UNIVERSITÀ DEGLI STUDI DI PADOVA

DEPARTMENT OF INFORMATION ENGINEERING

*Master Degree in ICT for Internet and Multimedia*

# RGB-D Multicamera Object Detection and Tracking Implemented through Deep Learning

*Supervisor*

**Dr. Stefano Ghidoni**

*Master Candidate*

**GuangZheng Zhang**

**Student ID: 1178043**

ACADEMIC YEAR 2018/2019
DATE: 07/10/2019
SESSION: THIRD PERIOD 2018/2019

**Abstract**

In this thesis we present the development of a multi object detection and tracking system in low light environment implemented by using a RGB-D multicamera system and the deep learning framework. For better understanding how the system works, some hardware and software components are presented such as RGB-D sensor cameras, multi object detection and tracking techniques. In addition a brief introduction of the main concepts of the neural networks are presented.

**Abstract**

In questa tesi viene presentato un sistema di identificazione e tracking di oggetti in ambienti poco illuminati implementato usando un sistema a multicamere RGB-D e l'ambiente del deep learning. Per capire meglio come il sistema funzioni, alcune componenti hardware e software vengono presentate e descritte come i sensori RGB-D, tecniche di identificazione e tracking di più oggetti. Inoltre viene presentata una breve introduzione ai concetti principali delle rete neurali.

# Contents

# Chapter 1

# Introduction

## 1.1 Introduction

Multi object recognition and tracking are two challenging problems in the computer vision applications, so a lot of works, studies and researches are done in these two areas.
In the recent years object recognition algorithms, in the RGB image domain, have achieved good results for a lot of computer vision, artificial intelligence and autonomous robotics applications [1, 2, 3].
With the advent of the *Convolutional Neural Network (CNN)* and the *Deep Neural Network (DNN)*, it is possible to learn and extract more accurate features with respect to the hand-crafted feature representations such as *Scale-Invariant Feature Transform (SIFT), Histogram of Oriented Gradients (HOG) and Spatio-Temporal Laplacian Pyramid Coding (STLPC)* [1] and consequently it is possible to achieve high accuracy detections and make the object recognition more robust against false detections [4].
However, there are still several limitations for the object recognition using data information only from the RGB image domain in many real world applications [5]. These limitations are caused by the fact that the real world applications operate in a 3 dimensional space while the RGB domain is only a projection of the 3D space into a 2 dimensional space which leads to an inevitable data loss. Moreover, object recognition faces other challenging problems such as complicated background, illuminance variations and occlusions. Thanks to the development of low cost depth sensor cameras such as Intel RealSense and Azure Kinect, that allow to provide high quality RGB images and to open a new dimension, *i.e. depth data for each pixel*, it is possible to overcome the above described challenges [1, 4]. These depth sensors give the possibility to use depth information of a scene in order to extract

more informative features since the depth data is more robust against the color, illuminance, rotation angle and scale variations with respect to the RGB images. Despite all the success obtained in the object recognition with RGB images, in the *RGB plus Depth (RGB-D)* image domain this task is still an open problem. Thanks to the more common use of applications that use this type of data, the RGB-D image domain becomes an interesting research area and allows the researchers to study and implement a lot of new applications such as autonomous driving [6], 3D object recognition [7, 8, 9] and robot objects picking [10], that are difficult to implement without depth information of the scene.

Object tracking is a novel procedure for detecting moving objects beyond time by utilizing video sequences and in the recent years it becomes, with the object recognition, one of the most challenging problem for computer vision applications and intelligent video surveillance systems [11]. The principal aim of the object tracking algorithm is to relate the target objects, their features or shapes, their location in the scene in a successive video sequences. However there are several issues like occlusion of an object in the scene, objects overlapping, deformation, complex object motion and real-time processing requirements [11, 12, 13]. All these issues can make the tracking systems inaccurate and causing a drift in the tracking process and consequently it is possible to lost the tracked target [11, 14]. In addition, the object classification and detection are essential for all object tracking algorithms, so a robust object detection algorithm can contribute to obtain a good object tracking system.

Since in a video surveillance system there is the need to cover a wide area and due to the finite camera field of view, it is difficult to track the complete object trajectory of the object of the interest in a wide area. For these reasons, a multi camera system is introduced in order to solve the above problems [14]. By introducing the multicamera system, the problem of overlapping in the cameras views is introduced and how to link the different images captured from the various cameras become an important research argument [14, 15].

A multi object tracking system aims to find the optimal trajectories of a set of moving targets in a video sequence that can have an indefinitely duration. This task is typically formulated as a data association task in which an object detection algorithm detects and localizes the objects and their bounding boxes in each frame. Finally the tracking algorithm associates the corresponding bounding boxes across the video frames [16].

However, it is also difficult to track and maintain an assigned identity for each tracked object in the multiple objects tracking system [17], especially in the case when the target objects, in some frames of the video sequence, are occluded or overlapped [18, 19].

In this thesis an object detection and tracking system developed for a low light environment will be presented. In order to better understand how the components of the system work, a review of the various RGB-D cameras present in the market, the importance of the CNN for the object detection and tracking and the usage of multicamera in various configurations will be presented. The next chapters of the thesis are organized in the following way: firstly the characteristics of the RGB-D cameras, how they work, differences between the various sensors, comparison with RGB only cameras and their advantages in low light environments will be presented in the chapter 2. Then the multicamera system will be introduced in the chapter 3, where there will be presented also the different configuration of the cameras positions and the problems related to the overlapping of the camera views. Since the usage of CNNs give an important improvement in the accuracy of the detection of the objects, this argument is presented in the chapter 4, where a description of how they work, their main concepts, different types and their contribution in the imaging processing, are discussed. In the chapter 5 the object detection and tracking process and their concepts will discussed. Then the project setup and the algorithm idealized for a low light environment, its testing settings, the results and some consideration of the algorithm will be presented in chapter 6 and final considerations, issues and future works are discussed in the chapter 7.

# Chapter 2

# RGB-D Sensor Camera

The detection and tracking system that will be presented in the chapter 6, will work in a low light environment. For this reason, the RGB-D sensor cameras are more suitable than the RGB cameras. This type of sensor camera combines the RGB color information with per pixel depth information. These type of sensors exist for years but initially their prices are very high and they are not for commercial purposes. In the recent years, more precisely in November 2010, Microsoft launched its first commercial RGB-D sensor camera named Kinect. It is developed as a new *Natural User Interface (NUI)* for its XBOX 360 gaming console with a price that is some order of magnitude cheaper than the non commercial ones. [20, 21]. After the success of the Microsoft Kinect, more and more low-cost commercial RGB-D cameras are developed by other manufacturers like Asus, Intel and Primesense with competitive prices [21]. Thanks to all these new RGB-D sensors, the researchers become more and more interested to study and develop new application, using these type of cameras, such as object reconstruction and 3D scanning, 3D mapping like the *Simultaneous Localization And Mapping (SLAM)* process [22], implementation of *Virtual Reality (VR) and 3D Augmented Reality (3D-AR)* [20] and scene reconstruction [21, 23].

## 2.1 Characteristics of the RGB-D camera

Traditionally there are different technologies of range sensing on which the low-cost RGB-D cameras mentioned above are based. These technologies can mainly grouped into two categories [21, 24]:

- Range sensing by triangulation that can be subdivided again into passive *(i.e. stereo vision)* and active *(i.e. structured light)* technologies.

- Range sensing by using the *Time of Flight (TOF)* technology.

### 2.1.1 Passive triangulation technology: Stereo Vision

Stereo vision is an area of computer vision in which the aim is the reconstruction of the 3D coordinates of the points in order to estimate its depth values [25]. Traditionally a stereo vision system is composed by the so called stereo camera that is a set of two RGB cameras aligned and placed horizontally, one on the left and one on the right (see Figure 2.1). The system capture the two images from the two cameras simultaneously, then the images are processed in order to retrieve the visual depth information by computing the disparity between the two images and creating a disparity map as the result of the process (see Figure 2.2) [21, 25].
This technology can reproduce disparity maps with high resolution at the expense of a more computation demanding stereo matching algorithms, especially when the global optimization techniques are used [26]. The stereo vision system is suitable for both outdoor and indoor environments but only in places with a good illumination since the technology is passive and it is based on the reflection of the natural light on the target in order to capture and process the images [24].

Figure 2.1: The aspect of a stereo camera.

Figure 2.2: Image captured by the left camera (left), image captured by the right camera (center) and the computed disparity map (right).

### 2.1.2 Active triangulation technology: Structured Light

Structured light sensors can be a valid alternative to the passive stereo vision cameras in some specific cases like controlled environment, weakly lighted environment and weakly textured environment [27]. A triangulation based structured light sensor is similar to the classic stereo vision sensor with one of the cameras replaced by a light source projector that can be a laser or a slide projector [27, 28]. The structured light pattern form can be a single dot, single slit, stripe patterns or grid, multiple dots (see Figure 2.3).

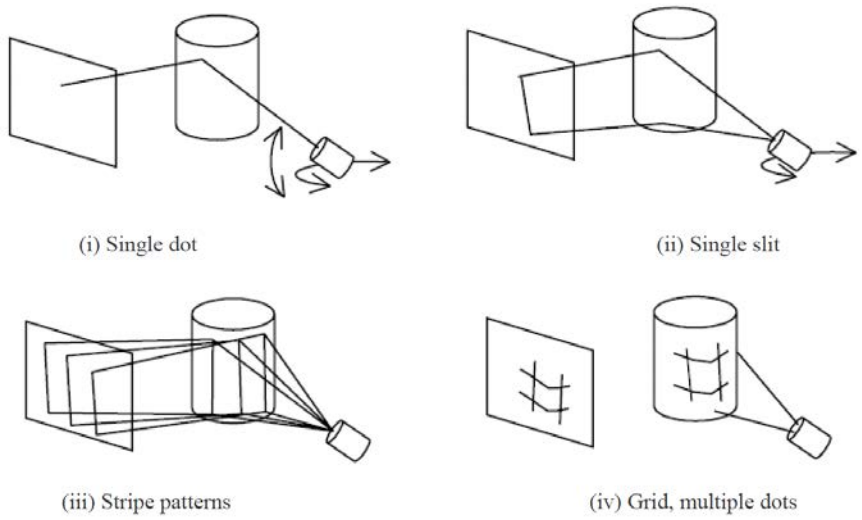



Figure 2.3: Geometry of a structured light sensor.

The structure light typology, instead, can be divided into three categories based on the type of the used light source or pattern [27]:

- *InfraRed Structured Light (IRSL)*: it use generally the near-infrared light spectrum (i.e. from 640 nm to 2500 nm) as the light source. Thanks to the spectral sensitivity of the CCD cameras (from 300 nm to 1100 nm), the is no need to use an infrared camera and a CCD camera is sufficient to acquire the images.

- *Imperceptible Structured Light (ISL)*: the system is composed by one light source and two cameras. The light source is composed by a light pattern followed by its inverse pattern and it is projected onto the scene with high frequency in order to have an uniform light pattern. One camera is synchronized with the first light pattern while the second retrieves classical gray scale or color images.

- *Filtered Structured Light (FSL)*: the light source in this case is filtered by an infrared filter, placed in front of the light source, that allows only some specific wavelengths to passing through. After the filter, it is very similar to the IRSL case.

The IRSL is one of the most commonly used technology in the low cost RGB-D cameras and these type of cameras will be used in the setup of the project that will be presented in the chapter 6.

### 2.1.3 The Time of Flight (ToF) technology

This technology is implemented in the so called ToF cameras. These cameras measure the distance of an object by calculating it from the time an emitted light signal takes to return to the camera. There are two main different ToF principle [29, 30]:

- Pulse modulated ToF: it measures the distance by means of a direct measurement of the time of a pulse light to do a round trip.
- *Continues Wave (CW)* modulated ToF: it uses an amplitude modulated light and it measures the distance by measure the phase difference between the emitted and the received signals (see Figure 2.4).

Because of the complex readout schemes and a low frame rates, the pulse modulated one is less used in the commercial 3D imaging, while the other one is already implemented in several commercial 3D camera systems (see Figure 2.5) [30].

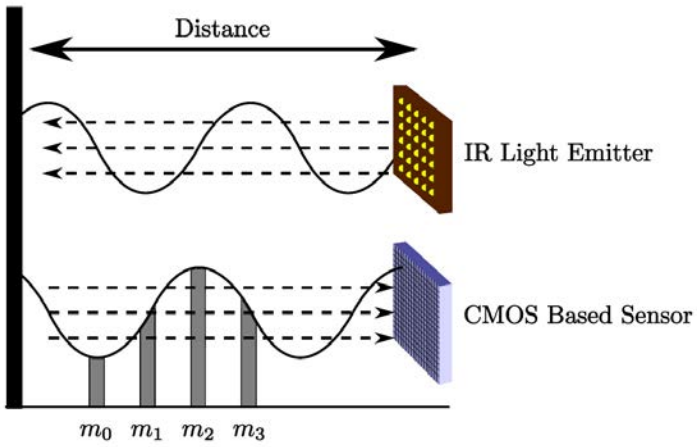



Figure 2.4: Distance measurement using the phase offset.

(a) SR3000, 176x144 (b) SR4000, 176x144

(c) O3D100, 64x48 (d) CanestaVision™, 64x64

(e) 19k, 160x120 (f) CamCube 2.0, 204x204

Figure 2.5: Current commercial ToF cameras. (a-b) Mesa Imaging AG©. (c) Ifm electronic©. (d) CanestaVision™. (e-f) PMD[Vision]®.

In the CW type, every pixel on the sensor measure 4 times the reflected light ($m_0, m_1, m_2, m_3$ in Figure 2.4) and this allows the parallel computing of the phase

$$\varphi = \arctan\left(\frac{m_3 - m_1}{m_0 - m_2}\right) \tag{2.1}$$

the offset

$$B = \frac{m_0 + m_1 + m_2 + m_3}{4} \tag{2.2}$$

and the amplitude

$$A = \frac{\sqrt{(m_3 - m_1)^2 + (m_0 - m_2)^2}}{2} \tag{2.3}$$

Using these information, it is easy to calculate the distance

$$D = L\frac{\varphi}{2\pi} \tag{2.4}$$

where $L$ is the *ambiguity-free distance range* that can be calculated, once the modulation frequency $f_m$ is known, by using the following equation

$$L = \frac{c}{2f_m} \tag{2.5}$$

where $c$ is the speed of the light in vacuum.
The ToF camera measurements usually suffer from some systematic errors such as the multipath effect, the scattering artifacts and the mixed pixel. All these errors compromise the accuracy of the measurements.

## 2.2 RGB-D Camera Devices

After the first version of Microsoft Kinect presented in 2010 and the big success that has obtained, in the recent years, lots of companies invest and develop their own RGB-D cameras for commercial purposes. Since these RGB-D cameras was created as NUI devices, their characteristics suffer some limitations like the *Field Of View (FOV)* is not enough large for mapping applications and the depth resolution deteriorates notably with the distance of the object from the camera increases (see Figure 2.6) [20].

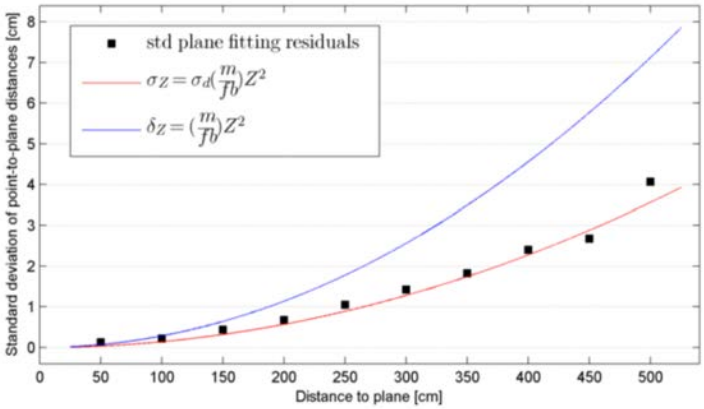



Figure 2.6: Microsoft Kinect depth resolution vs. depth (Khoshelham & Elberink 2012). Theoretical random error is shown in red (bottom curve), while the theoretical resolution is shown in blue (top curve).

In the following subsection, some RGB-D cameras are presented with their technical specifications.

### 2.2.1 Devices

#### Microsoft Kinect v1


Figure 2.7: The Microsoft Kinect v1.

This is the first commercial RGB-D camera presented in 2010 by the Microsoft for its XBOX 360 gaming console. Since it is developed as a NUI device, it has also an array of microphones (see table 2.1 for the technical specifications).

| | |
|---|---|
| **Field of View** | 43° vertical, 57° horizontal |
| **Depth Range** | 1.2 m up to 3.5 m |
| **Frame Rate (depth and color stream)** | 30 frame per seconds (fps) |
| **Stream Resolution (depth and color)** | VGA ($640 \times 480$) |
| **Audio Format** | 16 kHz, 16 bit mono Pulse Code Modulation (PCM) |

Table 2.1: The Microsoft Kinect technical specifications.

**Azure Kinect**

Figure 2.8: The Azure Kinect.

