a depth, due to the fact that most of the images have more than one channel; this is why usually kernels are also called *volumetric convolutional filters*.

These filters have the following hyperparameters:

- **spatial extent** e, which is basically the filter's width and height. Usually the kernels have little dimensions, at most equal to 7×7;
- **stride** s, which is the distance between the points where the filter is applied. If $s = 1$ the kernel is applied to each $e \times e$ cell of the matrix.

For sake of example, a very basic application of convolution will be here analyzed. In Fig. 4.6 an input matrix of dimension 7×7 is depicted, to the which a 3×3 kernel is applied and a the convolution's stride is set to $s = 2$. The filter is so executed on the first 3×3 cell of the matrix, the upper-right one. An element-wise matrix multiplication gives as a result the first element of the feature map. The next step will be to multiply the same filter with the following 3×3 cell in the matrix, remembering to skip one of the cells because of the stride equal to 2. In this case the *feature map* will be a matrix of dimension 3×3 because of the stride. If the stride was 1, the dimension would have been 5×5.

In a more realistic application, the dimensions would have been very larger: if we take as an example a RGB-image as an input, it is made up of 3 different channels. Let's assume the image dimensions to be 416×416, the input matrix would have been $416 \times 416 \times 3$. The kernel is necessary of the same depth of the input, so it must have 3 channels as the RGB image. The filters dimensions could be, for example $5 \times 5 \times 3$. Depending on the application, the stride could be 8, 16

or even 32; in object recognition, the greater is the stride, the bigger is the object to be detected.

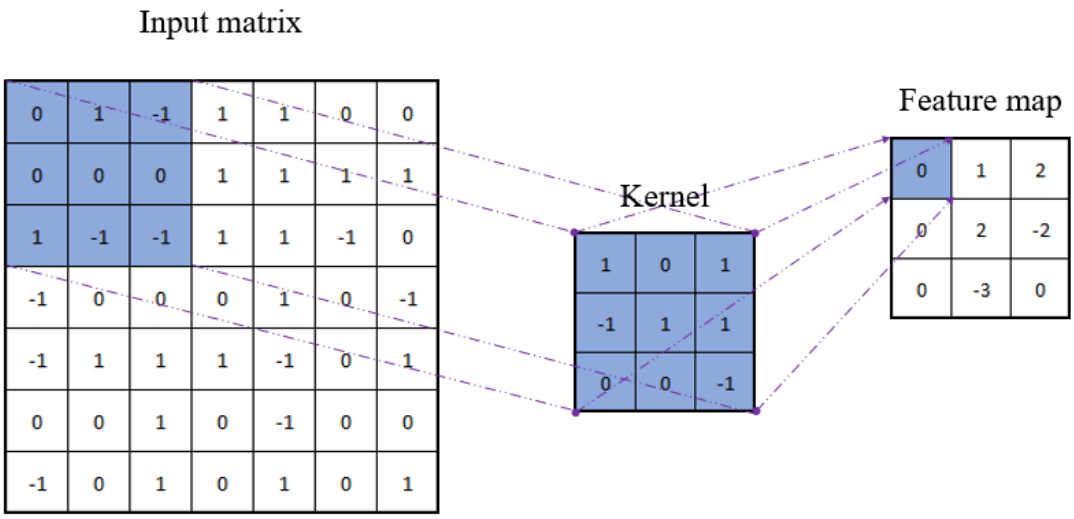


Figure 4.6: An example of simple convolution computation with hyperparameters $s = 2$, $e = 3$.

A particular type of node could be involved, inside of a CNN, into a slightly different kind of operation. Usually, after a convolutional layer can be found a *max pooling layer*. It can be characterized by the following hyperparameters:

- **spatial extent** e, which is the dimension of the cells collected;
- **stride** s, which is the distance between the points where the pooling operation is applied.

Again, it's reported in Figure 4.7 a very basic example of pooling operation, only for the reader's comprehension. Here it is applied a pooling layer with stride equal to 2 and spatial extent equal to 2. Thus the input matrix, originally of dimension 4×4 is transformed in another matrix which will be *locally invariant* to the previous, of dimension 2×2. The reason of this operation can be found in the will to create a *condensed* feature map. *De facto*, this operation does not modify the feature map, since even if it locally slightly differs, the result will still be the same. It's important to highlight that the major applications of the pooling layers are

always with parameters $(s, e) = (2,2)$ or $(2,3)$ respectively called *non-overlapping pooling layer* and *overlapping pooling layer*.

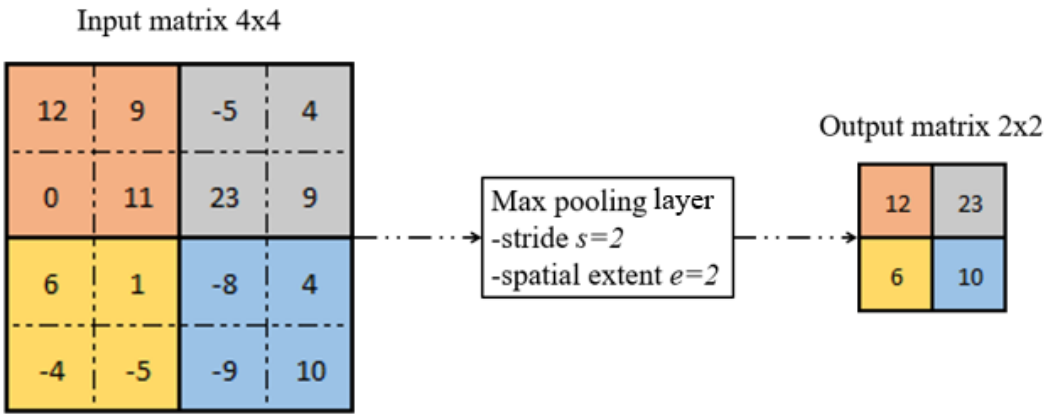


Figure 4.7: A simple example of non-overlapping pooling operation with parameters $s = 2$, $e = 2$.

Now that the whole picture has been outlined, it's interesting to take a look to the structure of a real implemented architecture, in particular one built for ImageNet, called VGGNet (Fig. 4.8).

Notice how the last section of the model's architecture is usually provided with fully connected (FC) layers, which are used to create relations between the features allowing to classify the input data.

YOLO (You Only Look Once)

Usually, object detection goals are reached by algorithms which break the problem in two stages: *detect* possible regions where the object can be, *classify* the image in the detected regions. Two among the most popular networks of this type are Fast-RCNN and Single-Shot MultiBox Detector (SSD), which both propose an area inside the image where the object could be found.

Differently from those, YOLO algorithm makes prediction on bounding boxes and probabilities all at once. This structural comparison is depicted in the Fig. 4.9. YOLO can achieve most of the state-of-art results with a completely different approach, that brings this model to the top of the most efficient and fastest real-time

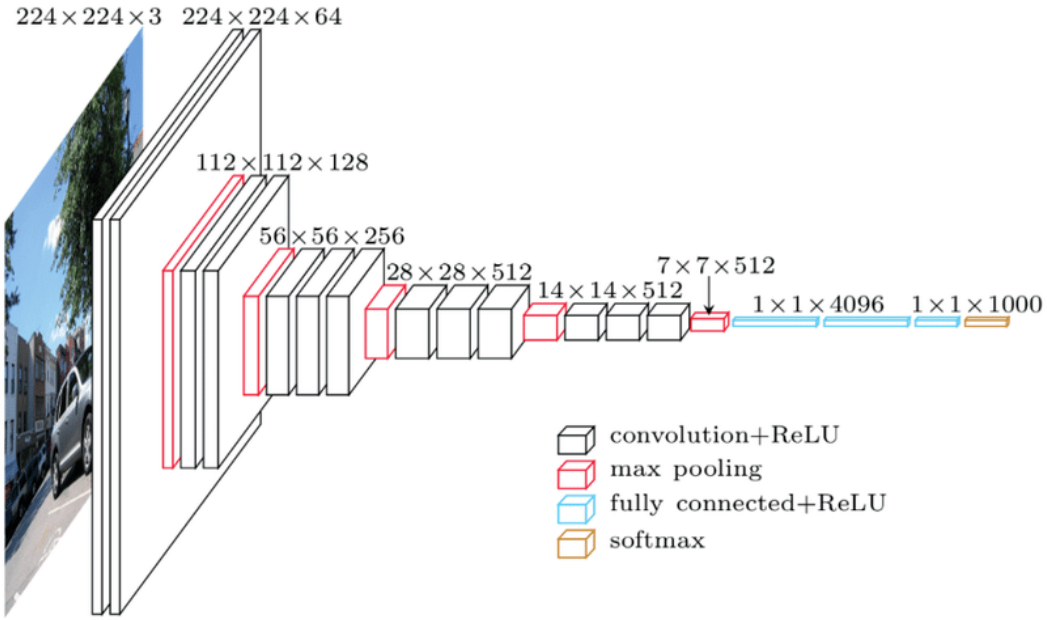


Figure 4.8: VGGNet architecture (source: [28])