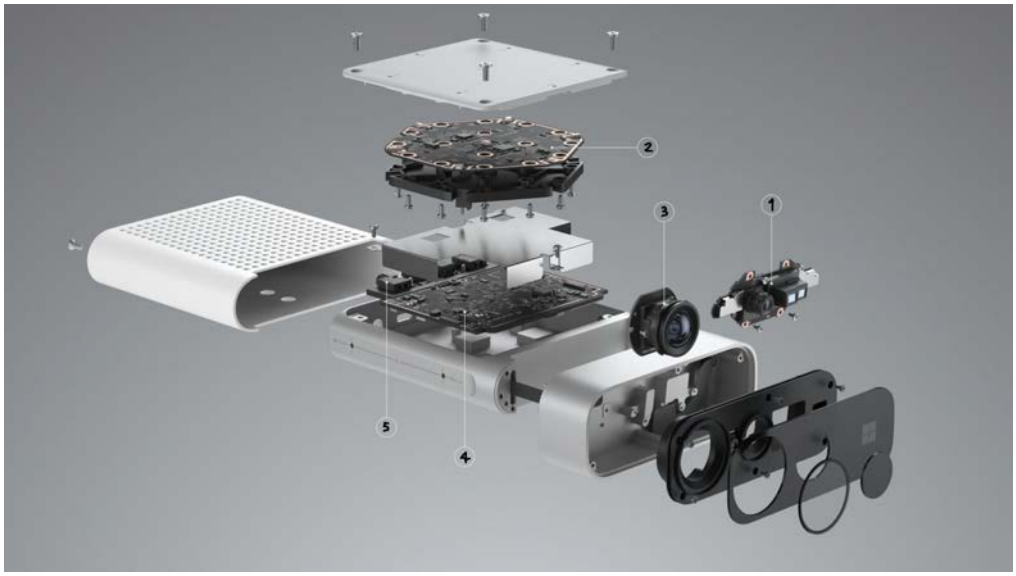



Figure 2.9: The Azure Kinect exploded.

The Azure Kinect is the newest version of kinect presented by the Microsoft Azure in this year. It incorporates the best AI sensors (see Figure 2.9): 1 MegaPixel time of flight depth camera (1), 7 microphone array (2), 12 MegaPixel RGB camera (3), Accelerometer and gyroscope, the so called *Inertial Measurement Unit (IMU)* for sensor orientation and spatial tracking (4). The Azure Kinect has an external sinc pins (5) that allows

to retrieve a synchronize sensor stream from multiple Kinect devices and it can be combined with the Azure services. The Azure Kinect is designed for industrial purposes and not for entertainment as the Microsoft Kinect v1 and v2 that were designed for the XBOX console.
The depth and RGB camera technical specification of the Azure Kinect can be found respectively in the table 2.3 and in the table 2.2.

| **Resolution (HxV)** | **Aspect Ratio** | **Format Options** | **FPS** | **Nominal FOV (HxV) post-processed** |
|---|---|---|---|---|
| 3840×2160 | 16:9 | MJPEG | 0, 5, 15, 30 | 90° × 59° |
| 2560×1440 | 16:9 | MJPEG | 0, 5, 15, 30 | 90° × 59° |
| 1920×1080 | 16:9 | MJPEG | 0, 5, 15, 30 | 90° × 59° |
| 1280 × 720 | 16:9 | MJPEG, YUY2, NV12 | 0, 5, 15, 30 | 90° × 59° |
| 4096×3072 | 4:3 | MJPEG | 0, 5, 15 | 90° × 74.3° |
| 2048×1536 | 4:3 | MJPEG | 0, 5, 15, 30 | 90° × 74.3° |

Table 2.2: The Azure Kinect RGB camera technical specifications.

| **Mode** | **Resolution** | **Field of Illumination (FOI)** | **FPS** | **Depth Range** | **Exposure Time** |
|---|---|---|---|---|---|
| Near FOV unbinned | 640 × 576 | 75° × 65° | 0, 5, 15, 30 | 0.5 - 3.86 m | 12.8 ms |
| Near FOV 2x2 binned | 320 × 288 | 75° × 65° | 0, 5, 15, 30 | 0.5 - 5.46 m | 12.8 ms |
| Wide FOV 2x2 binned | 512 × 512 | 120° × 120° | 0, 5, 15, 30 | 0.25 - 2.88 m | 12.8 ms |
| Wide FOV unbinned | 1024 × 1024 | 120° × 120° | 0, 5, 15 | 0.25 - 2.21 m | 20.3 ms |
| Passive IR | 1024 × 1024 | N/A | 0, 5, 15, 30 | N/A | 1.6 ms |

Table 2.3: The Azure Kinect depth camera technical specifications.

### Pico Zense DCAM710


Figure 2.10: The Pico Zense DCAM710 camera.

The Pico Zense DCAM710 is a RGB-D camera designed by the Pico Technology company. The depth information is retrieved by using the ToF technology combined with the *Charge Coupled Device (CCD)* sensor. The light in this case is emitted by a laser emitter with a wavelength of 850/940 nm, other technical specification can be found in the table 2.4.

| | |
|---|---|
| **Laser emitter** | 850/940 nm |
| **Depth sensing FOV** | 51° vertical, 69° horizontal |
| **Depth Range** | 0.2 m up to 5 m |
| **Frame Rate (depth and color stream)** | 30 fps |
| **Depth Stream Resolution** | 640 × 480 |
| **RGB Stream Resolution** | 1920 × 1080 |
| **Output Format** | Depth Map (RAW12), RGB (H.264) |

Table 2.4: The Pico Zense DCAM710 technical specifications.

### Intel RealSense D435


Figure 2.11: The Intel RealSense D435 camera.

The Intel RealSense D435 is an active stereo system designed by Intel. It uses an IR structured light and a stereo system in order to retrieve the depth information. Thanks to the small factor and the stereo component of the device, it is easy to be integrated into any solutions like robotics, VR or AR, drones in both indoor and outdoor environments. Principal technical specifications are grouped in the table 2.5.

| | |
|---|---|
| **Use Environment** | Indoor/Outdoor |
| **Depth Technology** | Active IR Stereo |
| **Depth FOV (HxVxD)** | 87° × 58° × 95° |
| **Depth Range** | 0.105 m up to 10 m |
| **Depth Stream Resolution & Frame rate** | Up to 1280 × 720 & up to 90 fps |
| **RGB FOV (HxVxD)** | 69.4° × 42.5° × 77° |
| **RGB Stream Resolution & Frame Rate** | 1920 × 1080 & 30fps |

Table 2.5: The Intel RealSense D435 camera technical specifications.

### MYNT EYE D1000-IR-120/Color

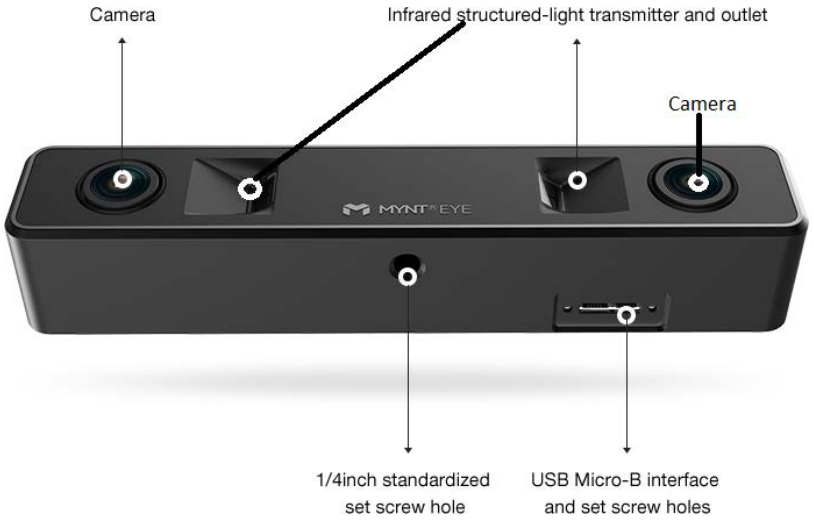



Figure 2.12: The MYNT EYE D1000-IR-120/Color camera.

The MYNT EYE D1000-IR-120/Color camera is an active stereo depth camera that uses the structured light technology and it is designed by the MYNTAI company. The camera integrated also an IMU that allows to retrieve information of the orientation and spatial position of the device. The camera can be used both in indoor and outdoor applications like service robots, drones, VR/AR, Volume Measurements, autonomous driving, industry, gesture, face and object recognition. The other technical specifications are described in the table 2.6.

| | |
|---|---|
| **Use Environment** | Indoor/Outdoor |
| **Depth Technology** | Structured light Stereo |
| **Depth FOV (HxVxD)** | $105° \times 58° \times 121°$ |
| **Depth Range** | 0.32 m up to 7 m |
| **Depth Stream Resolution** | $1280 \times 720, 640 \times 480$ |
| **RGB Stream Resolution** | Up to$2560 \times 720, 1280 \times 480$ & 30fps |
| **Frame Rate (depth and color)** | Up to 60 fps |
| **Output Data Format** | Depth map (RAW16), RGB (YUYV/MJPG) |

Table 2.6: The MYNT EYE D1000-IR-120/Color camera technical specifications.

### 2.2.2 Project Camera Selection

For the project that will be discussed in the chapter 6, the last three devices presented in the previous subsection are chosen for different setups. Since there are some compatibility problems between the Pico Zense DCAM710 cameras and the active USB extension cable, these ones have been discarded from the possible setup for the project. While the other two, Intel RealSense D435 and MYNT EYE D1000-IR-120/Color, are both suitable for the setup of the project. Since the MYNT EYE cameras have larger FOV than the Intel RealSense cameras, it is preferable to use the first one than the second one because it is possible to reduce the number of cameras used for covering the entire area of interest of the project.

## 2.3 Difference Between RGB-D Camera and RGB Camera

The RGB-D camera, in the hardware components, is essentially composed by one or two RGB sensors, a light emitter and a receiver. In the case of two RGB sensor, it is possible to create stereo systems that can be passive (only RGB cameras) or active (RGB cameras plus a light emitter for IR or laser light). In these systems, the depth information can be calculating by using the ToF technology or the structured light technology with the triangulation and epipolar geometry theory [21, 24].

Thanks to the additional depth sensor with respect to the traditional RGB camera, the RGB-D camera is more suitable for low light indoor environments. This is because the RGB camera do not have enough natural light in order to capture high quality images from the scene and the depth camera uses the active depth system, i.e. it has the light emitter, to illuminate the scene and capture good depth data for the request applications. The RGB-D camera is also more suggested for outdoor environments or applications that require the knowledge of the spatial position of the objects or the obstacles in the scene.
Unfortunately, due to the fact that the FOV of the RGB-D cameras are relatively small, it is difficult to cover the entire scene with a single camera. So there is the need of developing systems with multiple RGB-D cameras. All the necessary requirements, troubles and different configurations will be presented and discussed in the next chapter.

# Chapter 3

# RGB-D Multicamera System

The RGB-D sensor cameras, presented in the previous chapter, are all commercial low cost cameras. For this reason, its depth FOVs are much smaller than the high cost mapping specialized sensors and the quality of the depth data deteriorates with the increasing of the distance of the object from the camera [20]. So they cannot cover a wide area scene with a reasonable quality of the depth information. In order to overcome this problem, a system with multiple RGB-D cameras is introduced. With a multicamera system, it is possible to assign a specific area of the entire scene to one RGB-D camera, then combine the information retrieved from all the cameras in order to reconstruct the interested wide area [31].
In order to create an useful multicamera system, some important requirements have to be met [32]:

- The *interference* between the cameras should be as minimum as possible in order to guarantee an high accuracy of the depth measurement.
- The *total area coverage* should be as big as possible with a minimum number of cameras.
- The *setup* should be modular in order to be scalable for other installations.
- The *workload* of the image processing should be well distributed over multiple computers if there is the possibility to install more than one computer. Otherwise the single computer must have more than one USB-controller installed since a single camera uses the whole bandwidth of a USB-controller.

After all these requirements are met and depending on the dimensions and the shape of the wide area and the FOVs of the RGB-D cameras, the latters can be positioned in different configurations:

- Configured as an array of cameras.
- Configured as a matrix of cameras.
- Configured as a circle of cameras.

## 3.1 Requirements for a Multicamera System

The requirements described above are essential for the implementation of a good multicamera system but there exist also other requirements such as the use of USB extension cables that are related to the physical position and distance between the RGB-D cameras and the computers in the system setup and the need of an intrinsic and an extrinsic calibration of the cameras [32, 33]. In the following subsections, these requirements will be presented in detail.

### 3.1.1 Interference

For structural light sensors such as the RGB-D cameras presented in the previous chapter, its depth signal degrades drastically when multiple cameras are pointing to the same scene or to an overlapping area [31, 32, 34]. This signal degrading is principally caused by the projection of multiple structured light dot patterns of the multiple cameras onto the scene in a continuous way and without modulation. When a dot pattern of a device interferes with the others, it is called *crosstalk* [34] and it can cause a drastically drop of the accuracy of the depth measurement. For solving the interference problem, Butler et al. [34] presented the *Shake'n'Sense* method (see Figure 3.1). This method consists in a minimally vibration of the RGB-D camera using an offset-weight vibration motor in order to introduce an artificial motion blur. Since both the structured light diffractive optical element illuminator and the IR camera of the RGB-D camera are moving in harmony, its depth sensing works normally. However, this minimal motion causes a blurring of the structured light pattern from the other cameras and it can be used to eliminate the most of the crosstalk.

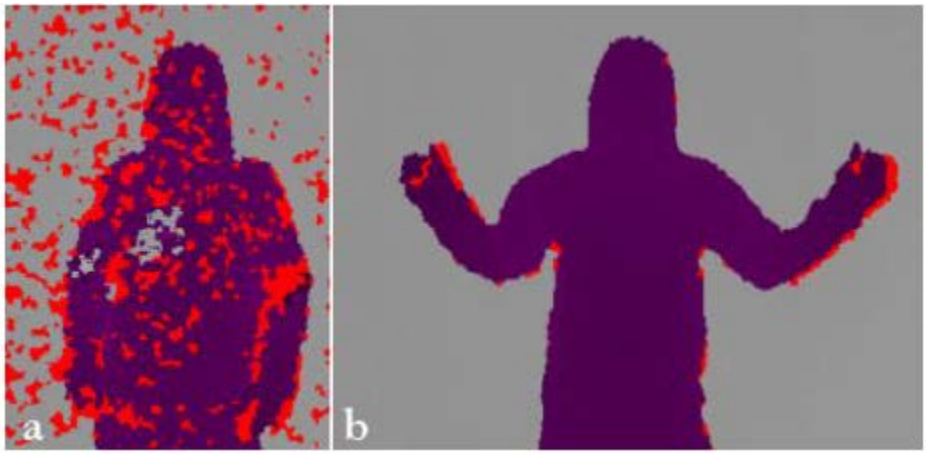



Figure 3.1: The depth noise caused by multiple overlapping Kinect patterns (left), the depth map obtained by using the Shake'n'Sense method (right).

### 3.1.2 Total Area Coverage

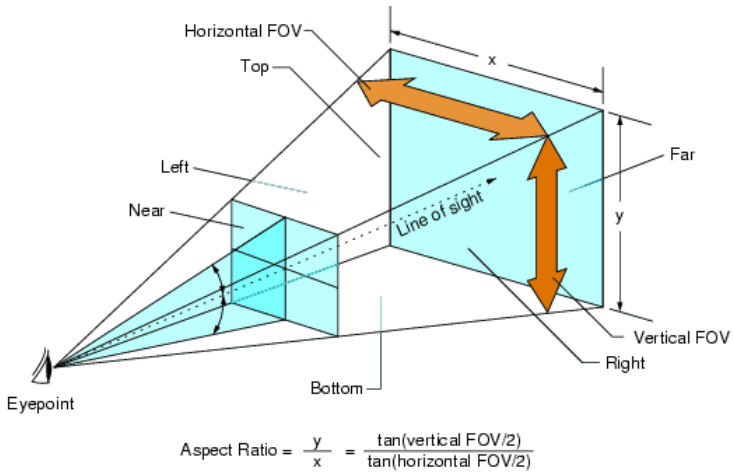



Figure 3.2: Using the illustrated formula, it is possible to determine the dimensions of the interested area.

When a multicamera system is implemented, it is a good practice to make the setup with possibly the minimum number of cameras to cover a maximum possible area. The number of used cameras depends on the dimension of the wide area to be covered, the FOVs of the RGB-D cameras and its location in the scene. Since the depth resolution deteriorates with the distance of the scene from the camera [20], the cameras need to be installed

at a right distance in order to guarantee a good depth resolution. By fixing an opportune distance, it is possible to determine the dimensions of the interested area covered by the camera (see Figure 3.2). Then, depending on the application, the cameras can be positioned in different configurations and in an overlapping or a non overlapping way in order to cover the entire scene. The first introduce the interference since in the overlapping areas there are more than one structured light pattern [34], while the second try to avoid the interference by positioning the cameras in the way that the overlapping area is very small and the interference is negligible [31]. The different topology of the cameras will be presented and discussed later in a dedicated section.

### 3.1.3 Setup Modularity

During the implementation of the multicamera system, it is better to make the setup as much modular as possible in order to allow a future extension of the current setup. If the setup is composed by modules and there is the need to cover more space in the scene, it is sufficient to add more than one of these modules including the additional computers if needed. An example is shown in Figure 3.3.

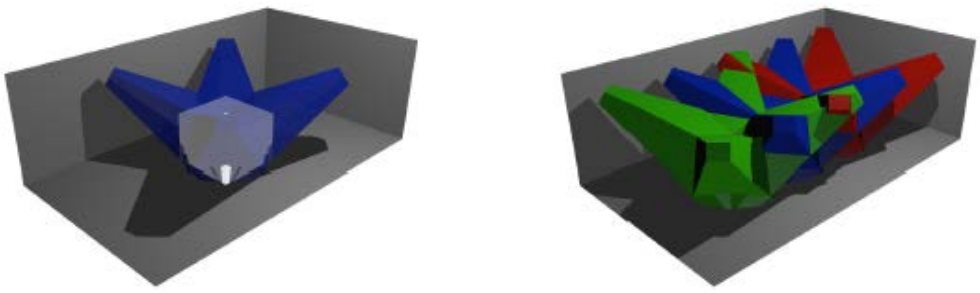


Figure 3.3: Example of setup with one module (left), example of extension of the setup with three modules (right).