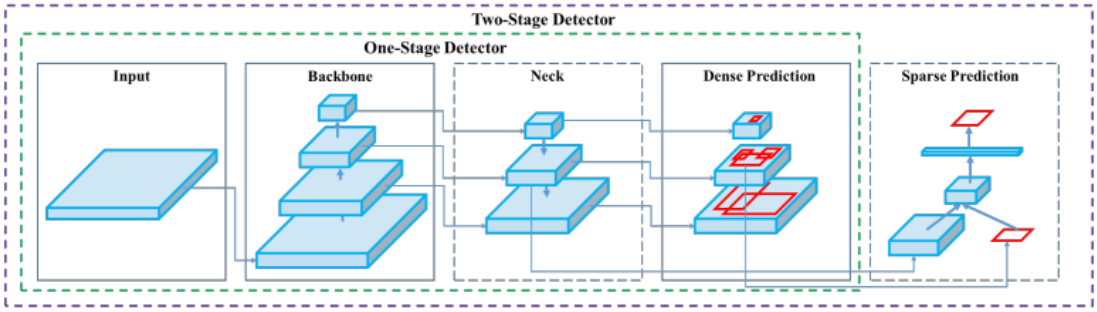


Figure 4.9: One-stage and two-stages detectors' architectures [29]

deep learning models (as shown by the graph in Fig. 4.10). The region-focused approach algorithm works in a very simple way: the picture is first separated into many grids, where $S \times S$ are the dimensions of each grid. The grids created from an input image are displayed in an example in the Fig. 4.11. Every grid cell will be able to detect items that enter it. For instance, a grid cell will be in charge of detecting an item if its center appears within that cell.

Three anchor boxes are predicted for each cell grid, along with confidence scores for each box. **Logistic regression** is used to compute **confidence ratings**, which

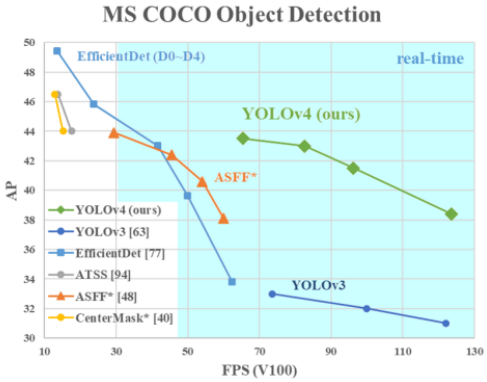


Figure 4.10: Comparison of the proposed YOLOv4 and other state-of-the-art object detectors. AP stands for Average Precision, FPS is the speed (frames per second) [29]

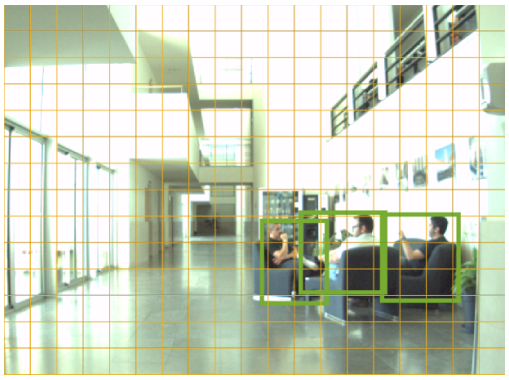


Figure 4.11: YOLO's grid cell example over a labelled image. Frame taken from output4 of the dataset

express how certain the model is that the box contains an item as well as how accurately it believes the box to be predicted. A brief parenthesis about logistic regression:

> Logistic regression is a simple and efficient method for binary and linear classification problems. It is a classification model, which is very easy to realize and achieves very good performance with linearly separable classes [30].

If the anchor box overlaps a ground truth item by more than any other anchor box before, allowing for object recognition, the confidence score should be 1. Finally, the **Non-Maximum Suppression (NMS)** approach is employed to get rid of numerous detections on the same item and selects the bounding box with best confidence score and best IoU parameter (see Fig. 4.12 for a visual definition of IoU).

YOLO performs a detection at 3 different layers:

- @layer 82 the grids have stripe 32, so the net is sparse and large objects are detected;
- @layer 94 the grids have stripe 16, so the medium objects are detected;

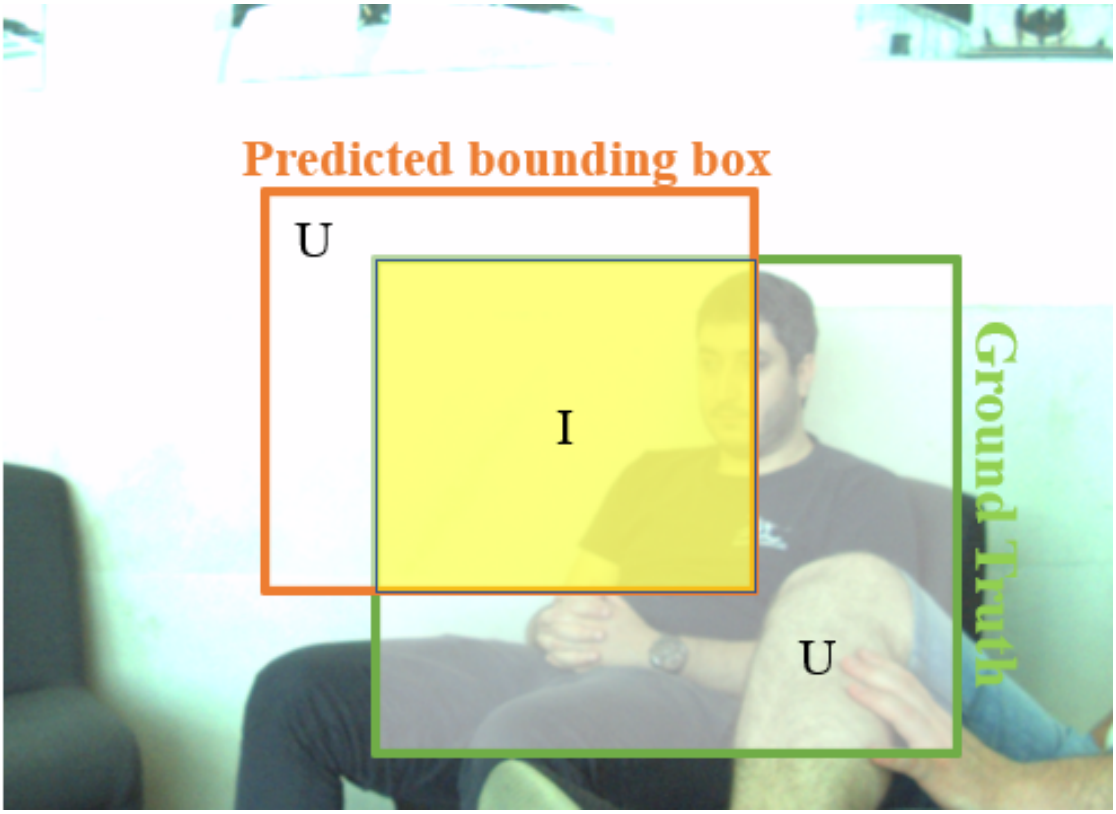


Figure 4.12: $IoU = \frac{Intersection\ Area}{Union\ Area}$

- @layer 106 the grids have stripe 8, so the net is very dense and small objects are detected

The Darknet53 (composed of 53 layers) is used twice for detection purposes, to totalize an amount of 106 layers. The following kind of layers are employed in YOLO and can be visualized in the Fig. 4.13:

- **convolutional layers**;
- **shortcut layers**, used to skip connections;
- **upsample layers**, usually before YOLO layers, upsample the previous feature map by a stride factor;
- **route layers**, copy N backward layers' feature maps;
- **YOLO layers**, detect the features (layers # 82, 94, 106).

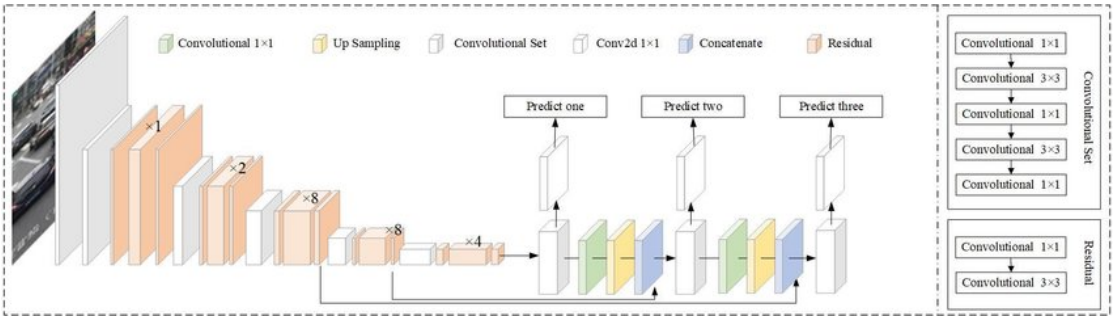


Figure 4.13: YOLOv3 architecture

Notice how no pooling layers are used in this architecture. Instead, YOLO presents convolutional 2×2 filters, which have the same goal of the pooling layers previously analyzed. YOLO's architecture can be seen also in another functional view and the network could be considered as the union of:

- a **backbone**, which is a convolutional neural network responsible for aggregating and forming image features at different definitions;
- a **neck**, that is responsible for the combining features processes;
- a **head**, which has the prediction decision layers.

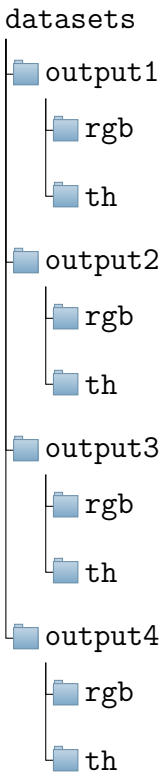
To conclude this section, it is important to highlight the differences among the different algorithms' versions, since in this work has been used the latest available, which is YOLOv5:

- **YOLO** [31]: first version released, achieved a mAP (*mean Average Precision*) score of 63.4 at a speed of 45 FPS;
- **YOLOv2** [32]: main improvement was the multi-scale training that allowed the model to predict at different input sizes. It achieved a mAP score of 76.8 at a speed of 67 FPS;
- **YOLOv3** [33]: showed the first introduction of the DarkNet53 backbone, which improved considerably the speed of the network. It achieved the same precision of the SSD321 (Single-Shot Detector) model, with a speed improved by a factor of 3;

- **YOLOv4** [29]: a massive improvement was made thanks to the introduction of different new features that are used in this model combined, which means that some of them work only on a specific problem statement or on a specific dataset. Some examples are: weighted residual connections, cross stage partial connections, cross mini-batch normalization, self-adversarial training and mish activation. For a further explanation the reader is referred to the paper that can be found in bibliography;
- **YOLOv5**: released only two months after the previous version, the main difference is the implementation in the PyTorch environment, done in order to remove the limitations due to the C programming language, on the which the v4 is based.

4.2 Data collection

The experimental part of this Dissertation consists, essentially, in the a mobile robot exploration (a perception system basis) of the different floors of the building (i.e., indoor scenario) in order it to record relevant frames that depict people during their normal daily activities.



The driving of the mobile platform is done through a remote controller that communicates with the PC, which is itself connected to the robot's motor and steering apparatus through ROS (an explanatory diagram can be found at the end of the thesis, in the appendix). Again thanks to a few lines of ROS code, inputs are recorded through the RGB and thermal cameras. By the end of the exploring tours what is obtained as output will be 4 folders, one for each inspected floor, with two subfolders, one for each camera.

The choice to use only 2 of the 4 available sensors is due to the fact that it's not necessary to overburden the outputs and consequently the convolutional neural networks and in order to reach the proposed goals we need only two outputs. Moreover, the chosen sensors (RGB and FLIR cameras) provide sufficient

coverage in every possible lighting and visibility condition and can be considered complementary. In Fig. 4.14 is represented an example of a frame and a depiction of the mobile platform busy in capturing data for a dataset in the ISR building.

Figure 4.14: From left to right: 1) Mobile platform at work in the fourth floor of the ISR building, producing output4 2) Detail from output2/th/002203.png 3) Detail from output2/rgb/002203.png

4.3 Data processing

After data collection, the first thing to do is process the frames in order to get data which is compatible with the algorithm's way to process images and labels. Since the two sensors (RGB and FLIR) that have been used in this thesis have been set to work at different frequencies (FLIR @30Hz, RGB @60Hz) it is necessary to have the same number of frames for the two sensors. For this reason, half of the frames captured from the Ximea camera are useless and deleted from the dataset. The final number of frames to be processed for each sensor is reported in Tab. 4.2.

Another difference between the two sensors is the different region of action of the cameras (this is shown in Fig. 4.15), and because of this the RGB's output frames have been cropped partially.

output	1	2	3	4	tot
number of frames	4666	2483	1348	2835	11331

Table 4.2: Number of frames to be processed

Figure 4.15: As can be seen by this frame comparison, the FLIR camera has a reduced region of action with respect to the RGB camera's one.

Moreover, an opportune labelling process has to be done in order to properly train the model. To do this, a tool called ImageLabeller belonging to the Computer Vision toolbox from the MATLAB environment has been used. The most representative frames are selected, where people are depicted and in good visibility conditions on both cameras' field of view. The chosen frames are labelled and used for the training phase of the project. In Fig. 4.16 are reported two different frames for sake of example.

Last thing to be done is to convert the labels in a suitable representation for the YOLO algorithm. It has been used a short script to do the conversion from the ground-Truth kind of data (MATLAB tool's output), and a simple text file for each frame, needed as input for the YOLO algorithm. The text file must have the following format: `n x y w h`

where each line corresponds to a different label, n_i is the number of the class, which in our case will be always 0 because the algorithm will be dealing with only one kind of object, *i.e.* people, x, y are the coordinates of the upper-left point of the rectangle and w, h are the width and the height. Note that all of these parameters (except the first one) must be normalized and expressed as a number between 0

Figure 4.16: Two different frames labelled. Note that labels are not always exactly the same for the two sensors. Moreover, the second frame shows how in low visibility environments it is necessary to adopt a second sensor, in this case a thermal camera.

and 1.

4.3.1 Training

The training phase of a model has the objective to find the best values of weights (see Subs. 4.1.2 as reference) for the nodes of the network. In particular it is necessary to feed the YOLO network with two folders: one contains images for the training and the other one contains labels in text format as specified above. In this way, each image file has a corresponding label file with one line for each label in the picture. A subset of 81 frames was chosen and the training was repeated twice, once for each sensor data.

The first important step is to setup the repository of Yolov5 and install the requirements necessary to run the algorithm. Note that all the code in this section is Python and exploits the PyTorch environment.

```
!git clone https://github.com/ultralytics/yolov5  # clone
%cd yolov5
%pip install -qr requirements.txt
```

```
import torch
import utils
display = utils.notebook_init()  # check
```

Now it is possible to use the network and the following command is useful for the training process:

```
!python train.py
--cfg yolov5x.yaml --img 640 --batch 14 --epochs 590
--data yolov5/data/config_rgb.yaml --weights '' --cache
```

The used parameters will be analyzed here:

- `train.py` is the script responsible for the training of the network;
- `--cfg` stands for 'configuration' and `yolov5x.yaml` is the specific model version to be used for training;
- `--img 640` is the dimension of the image to be cropped. In fact, YOLO uses a square image as input and automatically crops all images into sub-images of the desired dimension. In this case the predefined value of 640x640 is used;
- `--batch 14` is the number of samples used before the model is updated;
- `--epochs 590` is the number of times the dataset is passed to the algorithm. After each epoch, the accuracy and precision parameters should improve;
- `--data ...` is the configuration file where can be found some parameters like the images' and labels' path and the name of the labels. Of course this file is different for the FLIR and the RGB inputs;
- `--weights` usually takes as input a .pt file where the weights can be found. In this case, no file was chosen so that the model will take random values as first values. Note that it's necessary to chose a model in the `--cfg` parameter if the argument of `--weights` is not present (like in this case);
- `--cache` allows the creation of a cache file.