### 3.1.4 Workload and USB Extension Cable

The number of RGB-D cameras that can be connected to a single computer is directly proportional to the number of the USB controller installed in it [32]. This is because the datastream of a RGB-D camera uses all the bandwidth available on a USB controller. So, if there is only one USB controller in the computer, it is possible to connect only one RGB-D camera on it.

The total workload of the imaging processing can be equally distributed on the multiple computers if there is only one USB controller per computer. Otherwise the single computer must have a number of USB controller greater than or equal to the number of used cameras. This single computer must have an adequate hardware in order to process the incoming data without visible delay.
Since in most of the setups it is impossible to install the computers near to the RGB-D cameras that are spreaded in the scene and the USB cables of the cameras have more or less one meter long, the USB extension cables are needed to connect the cameras to the computers. Depending on the topology of the cameras and the location where the computers are installed, the length of the USB extension cable can vary from 10m to 20m. The power of the transmitted signal decreases with the increasing of the length of the cable. This is due to the attenuation induced by the cable, so for these lengths, it is suggested to use the active USB extension cable instead the normal one in order to amplify the transmitted signal.

### 3.1.5 RGB-D Multicamera Calibration

Since it is a multicamera system, there is the need to do two different type of calibration: intrinsic calibration and extrinsic calibration. The latter is subdivided again into extrinsic calibration on every RGB-D camera and extrinsic calibration between all the RGB-D cameras.

#### Intrinsic calibration of the RGB-D camera

Intrinsic calibration is a method used in order to estimate the focal length, optical center and camera distortion, radial and tangential, coefficients of the sensors that compose the RGB-D camera (i.e. RGB and IR camera) [32, 33]. The calibration procedure consists in holding a checkerboard with different positions in front of the camera, then using the Harris-Stephen corner detector and the Zhang's algorithm in order to detect the corners of the checkerboard and calculate the intrinsic parameters. This procedure is done multiple times, one for each sensor of the RGB-D camera.

#### Extrinsic calibration of each RGB-D camera on its own

Since the RGB-D camera is composed by two sensors, RGB and IR sensor, it is necessary to calibrate the relative position of the two sensors in order to overlay the captured color image on the depth one (see Figure 3.4).

Figure 3.4: Example of color image overlays the depth image after calibration.

### Extrinsic calibration between the RGB-D cameras of the setup

Since the setup has more than one RGB-D camera, it is necessary to calibrate all the cameras between them in order to create one single wide image or one single pointcloud with a common coordinate system. Unfortunately the calibration procedure can calibrate only two devices at time [31, 32], so it is necessary to fix one camera as the reference camera and calibrate all the others with respect to the reference one. In this way the calibration procedure need to be repeated $n-1$ times with $n$ the number of RGB-D cameras used for the setup. Essentially the calibration process consists in the rotation and translation of the coordinate system of the camera with respect to the reference one by using the rotation and the translation matrix (see Figure 3.5).

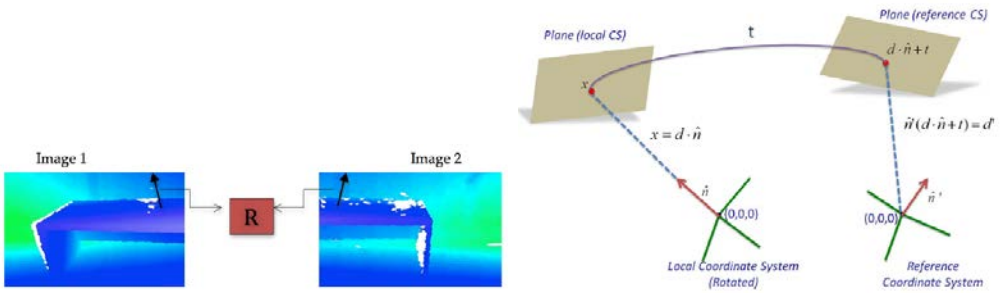



Figure 3.5: Rotation (left), Translation (right) of a coordinate system with respect to the reference one.

## 3.2 RGB-D Multicamera Topology

In a multicamera system, it is better to install the cameras in order to obtain the coverage of the entire scene with the minimum number of cameras. Dpending on the dimensions of the scene and the kind of retrieved data, it is possible to have different topology configurations: array, matrix or circle of cameras.

### 3.2.1 Array of RGB-D cameras

The RGB-D cameras are installed inline on the same direction of one of the dimension of the scene with all FOVs pointed inline on the same direction (see Figure 3.6). In this way the cameras are aligned along one axis. This eases the combination of the images obtained from the different cameras into a single bigger image. For maximizing the coverage area, the cameras are installed with a distance between them in order to avoid the overlapping and consequently reduce the interference [31].

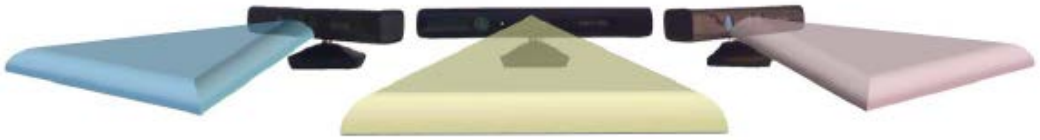


Figure 3.6: Example of an array of RGB-D cameras.

### 3.2.2 Matrix of RGB-D cameras

When using an array of cameras is not sufficient to cover the scene in both the dimensions, a matrix of cameras is needed in order to cover the entire scene. Depending to the distance of the scene from the cameras, it is possible to have both overlapping (see Figure 3.7) and non overlapping configuration. The second one is more suggested for the same reasons explained for the array topology.

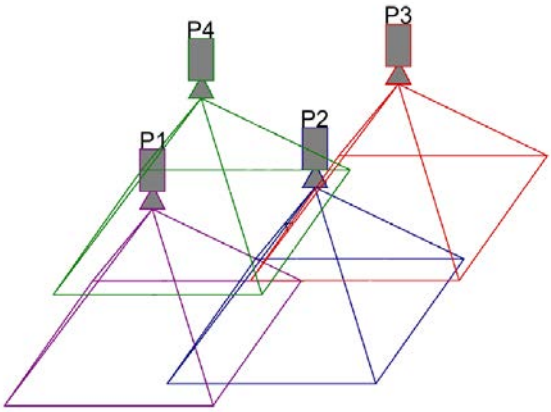



Figure 3.7: Example of an overlapping matrix of RGB-D cameras.

### 3.2.3 Circle of RGB-D cameras

This configuration is usually used to retrieve a pointcloud of the scene from the cameras. Here the cameras are installed in a circle way in order to cover all the directions of the scene (see Figure 3.3 (left)). In this way, it is possible to drastically reduce the occlusion of some part of the scene. Occlusions that are present if the cameras are all pointing in only one direction. With this configuration, the resulting pointcloud is more detailed than the pointcloud captured from only one camera (see Figure 3.8).

Figure 3.8: Example of a circle of RGB-D cameras.

# Chapter 4

# Neural Networks

An *Artificial Neural Network (ANN)* is a model of computation inspired by the structure of neural network present in the human brain. The first concept of this type of network was proposed in the mid of the 20th century [35, 36, 37, 38] and the first practical applications were proposed in the 80-90s. Unfortunately the performances of these first applications are lower than the *Support Vector Machine (SVM)* and other techniques. In the recent decades, with the improvement of the computational power of the computers, the neural network applications achieved an impressive improvement in the performances allowing in this way to use them for a lot of machine learning algorithms [35, 39]. In the following sections, the main concepts, how they work, different type of neural networks and their applications in image processing will be presented.

## 4.1 Artificial Neural Network Model

Since the ANN is a simplified model of the human brain, it is composed by a large number of basic computation devices called neurons that are connected to each other with a sophisticated communication network (see Figure 4.1). The neural network can be represented as a directed graph that goes from the input neuron to the output neurons, respectively the input and output layer depicted in the figure 4.1. In this graph the nodes represent the neurons and the edges are the links between the neurons.

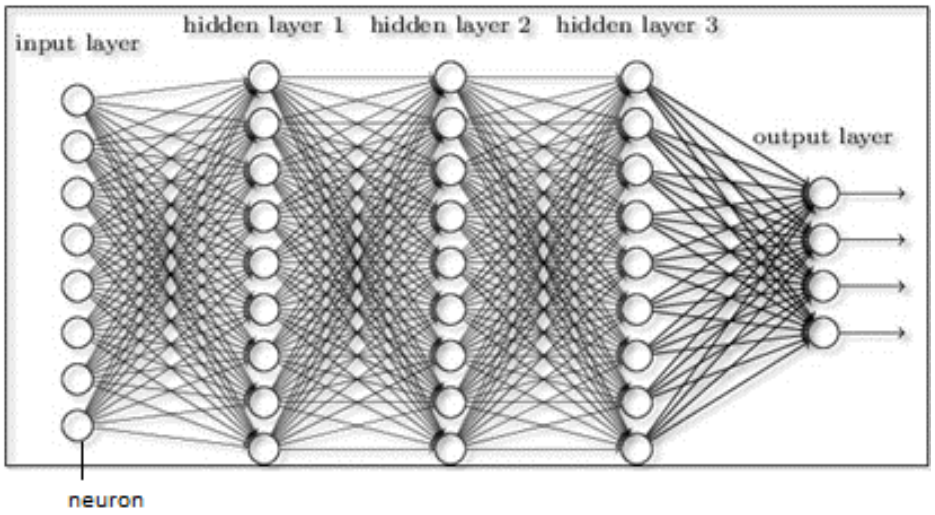



Figure 4.1: A neural network graph representation.

As we can see in the figure above, in addition to the input and output layer, in the graph there are other layers called hidden layers. These layers, as their names suggests, are hidden to the users of the neural network algorithms. The complexity of the neural network is proportional to the number of hidden layers present in it. More hidden layers there are, more specific features the network can learn but also the computational complexity increases. So there is a trade-off between the number of hidden layer and the computational complexity of the neural network when we design it.

### 4.1.1 Basic Computing Devices: Neurons

The neuron, sometimes called also perceptron, is the basic computing element of a neural network. It can be basically decomposed into two part [36, 37]:

- Linear weighted sum operator function $z$.
- Non linear activation function $\sigma$.

A representation of a neuron is shown in the Figure 4.2.

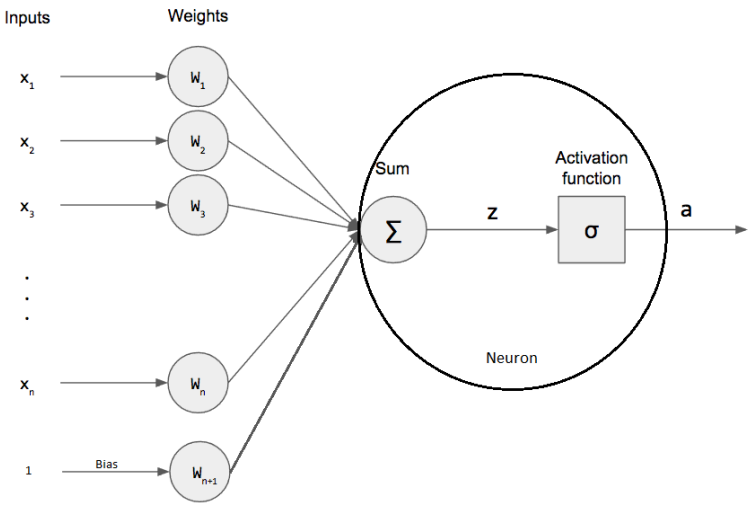



Figure 4.2: The representation of a neuron.

### Weighted Sum Function

In this part the function takes as input all the outputs of the previous layer weighted by their own weights and gives as result their sum with also a bias element. Mathematically speaking, the formula is

$$z = \sum_{k=1}^{n} w_k x_k + w_{n+1} \tag{4.1}$$

where $w_k$ is the weight of the $k$-th input, $x_k$ is the $k$-th input, $n$ the number of the input and $w_{n+1}$ is the bias.

### Activation Function

In this part the function takes as input the result of the previous part and through the activation function, it gives the output of the neuron.
There are many activation functions that we can choose to use [36, 37].

### Sign and Threshold Function

The sign function gives as output the sign of the input (see Figure 4.3).

$$\sigma(z) = sign(z), \qquad \mathbb{R}^n \to [-1; 1] \tag{4.2}$$

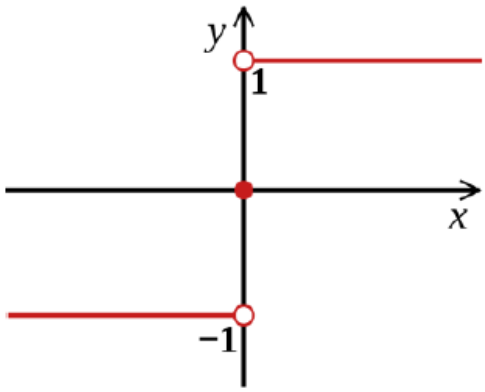



Figure 4.3: The sign activation function.

The unit step threshold function gives as output

$$\sigma(z) = \begin{cases} 0 & \text{if} \quad z < t \\ 1 & \text{if} \quad z \geq t \end{cases} \tag{4.3}$$

where $t$ is the threshold (see Figure 4.4).

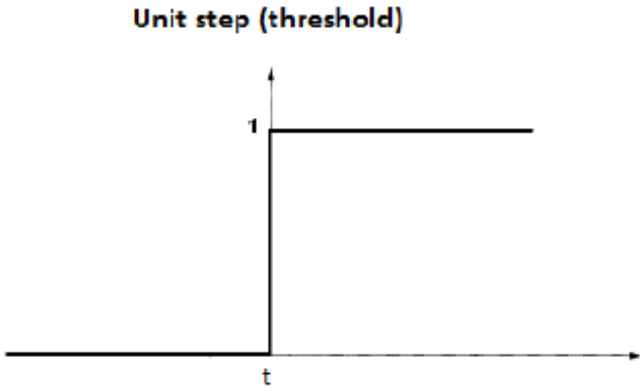



Figure 4.4: The unit step threshold activation function.

These two functions have similar characteristics and consequently same pros and cons.

**Pros**

- They are simple/fast functions.
- They are a nice interpretation as the firing rate of the neuron
  - -1 (sign) or 0 (threshold) = not firing
  - 1 (both) = firing

**Cons**

- The output is not smooth or continuous.
- They saturate and kill gradients, thus the neural network will barely learn.

### Sigmoid Function

This function takes a real valued number and maps it into a range between 0 and 1 (see Figure 4.5).

$$\sigma(z) = \frac{1}{1+e^{-z}}, \qquad \mathbb{R}^n \to [0,1] \tag{4.4}$$

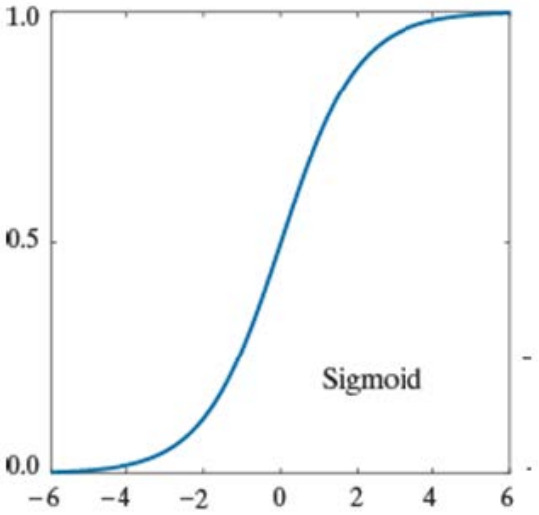



Figure 4.5: The sigmoid activation function.

**Pros**

- The output is smooth.
- It is a nice interpretation as the firing rate of a neuron
  - 0 = not firing at all
  - 1 = fully firing

**Cons**

- Sigmoid neurons saturate and kill gradients, thus neural network will barely learn.
- When the neuron's activation is 0 or 1 (saturate)
  - gradient at these regions is almost zero
  - almost no signal will flow to its weights
  - if initial weights are too large, then most neurons would saturate

**Hyperbolic Tangent Function**

This function takes a real valued number and maps it into a range between -1 and 1 (see Figure 4.6).

$$\sigma(z) = \tanh(z), \qquad \mathbb{R}^n \to [0, 1] \tag{4.5}$$

**Characteristics**

- Similar to the sigmoid function, the hyperbolic tangent neurons saturate.
- The output of this function is zero-centered.
- The hyperbolic tangent is substantially a scale sigmoid as shown in the following equation.

$$\tanh(z) = 2sigm(2z) - 1 \tag{4.6}$$

where the $sigm(\cdot)$ is the sigmoid function described above.

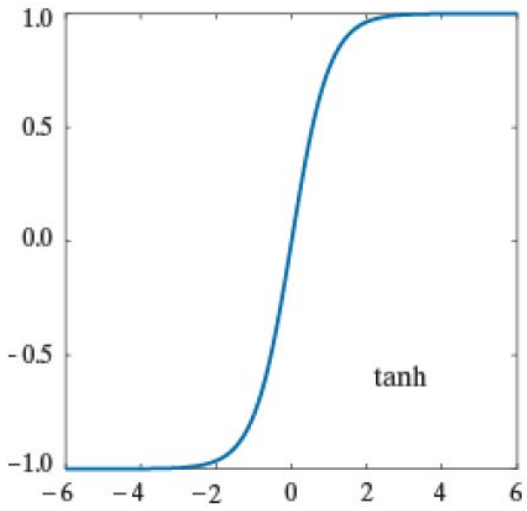



Figure 4.6: The hyperbolic tangent activation function.

### Rectified Linear Unit (ReLU) Function

This function takes a real valued number and gives as output the maximum between zero and the input (see Figure 4.7).

$$\sigma(z) = \max(0, z), \qquad \mathbb{R}^n \to \mathbb{R}^n_+ \tag{4.7}$$

### Characteristics

- It is the most used activation function in the neural networks nowadays.
- It trains much faster
  - accelerates the convergence of the *Stochastic Gradient Descent (SGD)*
  - due to linearity, there is no saturation
- It has less expensive operations
  - compared to the sigmoid or tanh functions
  - implemented by simply thresholding a matrix at zero
- It is more expressive and prevents the gradient vanishing problem.

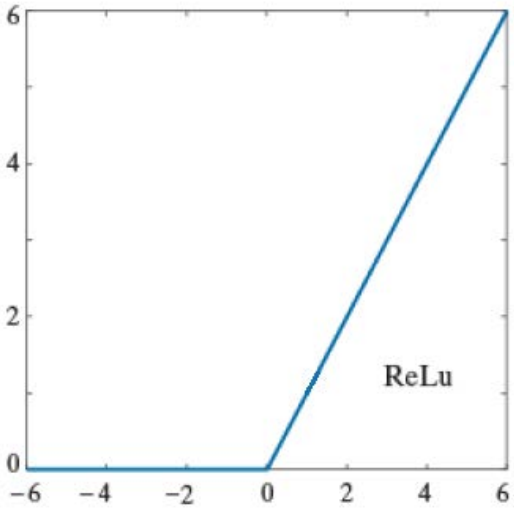



Figure 4.7: The ReLU activation function.

## 4.2 Fully Connected Neural Networks

The network depicted in the figure 4.8 is called fully connected neural network. In this network, each neuron of a layer takes as input all the outputs of the neurons of the previous layer and its output is the input for every neuron of the next layer. This is true for all layers with exception of the first layer where there is no previous layer and the last layer where the outputs are the outputs of the neural network. The number of neurons for each hidden layer can be different from layer to layer. More hidden layers there are, more deep becomes the neural network and consequently the number of connections and computational complexity increase. For this reason it is also called *Deep Neural Network (DNN)* [39]. When the represented graph of the network has no cycles, it is called *Feedforward network* [36, 37, 39]. This type of neural network is typically used for classification purposes. In fact, the number of the output neurons can represent the number of classes of the problem and the maximum value between all the outputs is the selected class for a given input data.

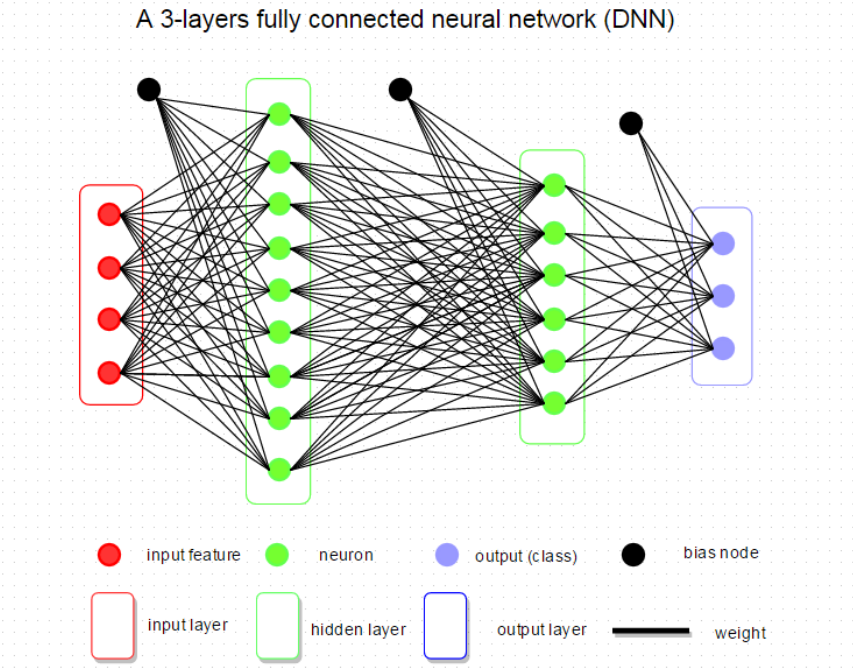



Figure 4.8: The representation of a DNN.

### 4.2.1 Training Process

In order to predict the given input data, a neural network need to be trained with a set of labeled data (i.e. supervised learning) that is forwarded through the network and predicted the corresponding label at the output of the neural network. Then, this predicted label is compared with the ground truth label and the prediction errors are calculated. The process continues with the back propagation of these errors from the last layer to the first one and updates the network weights (see Figure 4.9). This training process loops these steps until a small training error or a small marginal improvement in the error or an upper bound on the number of iterations is reached. The pseudocode of the training algorithm is shown in the figure 4.10.

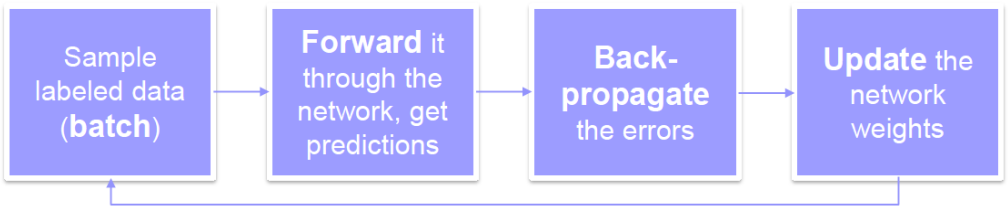



Figure 4.9: The training process of a DNN.

This is the final backpropagation algorithm, based on SGD, to train a NN

**Input:** training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$, NN (no weights $w_{ij}^{(t)}$)

**Output:** NN with weights $w_{ij}^{(t)}$

initialize $w_{ij}^{(t)}$ for all $i, j, t$;

**for** $S \leftarrow 0, 1, 2, \ldots$ **do** /* until convergence */

  pick $(\mathbf{x}_k, y_k)$ at random from training data;

  compute $v_{t,j}$ for all $j, t$; /* forward propagation */

  compute $\delta_j^{(t)}$ for all $j, t$; /* backward propagation */

  $w_{ij}^{(t)} \leftarrow w_{ij}^{(t)} - \eta v_i^{(t-1)} \delta_j^{(t)}$ for all $i, j, t$; /* update weights */

  **if** *converged* **then return** $w_{ij}^{(t)}$ for all $i, j, t$;

Figure 4.10: The pseudocode of the training algorithm of a DNN.

## 4.3 Deep Convolutional Neural Network

The standard neural networks have as input a vector of data, so for image processing, it is necessary to do a pre-processing of the images features before using them as the input of the neural networks. One approach is to vectorize the input image directly by orginizing the pixels based on a linear index. The resulting vector is given as the input of the neural network. Howerver, this approach totally ignores all the information that can be retrieved from the spatial relationship between the pixels in an image [37]. In order to overcome this problem and use all the possible information derived from the neighborhood pixels, the *Convolutional Neural Networks (CNNs)* are proposed (see Figure 4.11) [37, 40, 38, 39]. This type of neural networks are specifically designed to mimic the human visual cortex [39] and to elaborate 2 dimensional array data with a grid like topology as its input data like images and videos [35, 38, 41]. The main difference between a DNN and a CNN is that the CNN can learn 2D features directly from the raw image data since the input format is a 2D array against the 1D vector of the DNN. Despite this difference, the computations performed by the two type of neural networks are very similar: a sum of products is performed, then the bias value is added; the result is passed through an activation function and it becomes a single input for the next layer.

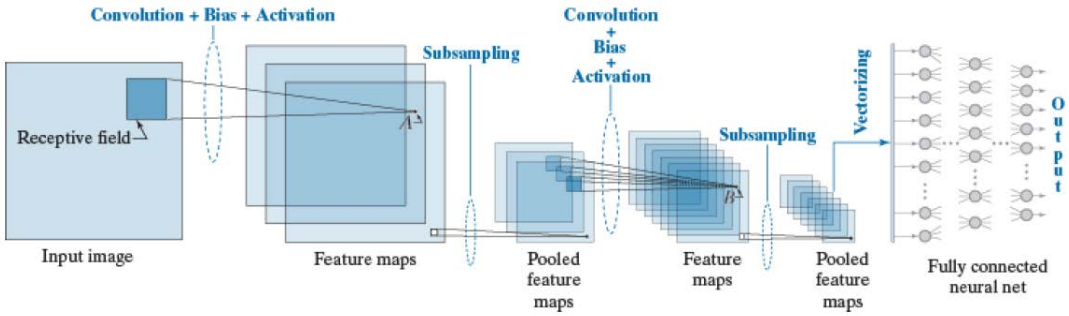



Figure 4.11: The representation of a CNN.

### 4.3.1 How a CNN works

The advantages of using the CNNs are that only the neighborhood nodes are connected, so only a set of weights and a bias value are used and shared between the nodes. This is called *weight sharing* [37, 40]. Using shared weights, it is possible to reduce the number of weights to train and the computational complexity of the network decreases.

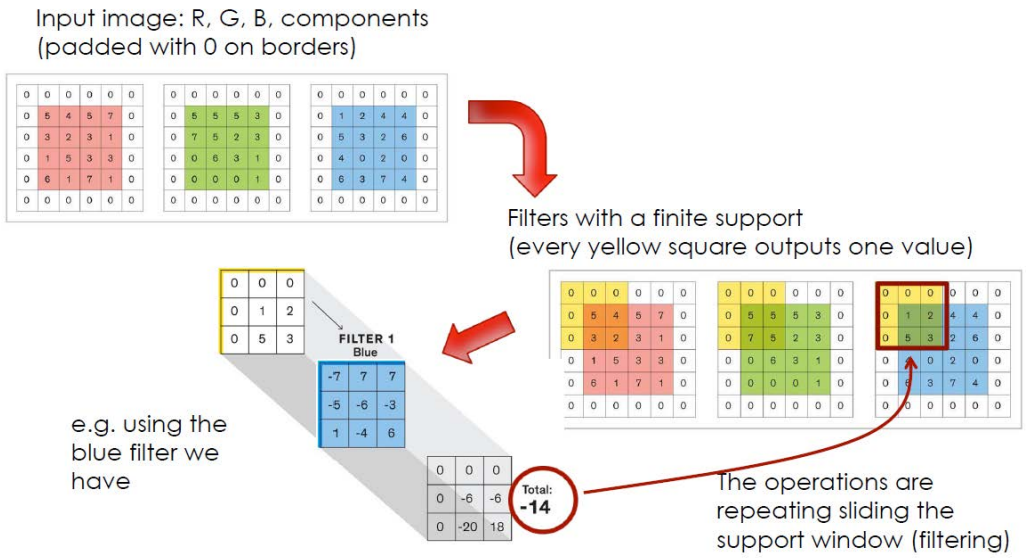



Figure 4.12: An example of the convolution.

In the CNNs terminology, the neighborhoods are called *receptive fields* (see Figure 4.11) and it only selects a region of pixels in the input image. The set of weights arranged in the receptive fields is called *kernel*. The first operation of the CNN is the convolution. It is performed by moving the

receptive field over the image and at each location, the result is given by the convolution between the location and the kernel (see Figure 4.12). The number of spatial increments for the movement of the kernel is called stride. This number can be one or also grater than one. Using a bigger stride is essentially for the data reduction and substitution of the subsampling process. If $n$ is the stride, the image resolution is reduced by $1/n$ times in each spatial dimension and the total amount of data per image is reduced by $(n^2-1)/n^2$ times. After the convolution, a bias value is added and the activation function is performed (see Figure 4.13). The output of the activation function

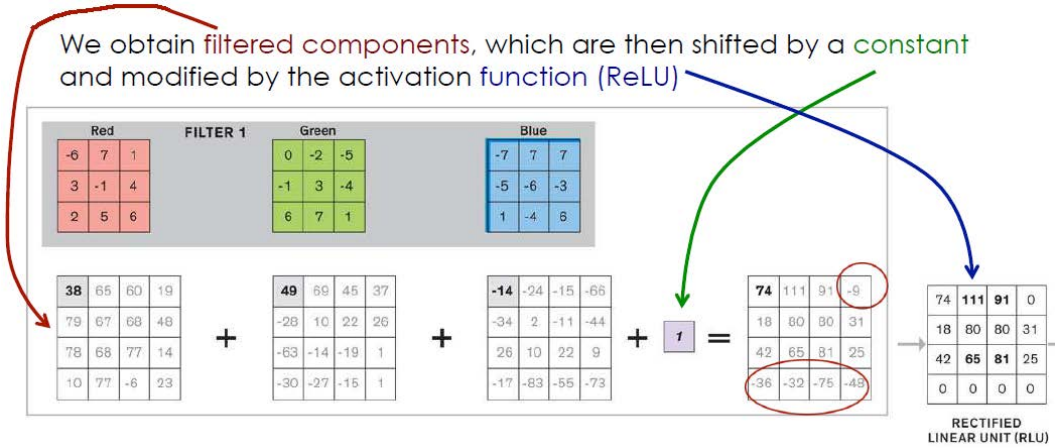



Figure 4.13: An example of the adding bias and activation function.

is stored in the 2D array of the next layer and it is called feature map since the role of the convolution is to extract features from the input image. Since usually the kernel size is small, it is possible to extract multiple feature maps by using multiple kernels and the collection of the feature maps of a layer is called *convolutional layer*.
After the convolution and activation functions, the feature maps are subsampled (i.e. pooled) in order to reduce the spatial resolution and to achieve the translational invariance. Pooling process is done by subdivided the feature map into small regions, typically a $2 \times 2$ matrix, and replacing the value of the considered region with a single value. This value can be computed by considering the maximum value of the values in the region (i.e. max pooling) or by considering the average value of the values in the region (i.e. average pooling). Usually the pooling regions are considered without overlapping (see Figure 4.14).

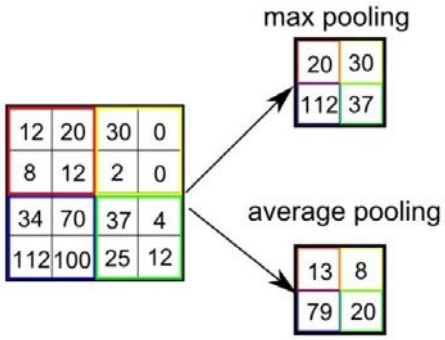



Figure 4.14: An example of the two types of pooling.

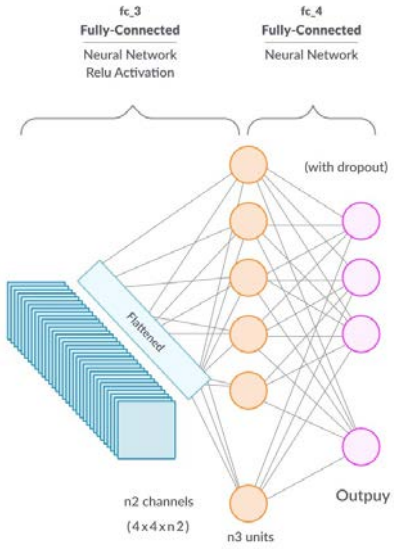



Figure 4.15: An example of the vectorizing of the last pooling layer and the fully connected layers.

All the processes describe above are repeated more times based on the number of convolutional layer of the considered CNN. Since the final objective of the CNN is to use the extracted features for classification purposes, the last part of the CNN is composed by a classic fully connected neural network. Since the latter accepts only vector inputs, the results of the last pooling layer are vectorized by using a linear indexing process (see Figure 4.15). At the end of the fully connected layers, for each class, a probability that the input image belongs to it is given. The final predicted class is the class with the biggest probability.

## 4.4 Typologies of Image Data Input for CNNs

As explained in the previous section, the CNNs are especially suitable for processing matrix like data such as images, videos and camera streams [35, 38, 41]. Since a video or a camera stream is composed by a finite (video) or infinite (camera stream) sequence of images called frames, it is possible to focus only on the analysis of different data provided by the cameras. Then make some considerations for the delay introduced by the image analysis in the real-time applications with the camera streams. The retrieved data depends on the typology of the used camera and can be only RGB data for the RGB cameras and RGB plus depth data for the RGB-D cameras. Since the latters are of interest for this thesis, the focus is on them and on the different type of data that the RGB-D cameras can retrieve through the different configurations described in the previous chapter:

- Only RGB images.
- Only Depth images.
- RGB and depth images together.
- Point cloud.

### 4.4.1 RGB Image Data

The RGB images are composed of Red, Green and Blue channels. For this reason when the image is the input of a CNN, it is necessary to do the convolution over all the three channels and then combine the results before the bias and the activation function (see Figure 4.13).
At the beginning the computation power is based only on the CPUs. So training a huge CNN with huge amount of data can require also months to finish the process [42]. For this reason, the first image datasets are only on the order of tens of thousands images such as NORB, CIFAR-10/100 [41, 43]. In the recent years, with the introduction of the GPU computing systems, the computational power increases a lot especially for image processing because of the particular ability of the GPUs to perform linear matrix operations. Thanks to this property, using the GPUs for training the CNNs can reduce drastically the computation time and consequently the training process becomes much faster [41, 42, 43]. So with the decreasing of the time of the training process, it is possible to create datasets with a huge amount of images. The dataset can be of order of hundreds of thousands images like the LabelMe dataset or of order of millions of images like the ImageNet dataset

[43]. The RGB images are usually used when the scene is well illuminated because in the low light environments the RGB cameras cannot capture well defined and low noise images, so it is difficult to recognize object in the scene or segment the captured images.

### 4.4.2 Depth Image Data

With the technologies described in the section 2.1, it is possible to obtain a depth representation of the scene. In this representation a lower luminance value in the depth map means that the objects are farther away from the depth camera (see Figure 4.16).