The runtime is usually about several hours to train the dataset, and it is strongly affected by the number of epochs to be completed (and consequently by the accuracy goal value to be reached). In Fig. 4.17 can be visualized the exact input that is fed to the network for both channels. The results are reported in the Subs. 4.3.2 because the accuracy parameters are computed after the validation.

Figure 4.17: Note how the images are automatically decomposed and combined in square frames (640x640) in order to be fed into the network. The "0" refers to the "person" label.

4.3.2 Validation

Inside the `train.py` code there is a section that automatically validates the weights found for each epoch and computes the accuracy parameters. Those parameters can be found in the Tab. 4.3 and are referred to the last epoch (590-th). The predicted labels for the training dataset are reported in Fig. 4.18. In the figure it is possible to notice how the confidence is not always equal to 1 (it's that number next to the label "person"), and the bounding boxes do not perfectly overlap with the labelled ones. This is because the accuracy parameters will never reach the perfection, although they result to be very good values, close enough to 100%.

A new precision parameter can be found in the table, called **mAP** (which stands for *mean Average Precision*) and it refers to different values of **IoU** (which stands for, as previously explained, *Intersection over Union*). In a simple way, it can be

Figure 4.18: Predicted labels, with respective confidence value.

said that these two values indicate the mean precision of prediction on bounding boxes that are at a certain percentage overlapping the labels. Intuitively it is possible to understand how the value of mAP @ 50% is always larger than the one of mAP @ 80%, for instance.

Sensor name	Precision	Recall	mAP @ 0.5	mAP @ 0.5:0.95
RGB	0.946	0.955	0.980	0.784
FLIR	0.958	0.919	0.967	0.774

Table 4.3: Train and validation results

In Figure 4.19 are reported the plots of precision, recall and mAPs over epochs. It is interesting to see how they tend to grow after each iteration, and reach acceptable values only after several epochs.

4.3.3 Inference

In this subsection the results of the detection over the dataset previously acquired will be shown. In order to let the neural network detect from a source, it's possible to use the following line of code:

```
!python detect.py --weights /content/drive/weights/best.pt
--img 640 --conf 0.25 --source /content/drive/flir/output1.avi
```

and the parameters are:

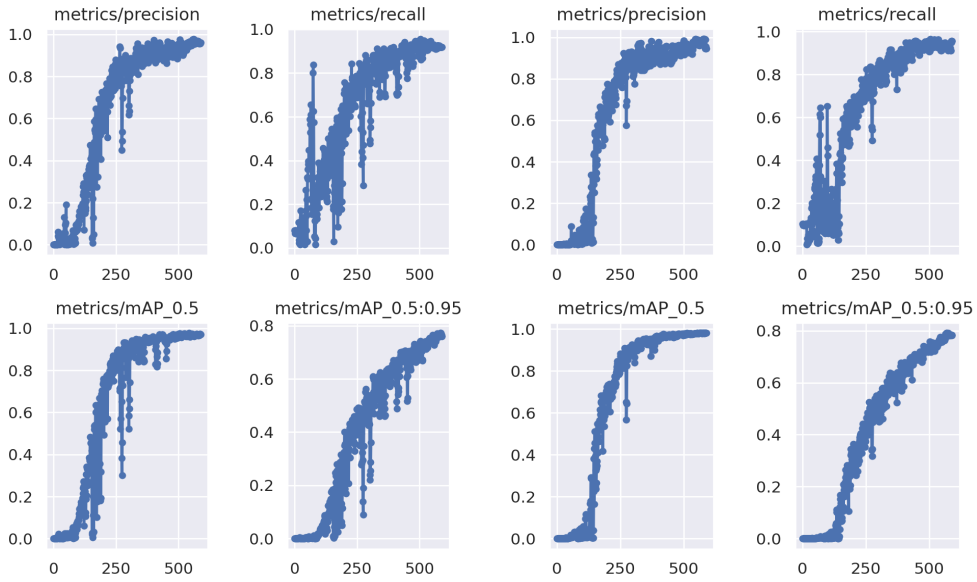


Figure 4.19: Comparison between the accuracy parameters' trends of the FLIR (left side) and of the RGB (right side) sensors' data.

- `detect.py` is the script to be run in order to start the detection;
- `--weights [path]` is the `.pt` file which is obtained in the training phase;
- `--img 640` is the dimension of the squares the image will be decomposed into (same as training paragraph);
- `--conf 0.25` is the confidence threshold a bounding box must overcome in order to be considered a valid label and establish a detection;
- `--source [path]` is the file that will be labelled. In this case we have 8 different videos to process (4 outputs for each of the two sensors).

Some frames taken from the network's detection results are shown in Fig. 4.20.

4.4 Sensors fusion

In order to obtain a robust detection system, it is useful to adopt one among the typical techniques of sensors combination/fusion. The main ones are reported here:

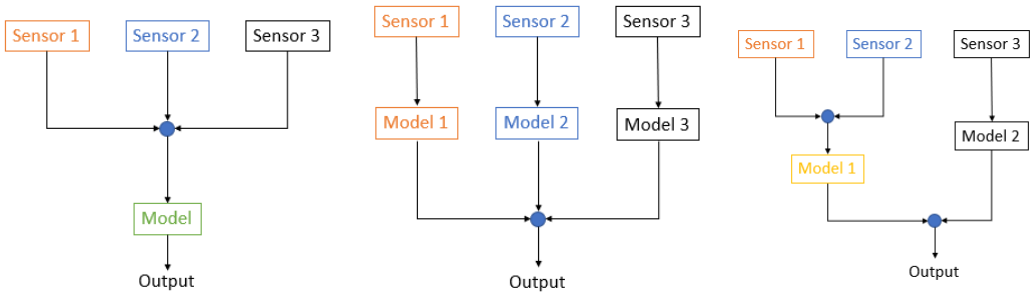
Figure 4.20: The results of the detection process.

- **early fusion**: it is a technique in which the fusion happens at data level, before entering into the detection stage (see Figure 4.21a). There are mainly two ways of doing this: combining data by removing the correlation between the two sensors or fusing data at their lower dimensional space e.g., thanks to statistical tools like *principal component analysis (PCA)*;

- **late fusion**: in this case, the fusion happens in the decision stage. Data coming from different sensors enter independently the neural network and the outputs of it are then re-combined, as shown in Figure 4.21b;

- **intermediate fusion**: this last solution can be collocated somewhere in between the other two. Thanks to intermediate fusion it is possible to combine

features, changing input data into a higher level of representation through multiple layers. In this case, the model learns a joint representation of each of the modalities. A schematic can be found in Figure 4.21c.



(a) Early fusion **(b)** Late fusion **(c)** Intermediate fusion

Figure 4.21: Fusion techniques simplified schematics. Here are reported three generic examples with 3 sensors data as inputs.

For the purpose of this thesis, a **late fusion** technique has been implemented. In order to do this, it is very important to use the results of the calibration phase of this work. In fact, it is crucial to have bounding boxes coordinates that refer to the same world coordinate frame. The outputs of the models *i.e.* , bounding boxes coordinates with a confidence score value associated, are going to be compared and combined, so they have to be computed in the same reference frame. To do it, the transformation depicted in the Equation 3.8 has been used, and the coordinates of the bounding boxes detected by the model with the thermal camera data as input were transformed to have a coherent reference system with respect to the color camera one. Once the two bounding boxes are comparable, it is possible to compute a measure of the overlapping of the two, using the ratio between the intersection and the union of the two rectangles. From this point, it has been used a decision model called **Support Vector Machine**. In the following subsection a little panoramic about this technique will be reported.