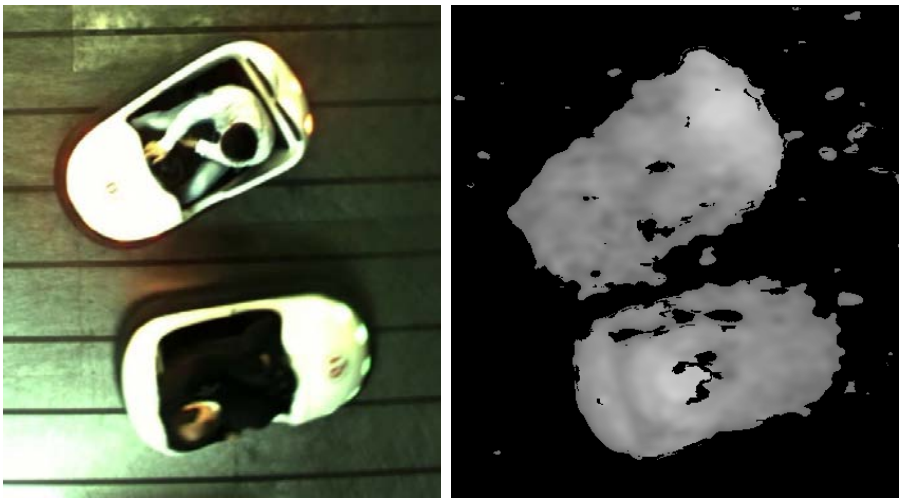


Figure 4.16: RGB image of two bumper cars (left), the corresponding depth image (right).

Since the depth image is a distance map of the scene, it is sufficient to store the depth information in a single channel instead in three channels as the RGB images. For this reason in the convolution process of the CNN, it is sufficient to do the convolution on a single channel. Usually the depth images are used in low light environments where it is difficult to extract useful information from RGB images or in environments where the distance of the objects in the scene from the depth camera is an important field.

### 4.4.3 RGB Plus Depth Image Data

Using only RGB images, it is not possible to obtain and use information about the spatial position, geometry and shape cues of the object in the scene with respect to the camera. Using only depth images, instead, it is not possible to obtain and use the color and texture information contained in the images. Since the RGB-D cameras can capture both RGB and depth images, it is possible to combine and use the information contained in both type of images [1, 3, 5]. There are two different methods for combining the information: one is to extract the features separately using two CNN and combine the results through concatenation layers in order to obtain a single result (see Figure 4.17) [1, 3, 5, 8, 44]. The other one is to combine the RGB and depth images before giving it as input of the CNN and then extract the features from the resulting images (see Figure 4.18) [1, 4].

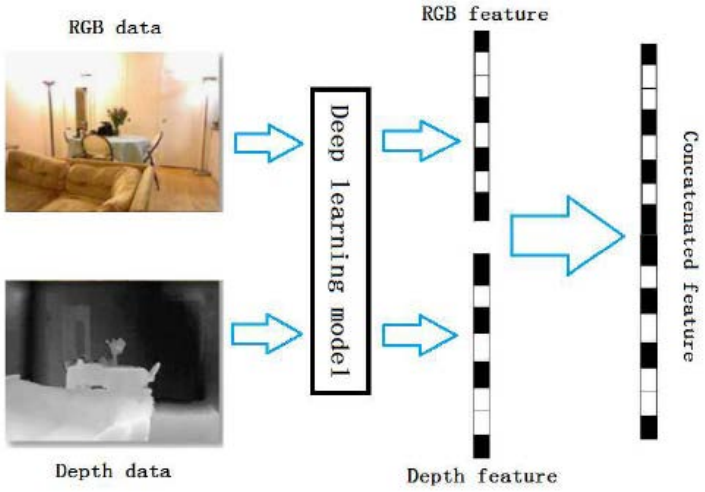



Figure 4.17: RGB-D features concatenation.

In the second method, the depth map can be treated as the fourth channel of the input image as shown in figure 4.18 and train the CNN on four channels [45]. Or the RGB image can be overlayed on the depth image and creating an image with the depth information distributed on the R, G and B three channels as shown in the figure 3.4.

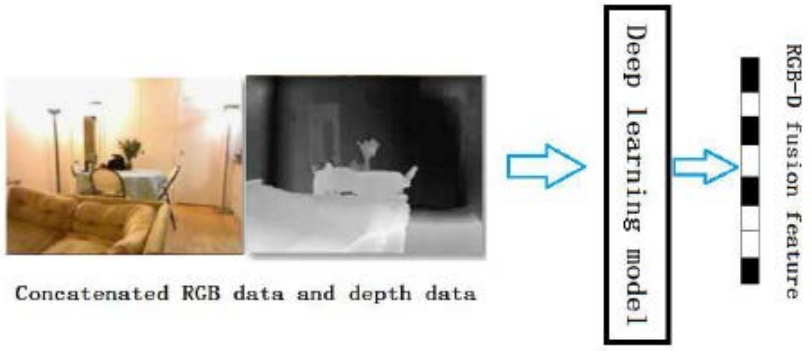



Figure 4.18: Deep features from RGB-D fusion.

### 4.4.4 Point Cloud Data

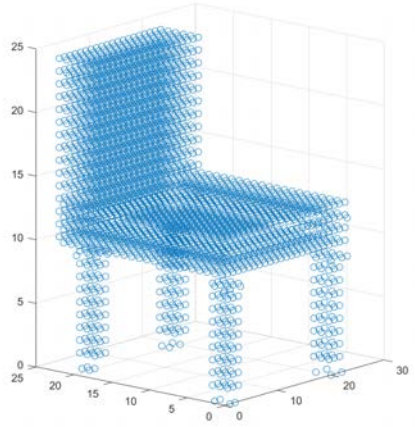


Figure 4.19: A voxel representation of a chair.

A point cloud is a 3D representation of the scene captured by the RGB-D cameras, so each point of the scene is a 3D point that contains its spatial position and color information. Most of the existing 3D object detection methods encode the point cloud into a voxel (i.e. volumetric pixel) representation before giving it to the CNN (see Figure 4.19) [46, 47, 48].

Since the voxels are 3D points, also the dimensions of the convolution kernel need to be changed from 2D to 3D by creating 3D convolutional layer with 3D sliding window and 3D pooling window (see Figure 4.20) [47, 48, 49].

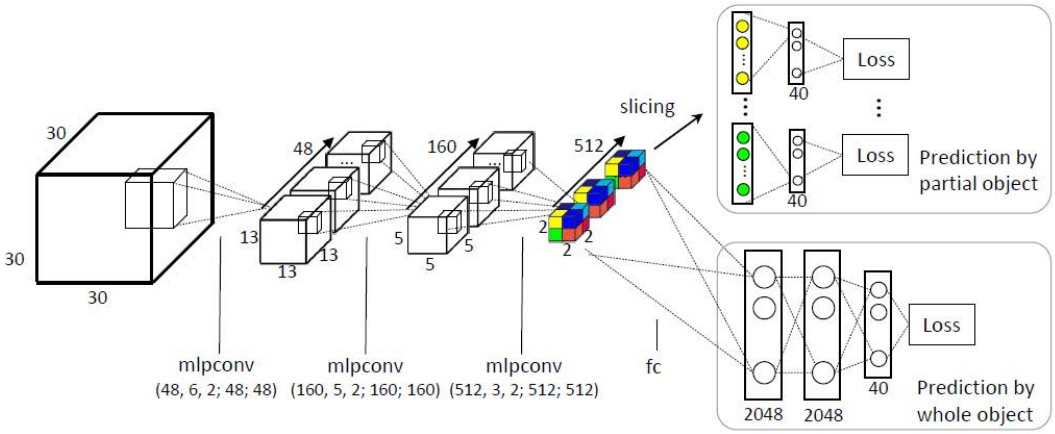



Figure 4.20: A CNN designed with 3D convolutional layers.

### 4.4.5 Video and Camera Streams

When processing video or camera streams, the most important thing is the delay introduced by the processing of each frame especially in the real-time applications where a big delay can compromise the proper functioning of the applications. For this reason, it is necessary to design a CNN with a computational delay as minimum as possible in order to maintain the frame rate of the video or camera stream at the output as closest as possible to the input frame rate.

# Chapter 5

# Object Detection and Tracking

Object detection and tracking is one of the most important and critical research area of real-time computer applications such as video surveillance, robot vision and traffic monitoring [11] because of the change in motion of the object, occlusion of the object in the interested scene, object overlapping, appearance and illumination variations and ground clutter [11, 12].
The main concepts of visual object tracking is to detect the features or shapes, the locations of the moving objects in a successive video sequences. In the recent years, much progress has been made in this research area but it is still far from reaching the accuracy of the human tracking ability [12].
An object tracking system is composed principally by three steps (see Figure 5.1).

1. Detection of the object in the scene.
2. Recognition of the detected object.
3. Tracking of the detected object through the successive video sequences.

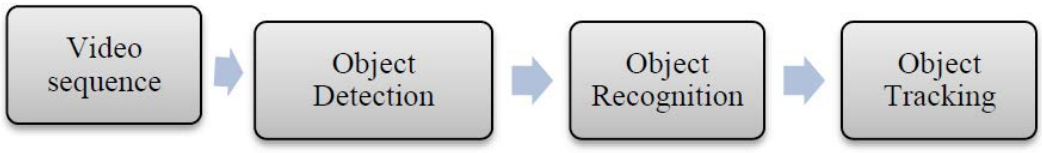



Figure 5.1: A basic flow of a tracking system.

Thanks to the great successes obtained by the CNNs in several computer vision tasks, they are extremely suggested for the implementation of the object detection and recognition steps of the tracking system while for the

tracking step, the CNNs are not used because of the huge amount of data that the CNN requires for the training process. Data that is not yet available in huge quantity for visual tracking [12].

## 5.1 Object Detection Through CNN

The main paradigm of the object detection implemented with CNNs is to train object detectors that operate on sub-images and apply these detectors in an exaustive way across all the locations and scales of the entire image [50]. The window described by the sub-image outline is called sliding window and it can have different aspect ratio and sizes based on the different scales of the target objects (see Figure 5.2) [51].

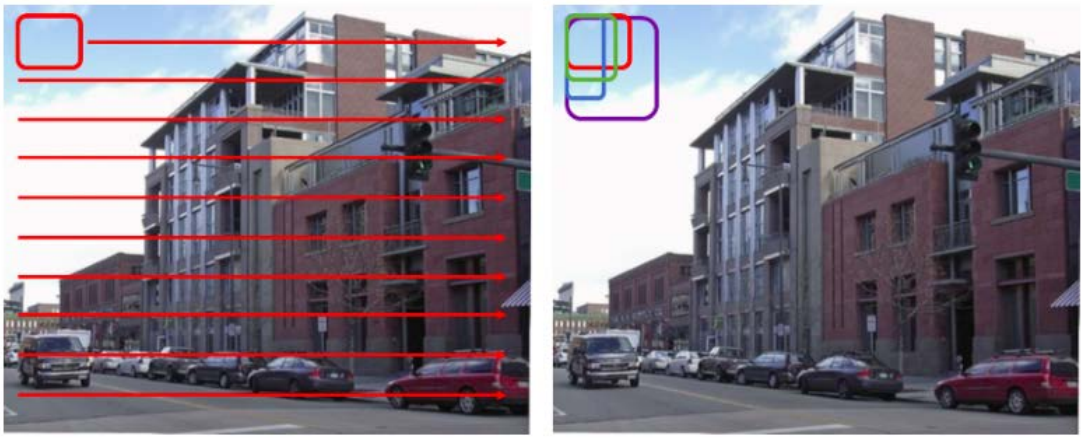


Figure 5.2: An example of sliding window (left) with different aspect ratio and sizes (right).

Since this paradigm is based on the exhaustive search through the entire image, the computational power demanding of the detection process increases a lot especially when the number of object classes grows. This is because many approaches, nowadays, are based on training separately multiple detectors for the multiple classes of the problem [50]. In order to overcome this issue, the training algorithms based on the GPUs are proposed since the latters are more powerful and faster with respect to the CPUs in this type of computations [41, 42, 43].
Using the sliding window paradigm, when the detector detects a target object, it returns some characteristics of a bounding box that encloses the detected object. These characteristics can be the coordinates in the entire image of the top-left corner or the center of the bounding box and its width

and height. With these information, it is possible to draw on the image the corresponding bounding box. If there are more instances of the same object category, the corresponding detector returns an array of the characteristics of the instances (see Figure 5.3).

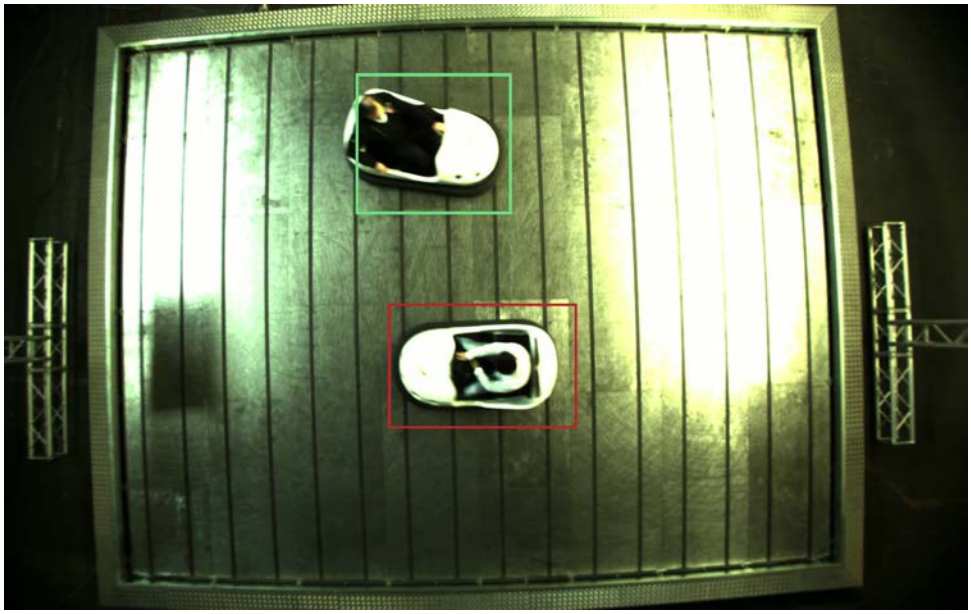


Figure 5.3: An example of the detection of two bumper cars with the corresponding bounding boxes.

## 5.2 Object Recognition Through CNN

In the recent years, a lot of studies and researches are done in the object recognition area and thanks to the advent of the CNNs and the availability of an enormous amount of labeled data, it is possible to train networks with a very high accuracy rate in recognizing objects [4]. Thanks to the surprising results obtained by using CNNs, the object recognition algorithms are used in many computer vision, artificial intelligence, autonomous driving and medical applications [1, 2, 3, 42].
With the advent of the low cost RGB-D cameras in commerce, the object recognition tasks can be used also for robotic grasping by detecting the object grasp where the depth information of the object is an important property for the robot that need to know how far is the object from itself (see Figure 5.4) [52, 53, 54]. An example of a CNN architecture for the prediction of the grasps is shown in the figure 5.5 [53].


Figure 5.4: A robot grasping an object (left), the object grasp in RGB and depth images (right).

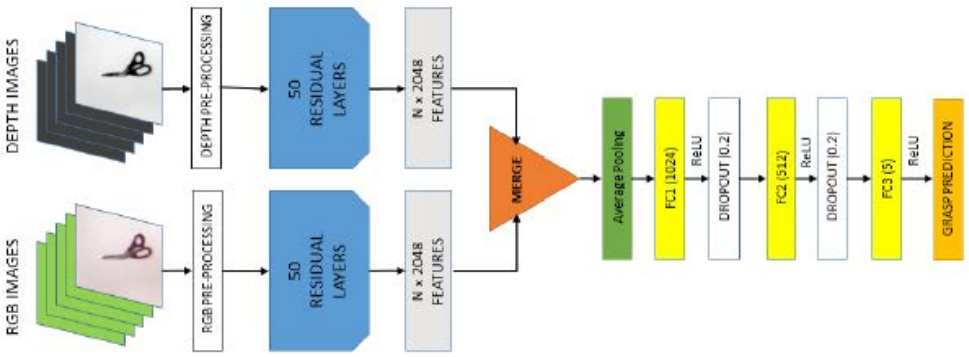



Figure 5.5: An example of CNN architecture for predicting the grasps.

Another new application in which the object recognition algorithms implemented with RGB-D cameras and CNN can be the detection and recognition of objects in 3D spaces. In these applications it is possible to draw and retrieve spatial information of the 3D bounding boxes that outline the objects (see Figure 5.6) [55].

Figure 5.6: An example of 3D bounding boxes in a 3D scene.

In the object tracking systems, the object recognition is the second step of the process. In this step by using the bounding boxes information obtained from the object detector, the detected objects become the input images of the recognition CNN and as output the network returns the corresponding object labels with a recognition confidence field. This field indicates the probability that the detected object belongs to the returned label class (see Figure 5.7).

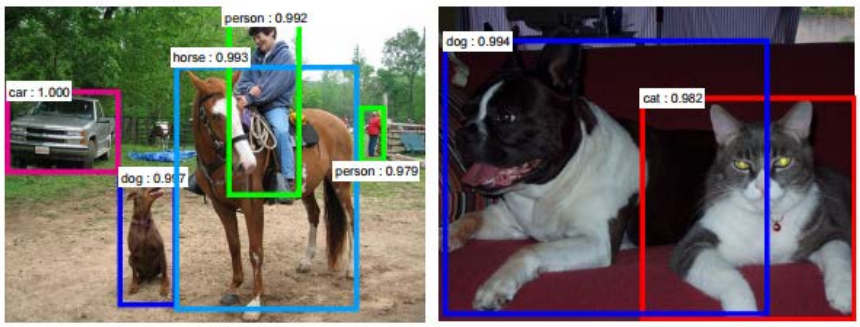



Figure 5.7: Examples of recognized objects with the labels and the confidence rate.