4.4.1 Linear Support Vector Machine

Given a linear function $f(x) = w_0 + w^T x$ it is possible to define an hyperplane χ such that:

$$\chi = \{x \in \mathbb{R}^n : w_0 + w^T x = 0\} \tag{4.1}$$

χ separates the hyperspace into two parts, where one above it and one below it can be expressed by:

$$\chi_+ = \{x \in \mathbb{R}^n : w_0 + w^T x \geq 0\} \qquad \chi_- = \{x \in \mathbb{R}^n : w_0 + w^T x \leq 0\} \tag{4.2}$$

and the distance between one point x and the hyperplane is:

$$dist(x, \chi) = -\frac{w_0 + w^T x}{||w||_2} \tag{4.3}$$

It is possible to classify point-features according to the fact that they belong either to one or to the other part of the hyperspace. In this sense, the hyperplane χ can be considered as a **decision boundary**. Let's suppose to have as output a label y_i which is equal to:

$$y_i = \begin{cases} +1 & \text{if} \quad x_i \in \chi_+ \\ -1 & \text{if} \quad x_i \in \chi_- \end{cases} \tag{4.4}$$

In this particular case, it is easy to notice how the product $y_i \times (w_0 + w^T x)$ is always positive in the case of a correct detection and negative for bad detection. We can then use as a loss function for training purposes the following one:

$$L(w_0, w) = \sum_{i=1}^{N} [-y_i \times (w_0 + w^T x_i)]_+ = \sum_{i=1}^{N} max((-y_i \times (w_0 + w^T x_i)), 0) \tag{4.5}$$

Notice that this function is equal to zero for the points that are correctly classified and proportional to the distance between the point x_i and the decision boundary for the points misclassified. The loss function leads to a minimization problem that

can be formulated as follows:

$$min \quad \frac{1}{2}||w||_2^2 + c\sum_{i=1}^{N}\xi_i \qquad s.t. : y_i(w_0 + w^T x_i) \geq 1 - \xi_i \qquad \xi_i \geq 0 \tag{4.6}$$

In the previous equation it was inserted a "slack variable" ξ_i. This is due to the fact that it is not always feasible to perfectly separate the points that are labelled into the two semi-hyperspaces. ξ_i represents the amount of misclassification of the i-th point, while c is a trade-off parameter. The one formulated in the expression 4.6 is a quadratic problem that is *always* feasible.

4.4.2 Application of SVM to case of study

In the presented case of study, it is necessary to define the feature space of the support vector machine. The inputs must be quantities that derive from the detections at the previous layers and must give an idea of the confidence of the detection and of the percentage of overlapping area of the bounding boxes. The objective of this part of the work is to improve detecting accuracy, by eliminating false positive detections. In a more practical way, the goal is to exclude all those bounding boxes which do not have a strong correspondence between the two sensors. In order to do this, it is important to properly train the SVM model *i.e.* , compute the optimal parameters w_0, w. This is done automatically thanks to the Python framework scikit-learn [34]. The model is trained using a dataset composed of two features as inputs: intersection over union and confidence scores. The labels will be two classes: $+1$ for the bounding boxes with good overlapping and confidence score and -1 for the ones that we consider as false positives and with bad overlapping correspondence. After the model has been trained, the SVM is used to predict labels for the whole dataset. For the bounding boxes predicted as true positives, we consider the maximum between the two scores and the corresponding bounding box.

In Figure 4.22 a scatter plot is showing the decision boundary (solid line) and the slacked decision boundaries (dotted lines). It can be seen how the algorithm manages to detect the bad predictions, which are characterized to have a small confidence score and a small overlapping area of the bounding boxes. The number

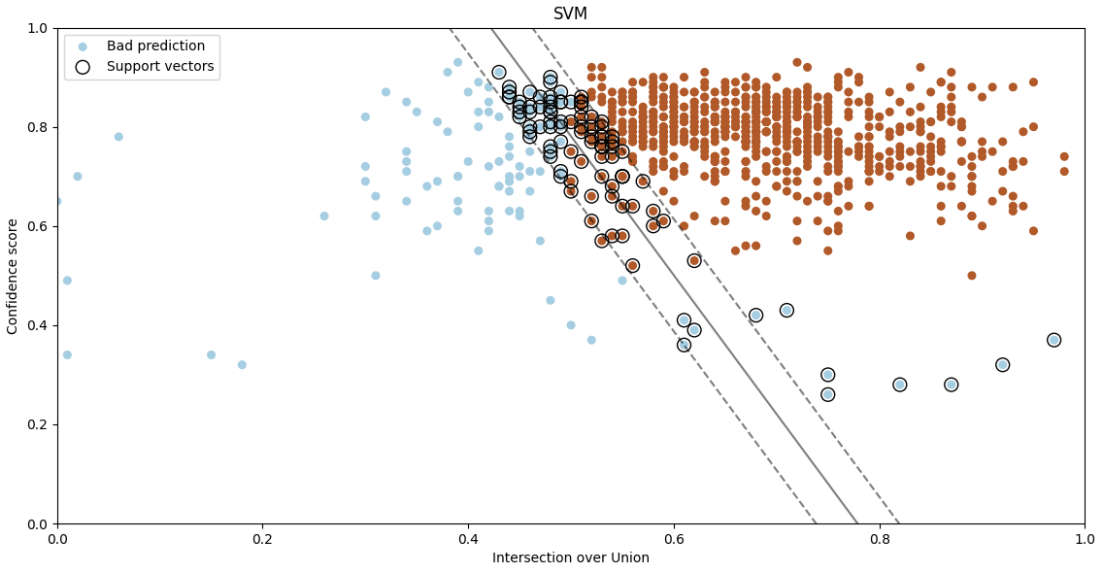


Figure 4.22: Graphic representation of the support vector machine training results

of support vectors can be changed; in this case has been used a number of 50 vectors for each label, and this choice has lead to a good result. The decision boundary is then used to predict with more precision the other outcomes, and a validation phase is done in order to compute the effectiveness and the accuracy of the algorithm. In the Table 4.4 the confusion matrix of the experiment is reported.

	P' (Predicted)	**N' (Predicted)**
P (Actual)	1157	36
N (Actual)	24	772

Table 4.4: Confusion matrix after data fusion

In the Table 4.5 the main accuracy results after data fusion are reported.
The comments about the comparison of the single mode *vs* multimodal results will be present in Chapter 5. In conclusion, in Figure 4.23 a summary schematic is reported, where the theory of the data fusion section is applied to a particular frame for the detection.

Precision	Recall
0.98	0.96

Table 4.5: Accuracy parameters after data fusion.

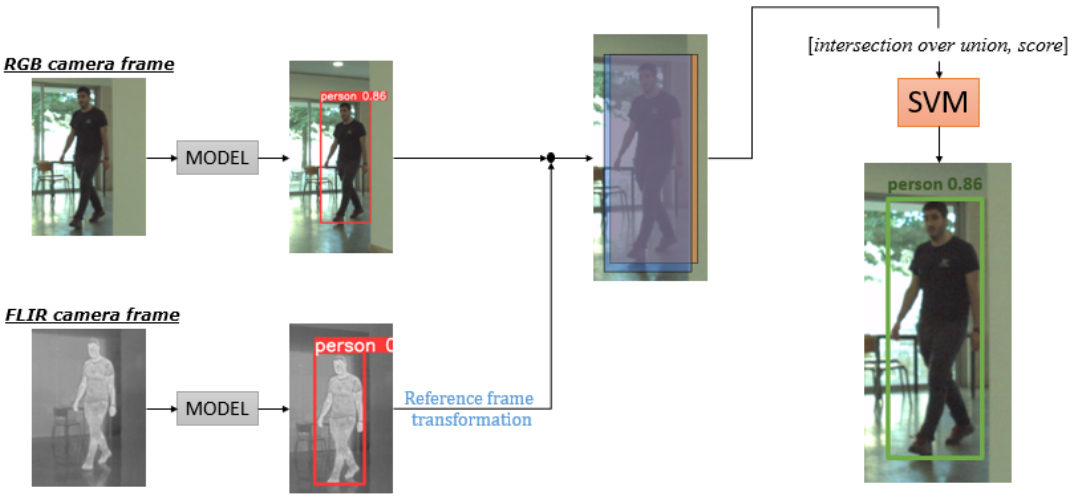


Figure 4.23: Full data fusion process schematic

Chapter 5

Discussion and conclusion

This dissertation exploits the powerful means of artificial intelligence and deep learning for machine perception purposes in an application domain related to robot perception. The application of a CNN and then a late-fusion approach neural network was necessary in order to obtain an effective system, capable of detecting people with a reliable accuracy. The physical architecture of the system is mainly composed of a combination of 3 cameras and one lidar sensor, mounted on a mobile robotic platform, through an aluminium profile and plastic sensor supports. The full sensory system could communicate through the sensors and their connection with an onboard core laptop. The presence of different sensors, even though not all of them were used in the same measure, led to a greater reliability of detection, thanks to the redundancy and the data fusion. From an operative and practical point of view, starting with the calibration of the sensors: it was strictly necessary in order to gain awareness about internal and external parameters of the cameras, and furthermore in order to have the possibility to fuse data coming from different sensors, which have different parameters, orientation in space and field of view. The calibration phase of the work has been done following the forward imaging model theory. By a more empiric standpoint, it has been done using different tools, but mainly in the Matlab framework; this because it offers also the possibility to do the validation of the results of the calibration in a simpler manner. In this section of the work it was critical the exploitation of the lidar sensor, that gave a precise and fast estimation of the position of the calibration checkerboard in different