## 5.3 Object Tracking

The object tracking is the term used to indicate the continuously identification of moving object position and its tracking in the video sequences or in the camera streams [11] from its first appearance in the scene to the last one [12] in order to generate the trajectory of the moving object [56]. In the recent years, several types of tracking techniques are developed and they can be essentially classified into three categories: point, kernel and silhouette based tracking (see Figure 5.8).

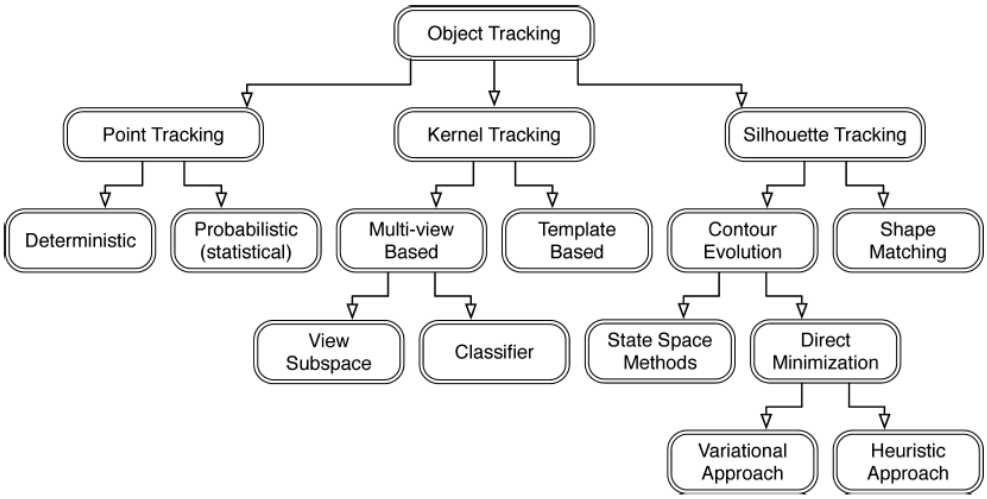



Figure 5.8: Principal tracking typologies.

Most of the existing algorithms are based on the kernel method due to the high accuracy with less computational cost that this method can achieve. But also the point based tracking methods are often used for its low computational cost even if the accuracy is lower [11]. In addition to these three types of tracking techniques, the correlation filter tracking has gained a lot of attention in the area of object tracking because of their computational efficiency and competitive performances [13, 57]. In this tracking technique, the position of the tracked object in the current frame is predicted by using the results of the correlation of the pixel information between the current and the previous frame. Thanks to the pixel correlation between two successive frames, this tracking method can quickly adapt to scale and rotation changes. It is also capable to detecting tracking failure and recovering the tracked object from occlusions during the movement of the target object (see Figure 5.9) [57].

Figure 5.9: Example of robustness of a correlation tracking.

With the advent of the CNNs in computer vision applications, another tracking technique is proposed and becomes rapidly an interest research area. This technique is called *tracking by detection* [58, 59]. In this tracking technique, the position of the object in the current frame is determined by the object detector described in the section 5.1. Then the tracking algorithm links the information of the detected object obtained in the successive frames in order to generate a trajectory of the object (see Figure 5.10,5.11).

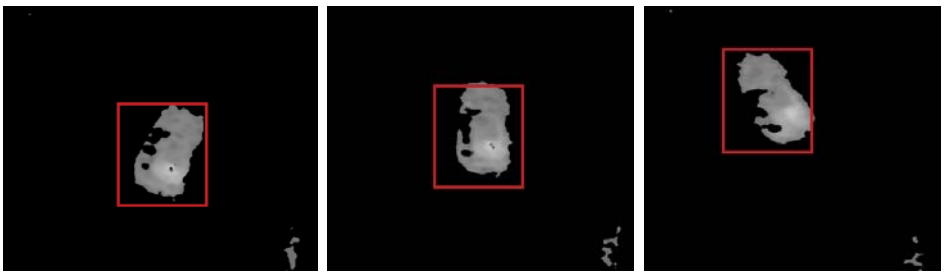


Figure 5.10: Example of the detection of a bumper car in three successive frames.

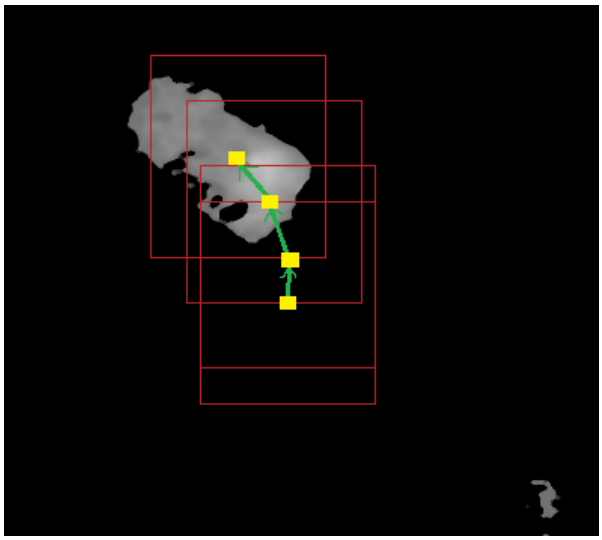


Figure 5.11: The reconstructed trajectory on the fourth frame.

### 5.3.1 Multi Object Tracking

The multi object tracking is important in many computer vision, security and video surveillance applications [19, 17]. This is still a challenging problem since when in the scene there are more instances of the same object category, it is necessary to assign an identity (ID) to each instance and tracking them without losing their IDs (i.e. without swapping or missing their IDs during the tracking process over the frames). All the techniques presented above are developed for tracking a single instance of an object category but the tracking by detection method is one of the suggested method for tracking multiple object tracking systems [18, 19] since the object detector can detect multiple instances of the same object category and return them in an array. Unfortunately using only this method, it is not possible to maintain the IDs of the detected objects since there are no information about it that can be used for linking the ID of an object from one frame to the next one. Furthermore, there are no guarantees that the detector detects the objects in the same order in two different frames, so the index order of the array cannot be a valid ID for the objects. Another problem of this method is that sometimes the detector can fail the detection of some objects due to occlusions or false negatives or can detect new objects. So the number of detected objects can be different in two successive frames. In order to overcome this problem and link the correct IDs of the objects in different frames, an ID association algorithm based on the centroids is proposed.

### Method of Centroids

Starting from the first frame, the method assigns an ID for each detected object, then it computes the centroid of each bounding box (see Figure 5.12, top-left) and go to the next frame. From this frame, the method retrieves the information related to the bounding boxes of the detected objects, computes firstly the new centroids, then it computes the euclidean distances between all centroids of the current and the previous frame (see Figure 5.12, top-right). Now, the method assigns the ID of the object to the one with the smallest euclidean distance from it. After that, if there are still centroids without IDs, they are treated as new instances and for this reason the method assigns to them the next available IDs (see Figure 5.12, bottom-left,5.12, bottom-right).

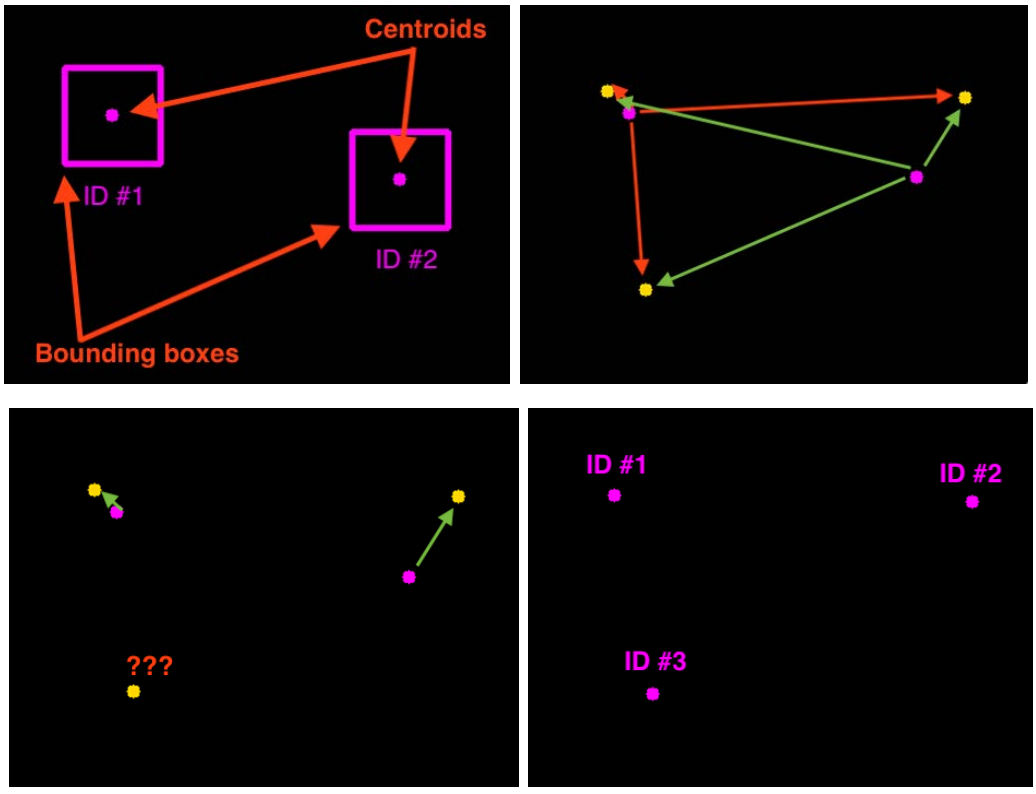



Figure 5.12: Example of the ID association based on the method of centroids.

When the detector detects less objects in the current frame with respect to the previous one, it is useful to use the correlation tracking technique in order to update the missing objects and maintain their IDs. Since this technique uses the correlation information between the sub-image enclosed

by the bounding box of the previous frame and the same location in the current one for updating the tracking object position, the ID association in this case is naturally done. The correlation tracking technique can solve the ID association problem between frames but it is still less accurate with respect to the tracking by detection technique. This is because if the background pixel information and the object pixel information are too similar, the result of the correlation tracker can be inaccurate and it can cause a drifting in the tracking of the object and consequently also loosing the target object.

# Chapter 6

# Bumper Cars Tracking in Low Light Environments

In the recent years, also in the entertainment environments like amusement parks, the computer vision and machine learning techniques become interest research areas. In fact, the Image Studio Consulting company that belongs to the Zamperla's Group, one of the biggest amusement park manufacturer in the world, starts to develop solutions in order to transform traditional and non interactive carousels into interactive ones by implementing add-ons in which computer vision and machine learning techniques are applied. The project that will be presented in the next sections of this chapter will be about an add-on installed onto the bumper cars carousel in order to detect and track the bumper cars inside the carousel in a indoor and low light environment.

## 6.1 Goal of the Project

The goal of the project is design a system that is able to detect and track the bumper cars and sending the positions of the detected bumper cars to a video game designed by using the Unreal Engine. This video game is projected onto the carousel and the tracked bumper cars become the players of the game. In this way, the designed system can add an interactive game to the carousel and give also a sense to the crashes of the bumper cars.

## 6.2 Project Setup: Hardware Components

Since we project the video game on the carousel, it is necessary to have an indoor and low lighted environment otherwise it is impossible to see what is projected if there are too much light in the environment. In this case, using only RGB cameras for capturing the carousel track cannot give us a well defined shapes of the bumper cars. In order to overcome this problem, we decide to use the RGB-D cameras described in the chapter 2 and more precisely the Intel RealSense D435 camera and the MYNT EYE D1000-IR-120/Color camera. The testing carousel track has the dimensions of $6m \times 8m$ (see Figure 6.1) and for an installation constraint, the camera cannot be located at an height that is higher than $4.5m$. So using only one camera we cannot cover the entire carousel track.

Figure 6.1: The testing carousel track.

In this specific case it is sufficient to use two cameras configured as an array of cameras as described in section 3.2.1. Both type of RGB-D cameras are suitable for the project but thinking for the modularity of the add-on in future installations where the carousel track has an area bigger than the testing one, it is more suggested to use the MYNT EYE cameras. Since they have FOVs that are larger than the Intel cameras, it is possible to cover more area by using less cameras with respect to the Intel cameras. A configuration with the MYNT EYE cameras is shown in the figure 6.2.

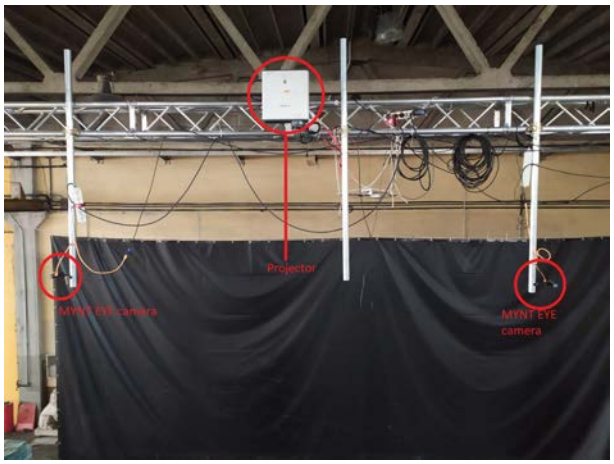



Figure 6.2: The setup of the cameras.

The cameras are oriented in the way shown in the above figure because, at this height, the long side of the covered area can reach more than $6m$ but less than $8m$ (see Figure 6.3). So using the camera in the other orientation, we cannot cover the long side of the carousel track.

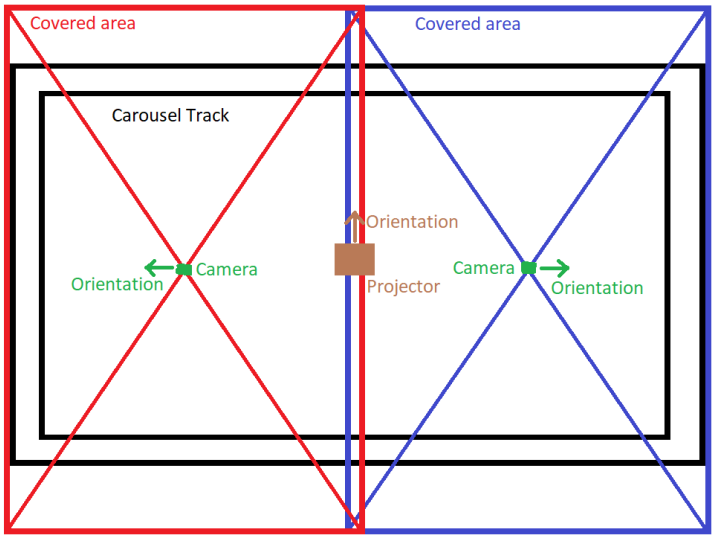



Figure 6.3: The setup of the cameras viewed from the top.

Since the length of the USB cables of the cameras are more or less about one meter long and the computer that processes the images captured by the cameras is located near the control panel of the carousel, we use the active USB extension cables of length 10 and 15 meters in order to extend the USB connection (see Figure 6.4). As discussed in section 3.1.4, we use a single computer with more than one USB controller in order to prevent the saturation on the bandwidth of the controller itself.

Figure 6.4: The active USB extension cable of MutecPower used in the testing setup.

Since the testing carousel is located in a factory that is dislocated from the offices, we use the Asus ROG G752VM laptop computer for testing the correct functioning of the designed system. Even if it is a laptop, with its Intel Core i7-6700HQ CPU and its NVIDIA GeForce GTX 1070 GPU, it can reach performances near to the real-time with also the video game in execution.

## 6.3 Project Setup: Software Components

The designed system need to detect and track the bumper cars in a completely autonomous way without any human support. In order to overcome this requirement, a neural network for detecting the bumper cars and a tracking algorithm are designed and they are better described in the following subsections.

### 6.3.1 The Bumper Cars Detection CNN

We design a neural network in order to give the system the total autonomy to detect the bumper cars present in the carousel. The programs developed for training and testing the neural network are written in C++ programming language. While the structure of the CNN, shown in figure 6.5 [60], is designed by using the machine learning API of the dlib library [60]. This library allows us to train and test the network using the computational power of our GPUs. This reduce drastically the training time that is the most time consuming part of the process.

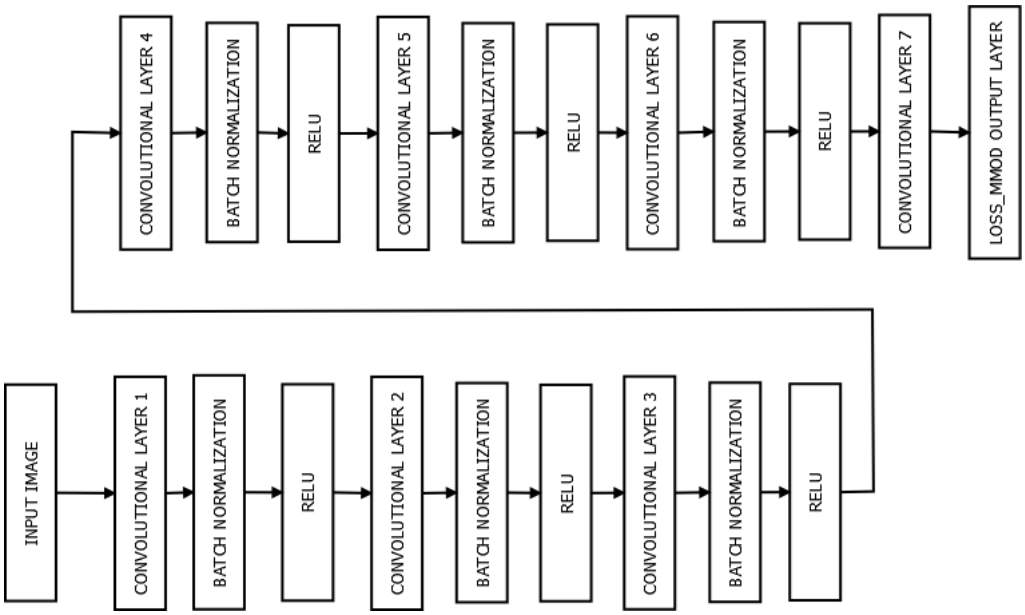



Figure 6.5: The structure of the designed CNN.