settings and conditions. After the collection of thousands of frames in different weather and lighting conditions, and their labelling, a fraction of them were used to train the convolutional neural network. The remaining part was used for validation purposes and for error estimations. After finding the best settings in order to get good recall and precision rates, the CNN-network was used to detect humans in a new and untouched (test) dataset. Finally, a technique of late fusion was used in order to improve the precision of the system, thanks also to a completely different machine learning algorithm, which is the Support Vector Machine. The collected results respected the expectations in terms of precision and accuracy. In fact, if we compare the Tables 4.3 and 4.5, it is possible to observe three conclusions mainly:

1. the accuracy parameters obtained from RGB and thermal cameras separately were satisfactory;

2. the precision coming from the thermal camera detecting system is slightly better compared to the one of the color (RGB) camera. As expected, the recall obtained with the color camera is significantly better than the thermal camera one. This basically means that if we use a color camera to feed a human detecting neural network, it will be more prone to commit false positives kind of errors. On the contrary, the thermal data are more affected by false negatives;

3. both precision and recall are considerably increased by the exploiting of late data fusion techniques and SVM application.

In light of this, and given the context here, it is possible to state that if we want a system that needs to avoid false positives (*i.e.* , type I errors), it is better to rely on thermal camera-based detecting structure. Differently, if it is more important to avoid false negatives (*i.e.* , type II errors), then it is better to exploit a RGB-camera detecting system. In both cases, the support of a data fusion approach will lead to an improvement of the results. Nevertheless, it is important to highlight that the collected results are certainly influenced by the difficulties met in the calibration process of the thermal camera. Probably, if we could use a different and more accurate calibration technique for the thermal camera, also the outputs of the neural network applied to the FLIR camera would have been better.

The possible improvements to be applied to this study can be divided into three main points:

1. **sensors**: it could be useful to have at disposal a more complex system of sensors, to provide better quality of data and different angles of field of view. In this sense, it can be interesting to also use cameras that are oriented to sense the back of the mobile platform, for the detection to be useful also in reverse gear. Furthermore, as stated before, it is possible to improve the quality of the calibration of the thermal camera, and eventually to use a FLIR with better definition (nonetheless, our sensor was very valuable). It is important to keep in mind also that for the purposes of this thesis, the detection phase was based only on the outputs of two of the 4 sensors available. It is possible to enhance this apparatus in order to improve accuracy. One last hypothetical scenario could be the one in which other sensors are used for detection, instead of RGB and FLIR. It can be interesting to analyze the advantages and drawbacks in a case in which a NIR camera was used instead of the color or the thermal one. Or, moreover, if the lidar data were used for the detection;

2. **real-time application**: the Yolov5 CNN gives the opportunity to detect live images coming from the sensors. Of course, this had to be supported by proper hardware and devices, that could implement in real-time the detection. In this case, it could be interesting to evaluate the opportunity to use the output of the neural network directly on the brakes of the mobile platform. So, the mobile platform would have a working detection and braking system embedded. In order to design a mobile platform able to avoid obstacles properly, it should be necessary to involve in the project some trajectory planning-related algorithm;

3. **data fusion with more inputs**: as highlighted in the Section 4.4, data fusion led to an improvement in detection results. The work in this thesis was based on data coming from two different sensors, and it is trivial to conclude that an increase in the number of the sensors feeding the Support Vector Machine detection algorithm would, in principle, result in a further improvement of the overall system performance. Moreover, data coming from the lidar sensor could be converted into depth maps and fused with other sensors, supplying the detecting apparatus with an additional data source.

Appendix A

Optical sensors models

The mechatronics laboratory of the University of Coimbra, where this whole practical experience was conducted, provided a large number of sensors that will be listed and briefly described in this section. Four plastic supports were designed in Solidworks ® environment and 3D-printed. They were useful for the attachment on the aluminium profile structure and for the protection of the sensor. A photo is reported in Fig. A.1.

For a further description of the sensors the reader is redirected to the official manuals of the sensors that can be found online.

A.1 Lidar: Velodyne®VLP-16

This sensor is widely used in guidance applications. It emits 16 layers of pulsed light waves which bounce off back to the lidar; then the sensor uses the time it took for each pulse to come back to calculate the distance of each material point in the surroundings. The VLP-16 model in particular has a 360° horizontal and a 30° vertical coverage and can be easily connected to a computer through ethernet cable for data collection and processing.

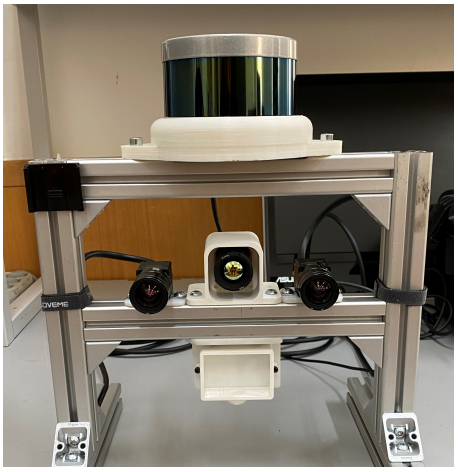


Figure A.1: The four sensors mounted on an aluminium support

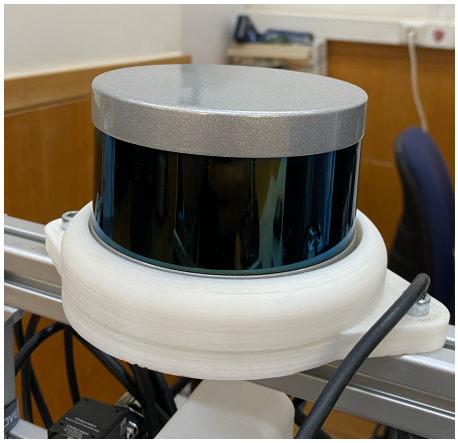


Figure A.2: Velodyne®VLP-16

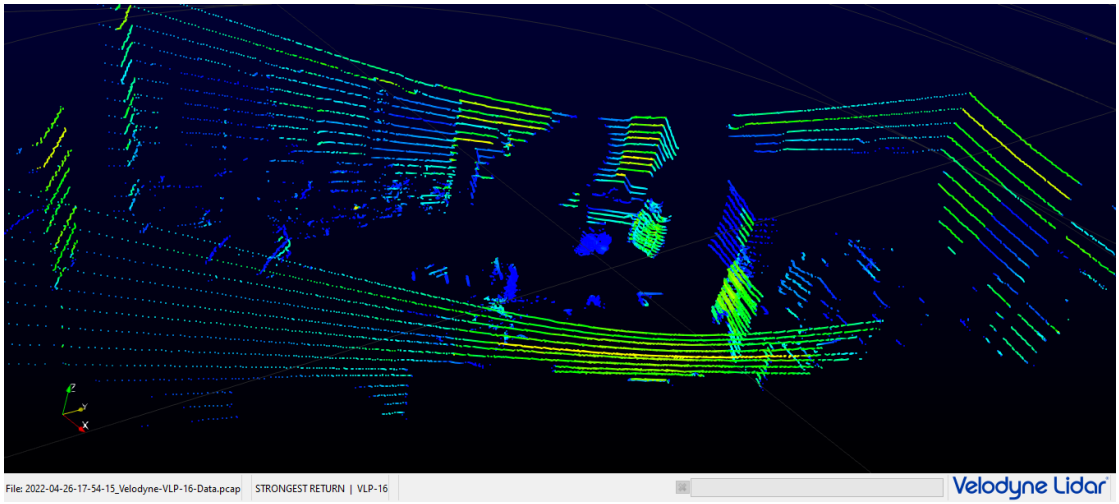


Figure A.3: Example of Velodyne®VLP-16 scan

A.2 Thermal camera: FLIR Boson®640

The thermal camera exploits the infrared-wavelength energy emitted by objects to detect them and converts it in visible-wavelength data to display as images. Its most common use is military applications but in this working case it is very useful to combine with the lidar data collection for human recognition purposes.

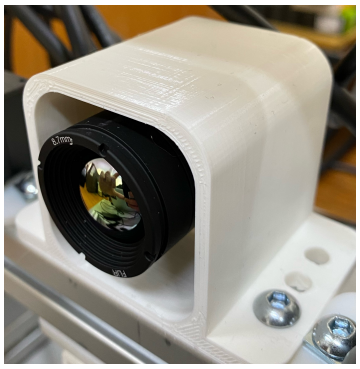


Figure A.4: FLIR Boson®640

Figure A.5: Example of FLIR Boson®640 image

A.3 RGB and NIR camera: MQ013CG-E2 and MQ013RG-E2 Ximea®

Figure A.6: MQ013RG-E2 Ximea®

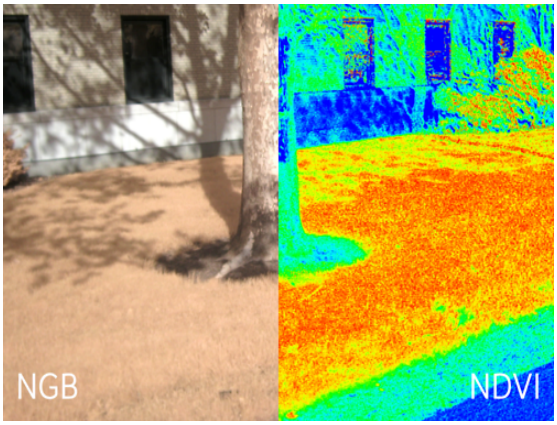


Figure A.7: Comparison between RGB (left half) and NIR (right half) camera outputs

These two cameras are very similar with each other. The substantial difference is in the captured wavelength of the light, which is red-green-blue in the first case, and near-infrared in the second. This allows the system to work properly in different light conditions.

Appendix B

Setup of the full apparatus

The cameras system is mounted on an aluminium profile support and attached on the mobile platform. The sensors are connected to a Linux-based computer that collects data through ROS, to which the three cameras are linked by means of USB 3.0 connection. The lidar sensor is connected directly to a microprocessor that is power supplied by a couple of batteries and links the sensor to the computer thanks to an ethernet connector. The two batteries also feed directly the PC and the motor of the mobile platform, which is remotely commanded by a joystick. This whole system is represented in Fig. B.1.

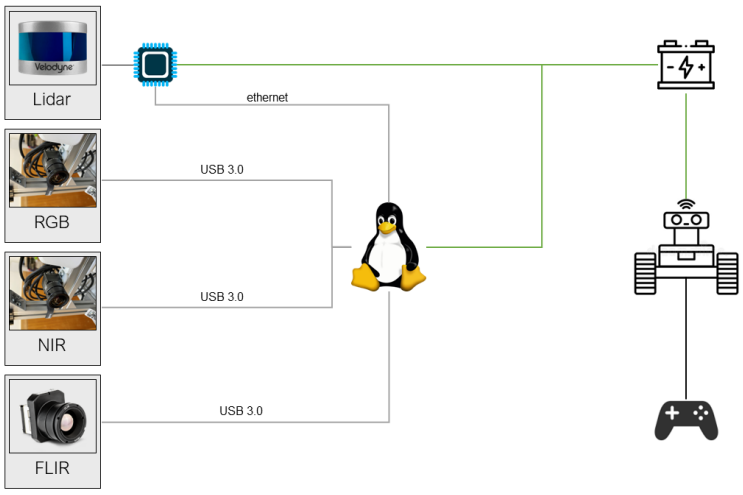


Figure B.1: Graphic representation of the full system

Appendix C

Point Clouds extraction starting from .pcap file

The Veloview software can perform a recording of the surrounding environment and export it in a .pcap file. Since the input parameters of the functions used in this work must be point clouds it is necessary to extract .pcd files starting from the whole recording. To do this the most relevant frames (in which the checkerboard is steady and in the range of the lidar) are selected and then exported thanks to the following Matlab script:

```
clc
clear all
close all
%Saving the PCD files of the data
calibfolder = 'C:\Users\feder\Desktop\Dissertation\01-Calibration\
    calib rgb-lidar';
pcdfolder = fullfile(calibfolder, '4\pcd');
%Reading the lidar file
veloReader = velodyneFileReader(fullfile(calibfolder,'4\videos\lidar
    scan.pcap'),'VLP16');
%Limits of the Lidar
xlimits = [-2 4];
ylimits = [-4 8];
zlimits = [-2 2];
player = pcplayer(xlimits,ylimits,zlimits);
%Label the Axes
xlabel(player.Axes,'X (m)');
```

```
ylabel(player.Axes,'Y (m)');
zlabel(player.Axes,'Z (m)');
frame=[388 459 483 540 571 612 648 690 744 774 798 825 865 924 963];
totalframe=length(frame);
i=1;
%Display
while(hasFrame(veloReader) && player.isOpen() && (i<=totalframe))
    ptCloud = readFrame(veloReader,frame(i));
    ptCloud = pointCloud(reshape(ptCloud.Location, [],3), 'Intensity'
    ,single(reshape(ptCloud.Intensity, [],1)));
    name = sprintf('%02d.pcd',i);
    pcwrite(ptCloud, fullfile(pcdfolder,name));
    view(player,ptCloud);
    i=i+1;
end
```

After running the script in the output folder will be found one pointcloud file with pcd extension for each relevant frame (xx.pcd with xx=increasing number from 01 to the number of frames) and will be visualized each of them as shown in the Fig. C.1.

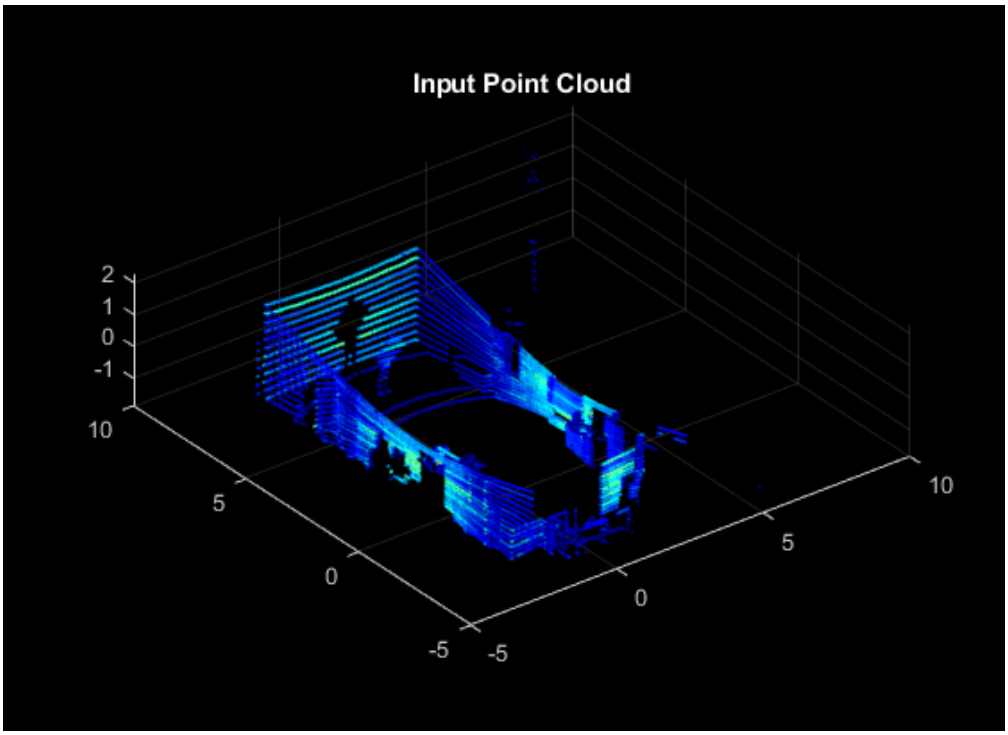


Figure C.1: Point Cloud visualization for the 11^{th} relevant frame

The number of the relevant frames in this work is 14, so the lidar-camera calibration dataset will be composed of 14 couples of images coming from the RGB camera and pointclouds coming from the lidar.