The convolutional layers of the CNN have different parameters: the first has 16 feature maps with filter size $5 \times 5$ and stride 2, the second and the third have 32 feature maps with filter size $5 \times 5$ and stride 2, the fourth up to the sixth have 55 feature maps with filter size $5 \times 5$ and stride 1 and the last one has one feature map with filter size $9 \times 9$ and stride 1. The loss_mmod layer returns as output a vector of detections in which every instance corresponds to the detection of a bumper car.
Using this CNN we train two different detectors: the first detects the bumper cars without any other specifications while the second, in addition to the bounding boxes of the detected bumper cars, it specifies also if the bumper car is empty or with someone inside it by returning respectively a status label "empty" or "full".

The training dataset is composed by 7804 labeled depth map images and the testing dataset is composed by 396 depth images that are not present in the training set. Both detectors are trained and tested with these two dataset before testing them on the stream captured from the testing carousel. Initially we use the first detector for retrieving the bounding boxes information of the bumper cars, but we realized that the status information can be an important field for developing the video games for the carousel. For this reason, we decide to use the second detector that give us also the status of the bumper cars.

### Training Process

Part of the images of the training set are captured directly from the testing carousel with a resolution of $1280 \times 720$ without merging the images of the two cameras, then we perform a data augmentation by rotating, eroding and dilating the images in order to create the other part of the training data. The input information for training the CNN, implemented by using dlib library, must contain the input images and the ground truth bounding boxes information for each image saved in a xml file (see Figure 6.6,6.7) because the library interprets that there is a detection in a specific location only if there is a bounding box in that location saved in the xml file.

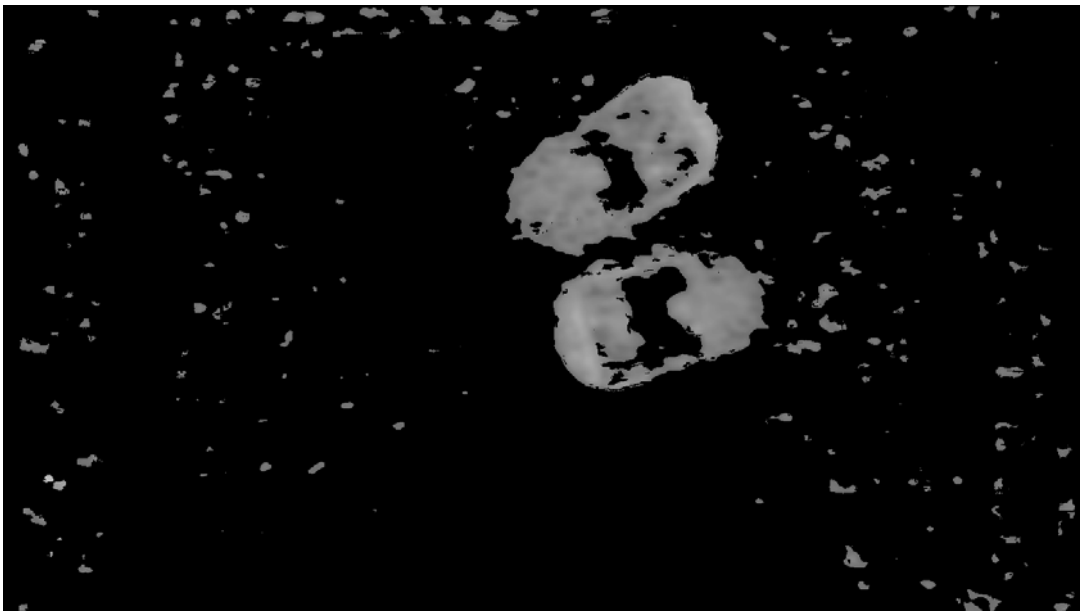


Figure 6.6: The input image file00000797.png of the training dataset.

```
<image file='../outputFrame/file00000797.png'>
    <box top='70' left='601' width='246' height='213'/>
    <box top='286' left='652' width='251' height='173'/>
</image>
```

```
<image file='../outputFrame/file00000797.png'>
    <box top='70' left='601' width='246' height='213'>
        <label>empty</label>
    </box>
    <box top='286' left='652' width='251' height='173'>
        <label>empty</label>
    </box>
</image>
```

Figure 6.7: The corresponding ground truth bounding boxes of the image file00000797.png for the first (top) and second (bottom) detector.

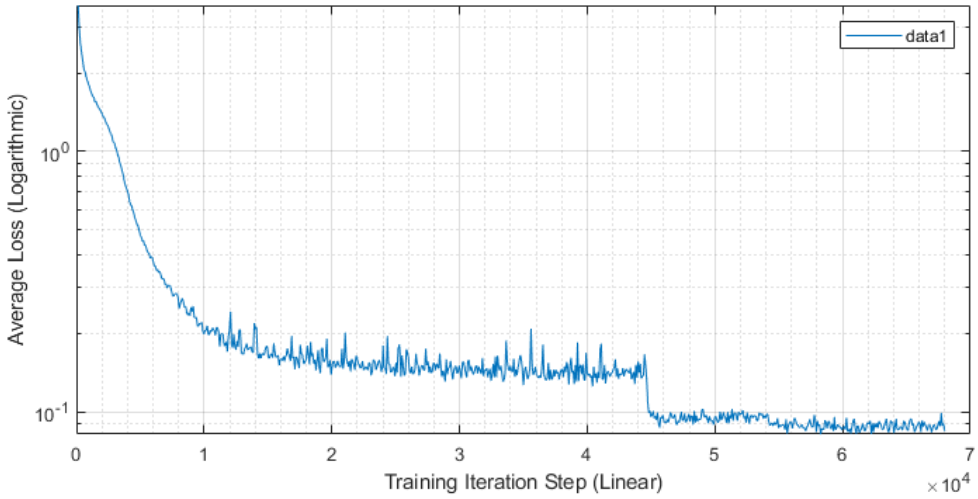



Figure 6.8: The average loss graph of the first detector.

For training the neural network, we use a desktop computer with the following hardware components: a Intel Core i7-6700 CPU, 32GB of RAM DDR4 and 2 NVIDIA GeForce GTX 1080 GPUs. Despite we use multi GPUs for training the network, the training process still takes about 24 hours for completing the process. Even if the number of training data is limited, the average loss reached by the first and second detector during the training process is respectively 0.0843139 and 0.168451. This is also thanks to the random cropper of the dlib library that, during the training process, performs further data augmentation. The figures 6.8, 6.9 show the average loss trend in the

graphs average loss vs training iteration step obtained in the training process for the two types of detector.

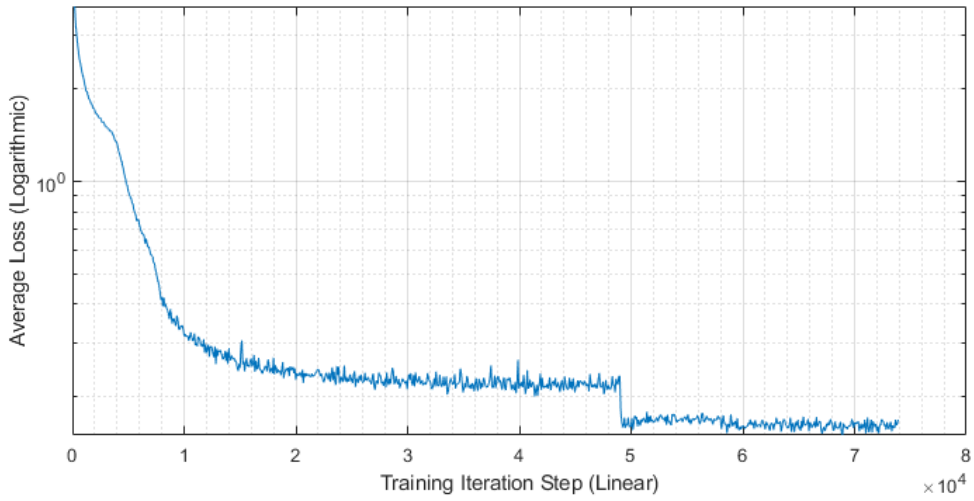



Figure 6.9: The average loss graph of the second detector.

On this training set, the two trained network have obtained an average precision of 0.97282 and 0.966157 respectively for the first and the second detector.

### Testing Process

To get an idea if the trained networks really worked without overfitting, we run them on a testing set that contains images that are not present in the training set. The obtained average precision of 0.983207 and 0.976556 respectively for the first and second detector show us that both the networks are well trained and without overfitting.
After the tests on the testing set, we perform the tests on the testing carousel with the stream images captured by the installed cameras.

### Input Image

Since we use an array of two cameras and for our project we need to have a single view of the carousel track, we perform a fusion of the two captured streams before using it as the input of the network. Thanks to the fact that the cameras are located with a distance between them in order to have as minimum as possible overlapping area, the interference inside this area is negligible. The two streams have a resolution of $1280 \times 720$ and the computer receives them in the orientation shown in the figure 6.10. However, for a

better compatibility between the detection program and the projected game, the streams need to be rotated in order to create a fused stream with the same orientation of the projector. Since the total area covered by the fused stream is bigger than the carousel track (see Figure 6.3), we select in the fused stream only the pixels that correspond to the carousel track and the resulting stream becomes the input of the neural network after the resize of the dimensions of the stream itself in order to speeding up the detection process(see Figure 6.11).

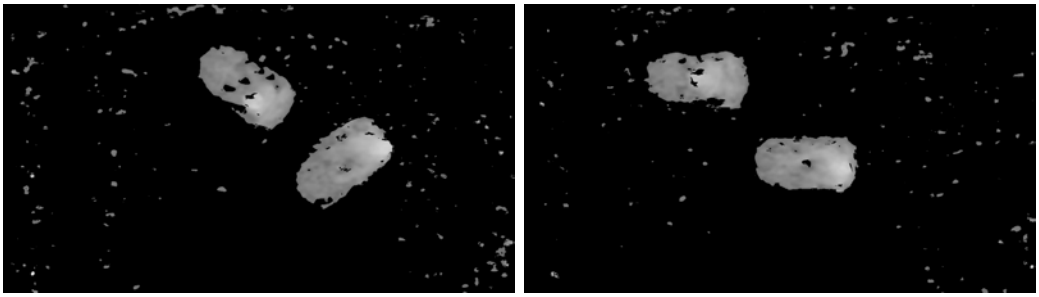


Figure 6.10: The images of the two streams that capture the entire carousel.

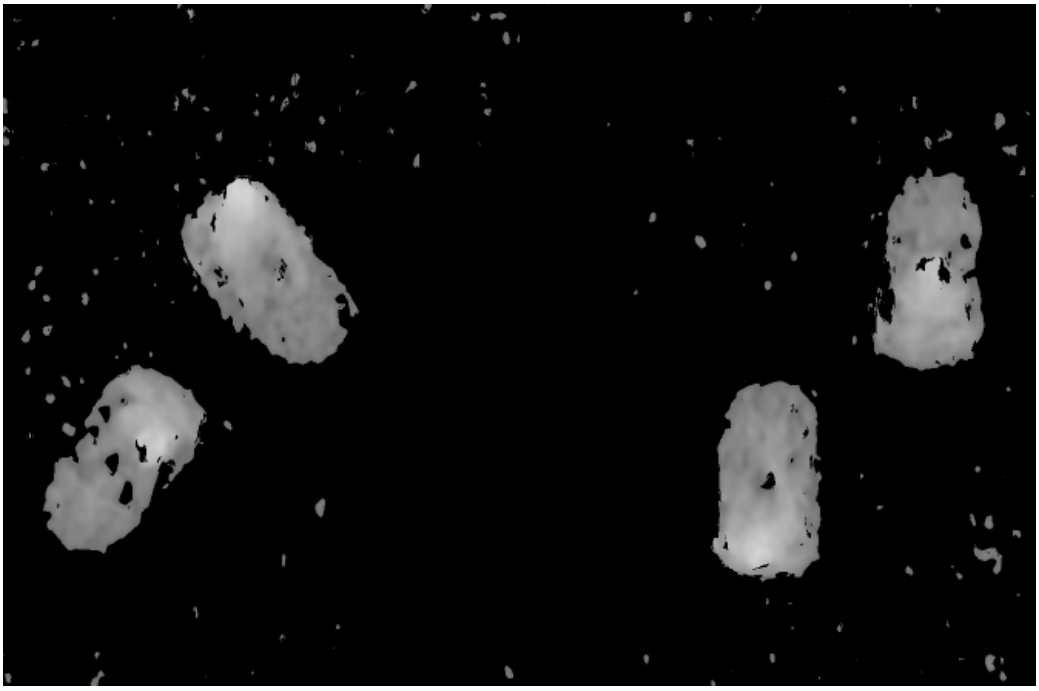


Figure 6.11: The fused image after selecting only the carousel track.

### 6.3.2 The Bumper Cars Tracking

In a multi object tracking system, the occlusion of the target object by part of the background scene, the overlapping between target objects and the entry, exit and return of a target object in the scene can compromise the correct functioning and the correct assignment of the IDs of the tracking system. Fortunately, in our project these three issues do not exist because of the following property of the setup:

- The cameras are located on the top of the carousel, so there are no bumper cars that can be occluded by the background objects.
- The camera stream view is a top view of the carousel track, so the bumper cars cannot be overlapped between them.
- The fused stream covers the entire carousel track, so the bumper cars cannot go outside the cameras views.

These characteristics of our project simplify the tracking process of the bumper cars.

#### Used Tracking Techniques

For tracking the bumper cars over the camera stream, we use principally the tracking by detection technique with the detector trained by our designed CNN presented in the previous subsection. When this technique gives some missing detections, we perform the correlation tracking by using the correlation tracker available in the dlib library. In our project, the detector returns a vector of detections in which each instance contains the information of the bounding box and the other information of the detected bumper car. For maintaining the IDs assigned to the bumper cars over the frames without swapping or missing them, we perform the method of centroids described in section 5.3.1.

#### Tracking Algorithm

Our tracking algorithm, for this moment, is designed for tracking simultaneously 10 bumper cars and it is composed by the following steps:

1. For the first frame the algorithm detects the bumper cars present in the carousel and assigns for each of them an unique ID.
2. For every next incoming frame, we perform the detection and using the method of centroids in order to update the bounding box locations and maintaining the IDs of the corresponding bumper cars.

3. If there are missing detections, we perform the correlation tracking technique exclusively on the missing ones in order to update their bounding box locations and IDs.

4. If there are more detections than the real number of bumper cars, we delete the false positives and update the information of the remaining detections.

5. We save the IDs and the bounding boxes information in the JSON format in order to sent it via UDP packets to the video game.

An example of the tracking result with the JSON format of the information are shown in figure 6.12,6.13.

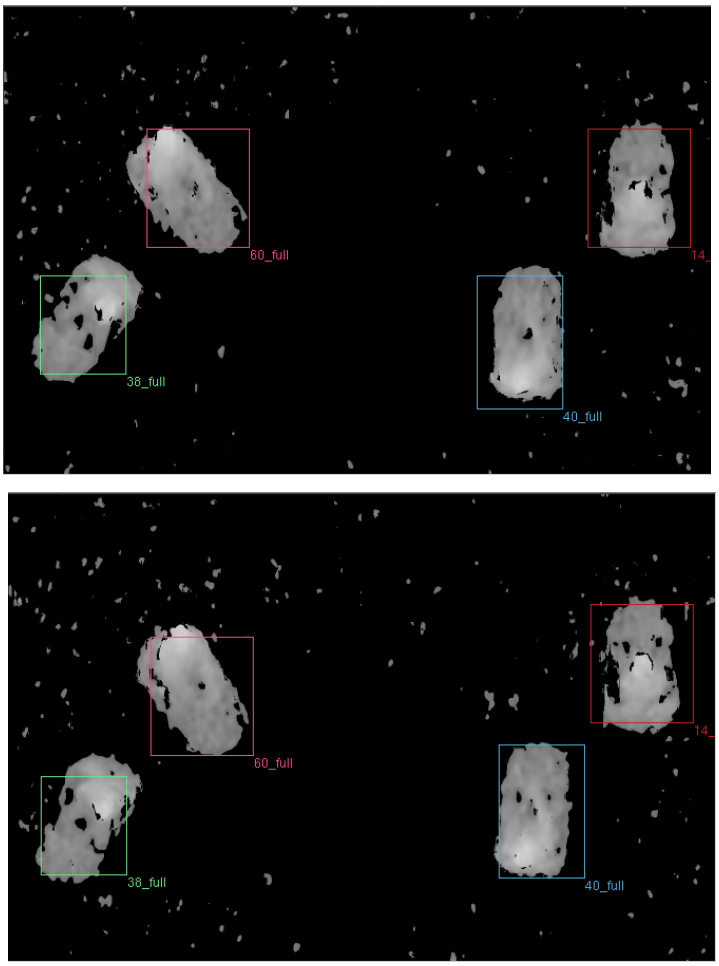
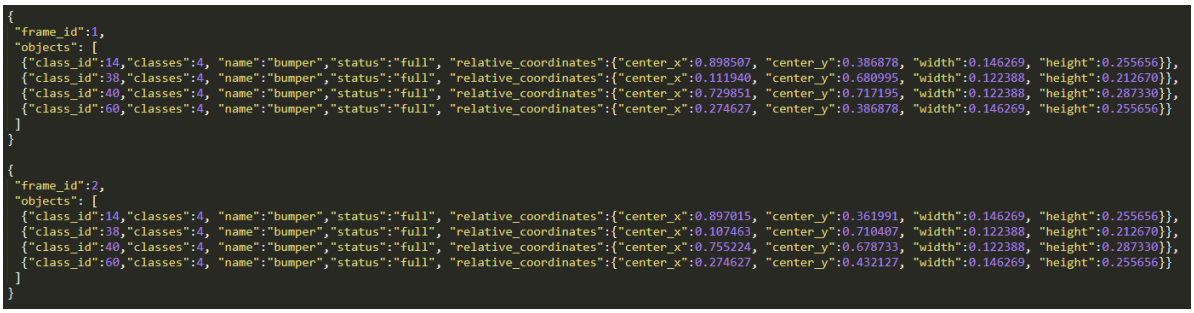



Figure 6.12: The two consecutive frames with the bounding boxes depicted on them.



```
{
 "frame_id":1,
 "objects": [
  {"class_id":14,"classes":4, "name":"bumper","status":"full", "relative_coordinates":{"center_x":0.898507, "center_y":0.386878, "width":0.146269, "height":0.255656}},
  {"class_id":38,"classes":4, "name":"bumper","status":"full", "relative_coordinates":{"center_x":0.111940, "center_y":0.680995, "width":0.122388, "height":0.212670}},
  {"class_id":40,"classes":4, "name":"bumper","status":"full", "relative_coordinates":{"center_x":0.729851, "center_y":0.717195, "width":0.122388, "height":0.287330}},
  {"class_id":60,"classes":4, "name":"bumper","status":"full", "relative_coordinates":{"center_x":0.274627, "center_y":0.386878, "width":0.146269, "height":0.255656}}
 ]
}

{
 "frame_id":2,
 "objects": [
  {"class_id":14,"classes":4, "name":"bumper","status":"full", "relative_coordinates":{"center_x":0.897015, "center_y":0.361991, "width":0.146269, "height":0.255656}},
  {"class_id":38,"classes":4, "name":"bumper","status":"full", "relative_coordinates":{"center_x":0.107463, "center_y":0.710407, "width":0.122388, "height":0.212670}},
  {"class_id":40,"classes":4, "name":"bumper","status":"full", "relative_coordinates":{"center_x":0.755224, "center_y":0.678733, "width":0.122388, "height":0.287330}},
  {"class_id":60,"classes":4, "name":"bumper","status":"full", "relative_coordinates":{"center_x":0.274627, "center_y":0.432127, "width":0.146269, "height":0.255656}}
 ]
}
```


Figure 6.13: The information of the above two frames sent in JSON format to the video game.

### Evaluation of the Algorithm

In order to evaluate the correctness of our tracking algorithm, we project on each bumper car a full filled colored circle, one color for each bumper car. If the color of the bumper car does not change during the test then the algorithm works in a proper manner. Thanks to the combination of the tracking by detection and correlation tracking techniques, our tracking algorithm can track the bumper cars without loosing them or changing their color during all the game. While for evaluate the delay introduced by the tracking system, we measure the computation time of the system from the fusion of the images to the sending of the JSON string to the game. Since the computation time of the system is a crucial field for design a real time application, we save the images only once in all the tests that we performed. In the saved frames, the algorithm uses 95.89% of the time the tracking by detection technique and 4.11% of the time the correlation tracking technique. These percentages indicate that the trained network works very well also on the camera streams. While the average computation time per image is $28.0237ms$. With this computation time we can process more or less 30 frames per second that is also the frame rate of the camera streams. This means that our algorithm can already reach the real time requirement of the project.

### 6.3.3 Issues of the Tracking System

The detector is trained in order to be scale invariant, so if in the captured images there are some noises that the detector detects as bumper cars, we need to ignore them (see Figure 6.14). For this reason we put a lower and an upper bound to the dimensions of the bounding boxes. So if a detected bounding box is too small or too big to represent a bumper car, then it is ignored and it is deleted from the vector of the detections. The bumper cars can crash together. So when we perform the method of centroids, we need to fix a minimum distance between the two centroids in order to determine if they are two crashed bumper cars or they are the same one.
These three parameters depend on the resize scale of the input frame. For this reason, if for different installations, the resize scale changes, then also these parameters need to be recalibrated.

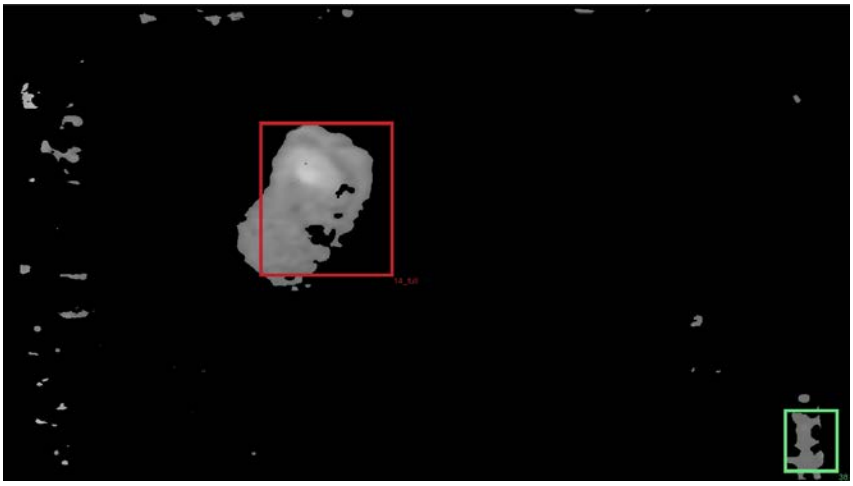


Figure 6.14: An example of the detection of a false positive (green rectangle).

### 6.3.4 The Video Game

The results of the detection and tracking algorithm are sent via UDP packets in the JSON format to the video game. These results are used in order to create the players in the game, in this case they are the bumper cars of the carousel. Then the game creates and spawns some targets that the players can take by passing over them. All these things are directly projected on the carousel and the people can drive the bumper cars in order to catch the target and gain points.

At the end of the game, who gains more points becomes the winner. Some images of the complete project are shown in the following figures.

Figure 6.15: An example of the complete project.

# Chapter 7

# Conclusions and Future Works

## 7.1 Conclusions

In this thesis we presented a bumper cars detection and tracking system in a low light environment by using a CNN and a tracking algorithm designed ad-hoc for the project. For better understanding how the various hardware and software components work, we firstly presented the stereo vision, structured light and ToF technologies that can be used for implementing RGB-D cameras. Then we presented some low cost RGB-D devices and their characteristics and chose from them the cameras used for our project. For increment the covered area, we presented the requirements for designing a good multicamera system, the different camera calibration process and the different topology configuration of the cameras in this system. Since the deep learning framework gave us an important contribution in designing a completely autonomous system for our project, we described its main concepts like the basic computing devices (i.e. neurons) with their activation functions, the main steps of the training process. Then we described how a CNN works and we presented the typologies of image data input that can be used for the CNNs. At this point we presented the concepts of object detection and recognition using the CNNs and the different typologies of tracking techniques that can be used for tracking single or multiple objects. In particular we described the characteristics of the tracking by detection and correlation tracking techniques that are the two used in our project. Then we presented the method of centroids for maintaining the IDs of the tracked objects. In the chapter 6 we presented our project in which we defined the goal that we need to reach, the chosen RGB-D cameras, USB extension cables and computers. Then we presented the CNN architecture designed for the project, its training and testing processes. Finally we described the tracking algorithm,

its evaluation and issues and an example of a designed video game.

## 7.2 Future Works

Since the project is only a prototype of the real add-on for the carousel, we need to fine tune the various parameters in order to reach a higher accuracy with respect to the one reached in the prototype. Then there are also some hardware limits that we need to test when we develop the real add-on such as the maximum number of cameras that a single computer can manage without increasing too much the image processing time and the maximum length of the active USB extension cable that we can use before loosing the signal. Finally, we need also to configure a computer with an adequate hardware components in order to run both the tracking system and the video game without decreasing the frame rate of the output since for the prototype we use a laptop computer.
In this project we use the RGB-D cameras in order to retrieve the depth map of the carousel in a 2D image. As an alternative to this approach, we can study and develop a system that works in the 3D space by implementing pointcloud fusion techniques that can achieve a computational time near to the real-time and designing a 3D object detector that works directly on the pointcloud in order to detect the bumper cars.

# Bibliography


[1] L. Shao, Z. Cai, L. Liu, and K. Lu, "Performance evaluation of deep feature learning for rgb-d image/video classification," *Information Sciences*, vol. 385, pp. 266–283, 2017.

[2] O. Mees, A. Eitel, and W. Burgard, "Choosing smartly: Adaptive multimodal fusion for object detection in changing environments," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 151–156, Oct 2016.

[3] Y. Cheng, X. Zhao, R. Cai, Z. Li, K. Huang, Y. Rui, *et al.*, "Semi-supervised multimodal deep learning for rgb-d object recognition," 2016.

[4] H. F. M. Zaki, F. Shafait, and A. Mian, "Convolutional hypercube pyramid for accurate rgb-d object category and instance recognition," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1685–1692, May 2016.

[5] Z. Wang, J. Lu, R. Lin, J. Feng, *et al.*, "Correlated and individual multi-modal deep learning for rgb-d object recognition," *arXiv preprint arXiv:1604.01655*, 2016.

[6] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum pointnets for 3d object detection from rgb-d data," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[7] Z. Deng and L. Jan Latecki, "Amodal detection of 3d objects: Inferring 3d bounding boxes from 2d ones in rgb-depth images," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[8] R. Socher, B. Huval, B. Bath, C. D. Manning, and A. Y. Ng, "Convolutional-recursive deep learning for 3d object classification," in *Advances in neural information processing systems*, pp. 656–664, 2012.

[9] J. Wang, J. Lu, W. Chen, and X. Wu, "Convolutional neural network for 3d object recognition based on rgb-d dataset," in *2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA)*, pp. 34–39, June 2015.

[10] A. Nguyen, D. Kanoulas, D. G. Caldwell, and N. G. Tsagarakis, "Detecting object affordances with convolutional neural networks," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2765–2770, Oct 2016.

[11] M. Tiwari and R. Singhai, "A review of detection and tracking of object from image and video sequences," *Int. J. Comput. Intell. Res*, vol. 13, no. 5, pp. 745–765, 2017.

[12] T. Kokul, C. Fookes, S. Sridharan, A. Ramanan, and U. A. J. Pinidiyaarachchi, "Gate connected convolutional neural network for object tracking," in *2017 IEEE International Conference on Image Processing (ICIP)*, pp. 2602–2606, Sep. 2017.

[13] H. Nam and B. Han, "Learning multi-domain convolutional neural networks for visual tracking," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[14] W. Chen, L. Cao, X. Chen, and K. Huang, "An equalized global graph model-based approach for multicamera object tracking," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, pp. 2367–2381, Nov 2017.

[15] Y. T. Tesfaye, E. Zemene, A. Prati, M. Pelillo, and M. Shah, "Multi-target tracking in multiple non-overlapping cameras using constrained dominant sets," *arXiv preprint arXiv:1706.06196*, 2017.

[16] J. Son, M. Baek, M. Cho, and B. Han, "Multi-object tracking with quadruplet convolutional neural networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[17] A. G. A. Perera, C. Srinivas, A. Hoogs, G. Brooksby, and Wensheng Hu, "Multi-object tracking through simultaneous long occlusions and split-merge conditions," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 1, pp. 666–673, June 2006.

[18] Y. Xiang, A. Alahi, and S. Savarese, "Learning to track: Online multi-object tracking by decision making," in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[19] Junliang Xing, H. Ai, and S. Lao, "Multi-object tracking through occlusions by local tracklets filtering and global tracklets association with detection responses," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1200–1207, June 2009.

[20] K. Litomisky, "Consumer rgb-d cameras and their applications," 2012.

[21] M. Zollhöfer, P. Stotko, A. Görlitz, C. Theobalt, M. Nießner, R. Klein, and A. Kolb, "State of the art on 3d reconstruction with rgb-d cameras," *Computer Graphics Forum*, vol. 37, pp. 625–652, 05 2018.

[22] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3-d mapping with an rgb-d camera," *IEEE Transactions on Robotics*, vol. 30, pp. 177–187, Feb 2014.

[23] M. Dou, L. Guan, J.-M. Frahm, and H. Fuchs, "Exploring high-level plane primitives for indoor 3d reconstruction with a hand-held rgb-d camera," in *Asian Conference on Computer Vision*, pp. 94–108, Springer, 2012.

[24] A. Vit and G. Shani, "Comparing rgb-d sensors for close range outdoor agricultural phenotyping," *Sensors*, vol. 18, no. 12, 2018.

[25] R. A. Hamzah and H. Ibrahim, "Literature survey on stereo vision disparity map algorithms," *Journal of Sensors*, vol. 2016, 2016.

[26] R. Nair, R. Kai, F. Lenzen, S. Meister, H. Schäfer, C. S. Garbe, M. Eisemann, M. Magnor, and D. Kondermann, "A survey on time-of-flight stereo fusion," *Lecture Notes in Computer Science*, vol. 8200, pp. 105–127, 2013.

[27] Y. V. David Fofi, Tadeusz Sliwa, "A comparative survey on invisible structured light," 2004.

[28] D. Caspi, N. Kiryati, and J. Shamir, "Range imaging with adaptive color structured light," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, pp. 470–480, May 1998.

[29] S. Foix, G. Alenya, and C. Torras, "Lock-in time-of-flight (tof) cameras: A survey," *IEEE Sensors Journal*, vol. 11, no. 9, pp. 1917–1926, 2011.

[30] F. Chiabrando, F. Nex, D. Piatti, and F. Rinaudo, "Integration between calibrated time-of-flight camera data and multi-image matching approach for architectural survey," in *Image & Signal Processing for Remote Sensing XVI*, 2010.

[31] E. J. Almazan and G. A. Jones, "Tracking people across multiple non-overlapping rgb-d sensors," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2013.

[32] W. Lemkens, P. Kaur, K. Buys, P. Slaets, T. Tuytelaars, and J. De Schutter, "Multi rgb-d camera setup for generating large 3d point clouds," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1092–1099, Nov 2013.

[33] M. Munaro, F. Basso, and E. Menegatti, "Openptrack: Open source multi-camera calibration and people tracking for rgb-d camera networks," *Robotics and Autonomous Systems*, vol. 75, pp. 525–538, 2016.

[34] D. A. Butler, S. Izadi, O. Hilliges, D. Molyneaux, S. Hodges, and D. Kim, "Shake'n'sense: reducing interference for overlapping structured light depth cameras," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1933–1936, ACM, 2012.

[35] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[36] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[37] R. C. Gonzalez and R. E. Woods, *Digital image processing*. Upper Saddle River, N.J.: Prentice Hall, 2008.

[38] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[39] C. Seifert, A. Aamir, A. Balagopalan, D. Jain, A. Sharma, S. Grottel, and S. Gumhold, "Visualizations of deep neural networks in computer vision: A survey," in *Transparent Data Mining for Big and Small Data*, pp. 123–144, Springer, 2017.

[40] F. Q. Lauzon, "An introduction to deep learning," in *2012 11th International Conference on Information Science, Signal Processing and their Applications (ISSPA)*, pp. 1438–1439, July 2012.

[41] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[42] D. Ciresan, A. Giusti, L. M. Gambardella, and J. Schmidhuber, "Deep neural networks segment neuronal membranes in electron microscopy images," in *Advances in neural information processing systems*, pp. 2843–2851, 2012.

[43] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[44] A. Eitel, J. T. Springenberg, L. Spinello, M. Riedmiller, and W. Burgard, "Multimodal deep learning for robust rgb-d object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 681–687, Sep. 2015.

[45] L. A. Alexandre, "3d object recognition using convolutional neural networks with transfer learning between input channels," in *Intelligent Autonomous Systems 13*, pp. 889–898, Springer, 2016.

[46] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[47] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 922–928, Sep. 2015.

[48] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[49] C. R. Qi, H. Su, M. Niessner, A. Dai, M. Yan, and L. J. Guibas, "Volumetric and multi-view cnns for object classification on 3d data," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[50] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, "Scalable object detection using deep neural networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[51] Z. Cai, Q. Fan, R. S. Feris, and N. Vasconcelos, "A unified multi-scale deep convolutional neural network for fast object detection," in *european conference on computer vision*, pp. 354–370, Springer, 2016.

[52] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 705–724, 2015.

[53] S. Kumra and C. Kanan, "Robotic grasp detection using deep convolutional neural networks," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 769–776, Sep. 2017.

[54] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg, "Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics," *arXiv preprint arXiv:1703.09312*, 2017.

[55] J. Lahoud and B. Ghanem, "2d-driven 3d object detection in rgb-d images," in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[56] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *Acm computing surveys (CSUR)*, vol. 38, no. 4, p. 13, 2006.

[57] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2544–2550, June 2010.

[58] M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. V. Gool, "Robust tracking-by-detection using a detector confidence particle filter," in *2009 IEEE 12th International Conference on Computer Vision*, pp. 1515–1522, Sep. 2009.

[59] M. Andriluka, S. Roth, and B. Schiele, "People-tracking-by-detection and people-detection-by-tracking," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, June 2008.

[60] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.