Bibliography

[1] Ibrar Yaqoob, Latif U Khan, SM Ahsan Kazmi, Muhammad Imran, Nadra Guizani, and Choong Seon Hong. «Autonomous driving cars in smart cities: Recent advances, requirements, and challenges». In: *IEEE Network* 34.1 (2019), pp. 174–181 (cit. on p. 1).

[2] Klaus R. Kunzmann. «Smart cities: a new paradigm of urban development». In: (). URL: http://www.carocci.it/files/riviste/digitali/01_kunzmann.pdf (cit. on p. 1).

[3] Sebastian Thurun. *What we're driving at.* 2010. URL: https://googleblog.blogspot.com/2010/10/what-were-driving-at.html (cit. on p. 2).

[4] John Markoff. «Google Cars Drive Themselves, in Traffic». In: (2010). URL: https://www.nytimes.com/2010/10/10/science/10google.html?smid=url-share (cit. on p. 2).

[5] Romuald Aufrère, Jay Gowdy, Christoph Mertz, Chuck Thorpe, Chieh-Chih Wang, and Teruko Yata. «Perception for collision avoidance and autonomous driving». In: *Mechatronics* 13.10 (2003), pp. 1149–1161 (cit. on p. 2).

[6] M.I. Jordan and T.M. Mitchell. «Machine learning: Trends, perspectives, and prospects». In: (). URL: http://www.cs.cmu.edu/~tom/pubs/Science-ML-2015.pdf (cit. on p. 2).

[7] Hyunggi Cho, Young-Woo Seo, BVK Vijaya Kumar, and Ragunathan Raj Rajkumar. «A multi-sensor fusion system for moving object detection and tracking in urban driving environments». In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 1836–1843 (cit. on pp. 5, 6).

[8] J Burlet and M Dalla Fontana. «Robust and efficient multi-object detection and tracking for vehicle perception systems using radar and camera sensor fusion». In: *IET and ITS Conference on Road Transport Information and Control (RTIC 2012)*. IET. 2012, pp. 1–6 (cit. on pp. 5, 7).

[9] Cristiano Premebida, Oswaldo Ludwig, and Urbano Nunes. «LIDAR and vision-based pedestrian detection system». In: *Journal of Field Robotics* 26.9 (2009), pp. 696–711 (cit. on p. 5).

[10] Zhi Yan, Li Sun, Tom Duckctr, and Nicola Bellotto. «Multisensor online transfer learning for 3d lidar-based human detection with a mobile robot». In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 7635–7640 (cit. on p. 5).

[11] Gonçalo Monteiro, Cristiano Premebida, Paulo Peixoto, and Urbano Nunes. «Tracking and classification of dynamic obstacles using laser range finder and vision». In: *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2006, pp. 1–7 (cit. on p. 6).

[12] Thomas Herpel, Christoph Lauer, Reinhard German, and Johannes Salzberger. «Multi-sensor data fusion in automotive applications». In: *2008 3rd International Conference on Sensing Technology*. IEEE. 2008, pp. 206–211 (cit. on p. 6).

[13] Cristiano Premebida, Luis Garrote, Alireza Asvadi, A Pedro Ribeiro, and Urbano Nunes. «High-resolution lidar-based depth mapping using bilateral filter». In: *2016 IEEE 19th international conference on intelligent transportation systems (ITSC)*. IEEE. 2016, pp. 2469–2474 (cit. on pp. 7, 8).

[14] Z. Zhang. «A flexible new technique for camera calibration». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334. DOI: 10.1109/34.888718 (cit. on p. 8).

[15] Jean-Yves Bouguet. *Camera Calibration Toolbox for Matlab*. en. 2022. DOI: 10.22002/D1.20164. URL: https://data.caltech.edu/records/20164 (cit. on pp. 8, 23).

[16] Zean Bu, Changku Sun, Peng Wang, and Hang Dong. «Calibration of Camera and Flash LiDAR System with a Triangular Pyramid Target». In: *Applied Sciences* 11.2 (2021). ISSN: 2076-3417. DOI: 10.3390/app11020582. URL: https://www.mdpi.com/2076-3417/11/2/582 (cit. on pp. 8, 9).

[17] Ruixuan Liu, Hengrui Zhang, and Sebastian Scherer. «Multiple Methods of Geometric Calibration of Thermal Camera and A Method of Extracting Thermal Calibration Feature Points». In: (2018). URL: https://henryzh47.github.io/assets/documents/multiple-methods-geometric.pdf (cit. on p. 9).

[18] Jahanzaib Shabbir and Tarique Anwer. «A Survey of Deep Learning Techniques for Mobile Robot Applications». In: *ArXiv* abs/1803.07608 (2018) (cit. on p. 9).

[19] Benjamin Lewandowski, Jonathan Liebner, Tim Wengefeld, Steffen Müller, and Horst-Michael Gross. «Fast and Robust 3D Person Detector and Posture Estimator for Mobile Robotic Applications». In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 4869–4875. DOI: 10.1109/ICRA.2019.8793712 (cit. on p. 10).

[20] Junwoo Lee and Bummo Ahn. «Real-Time Human Action Recognition with a Low-Cost RGB Camera and Mobile Robot Platform». In: *Sensors* 20.10 (2020). ISSN: 1424-8220. DOI: 10.3390/s20102886. URL: https://www.mdpi.com/1424-8220/20/10/2886 (cit. on pp. 10, 11).

[21] Florian Spiess, Lucas Reinhart, Norbert Strobel, Dennis Kaiser, Samuel Kounev, and Tobias Kaupp. «People detection with depth silhouettes and convolutional neural networks on a mobile robot». In: *Journal of Image and Graphics* 9.4 (2021), pp. 135–139 (cit. on p. 10).

[22] Connor Shorten, Taghi M. Khoshgoftaar, and Borko Furht. «Deep Learning applications for Covid 19». In: 2021 (cit. on p. 44).

[23] F Alam et al. «Fighting the COVID-19 infodemic in social media: a holistic perspective and a call to arms». In: 2020 (cit. on p. 44).

[24] Jia Deng and al. «ImageNet: A Large-Scale Hierarchical Image Database». In: *Computer Vision and Pattern Recognition*. 2009 (cit. on p. 45).

[25] *"Anatomy and Physiology" by the US National Cancer Institute's Surveillance, Epidemiology and End Results (SEER) Program*. 2019. URL: https://it.m.wikipedia.org/wiki/File:Neuron.svg (cit. on p. 46).

[26] Vernon B. Mountcastle. «Modality and topographic properties of single neurons of cat's somatic sensory cortex». In: *Journal of Neurophysiology*. 1957 (cit. on p. 46).

[27] *"A simplified view of an artifical neural network", Vectorized by Mysid in CorelDraw on an image by Dake*. 2006. URL: https://it.m.wikipedia.org/wiki/File:Neural_network.svg (cit. on p. 47).

[28] Timea Bezdan and Nebojsa Bacanin. «Convolutional Neural Network Layers and Architectures». In: Jan. 2019, pp. 445–451. DOI: 10.15308/Sinteza-2019-445-451 (cit. on p. 52).

[29] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. «YOLOv4: Optimal Speed and Accuracy of Object Detection». In: *CoRR* abs/2004.10934 (2020). arXiv: 2004.10934. URL: https://arxiv.org/abs/2004.10934 (cit. on pp. 52, 53, 56).

[30] Abdulhamit Subasi. «Chapter 3 - Machine learning techniques». In: *Practical Machine Learning for Data Analysis Using Python*. Ed. by Abdulhamit Subasi. Academic Press, 2020, pp. 91–202. ISBN: 978-0-12-821379-7. DOI: https://doi.org/10.1016/B978-0-12-821379-7.00003-5. URL: https://www.sciencedirect.com/science/article/pii/B9780128213797000035 (cit. on p. 53).

[31] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. «You Only Look Once: Unified, Real-Time Object Detection». In: (2016) (cit. on p. 55).

[32] Joseph Redmon and Ali Farhadi. «YOLO9000: Better, Faster, Stronger». In: (2017) (cit. on p. 55).

[33] Joseph Redmon and Ali Farhadi. «YOLOv3: An Incremental Improvement». In: (2018) (cit. on p. 55).

[34] F. Pedregosa et al. «Scikit-learn: Machine Learning in Python». In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 67).