POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Politecnico
di Torino

Master's Degree Thesis

Design of a system for the detection and monitoring of falling waste

Supervisors

Prof. Marcello CHIABERGE

Simone CAVARIANI

Federico FEDI

Candidate

Gabriele RODOVERO

December 2021

Abstract

Nowadays, waste management is inefficient because of the inadequacy of the waste differentiation and recycling process. This issue causes an increase in costs in economic and especially in environmental terms. Indeed, since it is more economically convenient to produce new objects by using raw materials instead of recycled ones, the waste continues to increase, and so do the polluting emissions of carbon dioxide.
This thesis work has been carried out in collaboration with ReLearn, an innovative Italian startup whose mission is to optimize the waste treatment process and consider waste no longer a problem but a resource. To simplify recycling and improve waste management, they have developed a smart bin called Nando that is able to automatically differentiate all the waste inserted inside it. By using robotics and artificial intelligence, Nando recognizes the material of which the waste is composed and then sorts it into the correct bin.
This thesis project aims to design a system to estimate the volume of objects placed inside Nando. Volume measurements can be useful to monitor the bin fill level and detect possible objects that get stuck falling into the appropriate container. In order to avoid adding additional hardware that would entail additional costs, the measurement has been performed using only the camera already present inside the bin that was used to recognize the waste material.
Taking advantage of the recent progress in the field of depth estimation achieved by deep learning methods, it has been possible, starting from a single RGB image of the object captured inside the bin, to predict the corresponding depth image.F The Deep Convolutional Neural Network (DCNN) used to estimate depth has been trained on a dataset specifically built for the purpose of the thesis. Subsequently, from the depth image obtained in the previous step, the volume of the waste has been computed with an estimation algorithm specifically developed. Finally, the outcome is a system able to estimate the volume of an object starting exclusively from an RGB image that portrays it. The results of the performed simulations show good scalability of the algorithm and reliable estimation results despite using only a simple camera.

Table of Contents

List of Tables

List of Figures

Acronyms

AI
Artificial Intelligence

ML
Machine Learning

CNN
Convolutional Neural Network

OpenCV
Open Source Computer Vision Library

TOF
Time Of Flight

SFS
Shape From Shading

WTA
Winner Takes All

MDE
Monocular Depth Estimation

MSE
Mean Squared Error

RMSE
Root Mean Squared Error

Chapter 1

Introduction

Nowadays, producing any object using recycled material costs more than producing a new one. This problem is due to the lack or inadequacy of waste separation and recycling methods, which are based on inefficient processes that cause an increase in costs both in economic and environmental terms. Indeed, waste is the fourth cause of polluting atmospheric emissions. Moreover, waste management also affects the daily health, productivity, and cleanliness of the communities. The expansion of the world population leads to an increase in waste, which in turn causes a rise in environmental pollution. This chain effect makes waste a significant global issue.

1.1 ReLearn

ReLearn is an innovative Italian startup whose mission is to improve waste management with technology. ReLearn team develops novel solutions aimed at improving the waste management process. Their objectives are to simplify the separate collection processes, innovate using artificial intelligence, and reduce carbon dioxide emissions due to unethical disposal practices.
Frequently, in our cities, waste separation is not accurately executed. As a consequence, plastic, glass, metal, and paper are not correctly separated. The reasons behind this are the lack of adequate recycling bins in public spaces and commercial areas, as well as human irresponsibility. Due to these inefficient practices, recycling significant amounts of waste is extremely difficult, and we lose the opportunity of giving new value to these materials. This is why ReLearn has developed Nando.

1.1.1 Nando

Nando is a smart bin with a great mission: apply technology to automatize, simplify, and improve waste collection. Its functioning is based on four milestones: image

recognition, IoT sensor monitoring, machine learning, app and data collection.
Nando is able to automatically differentiate all the waste inserted inside of it. Thanks to machine learning and image recognition techniques, it can recognize the material of the object inserted and subsequently deliver it inside the correct container. Specifically, it can automatically differentiate both the macro-categories (paper, plastic, glass, and aluminum) and the micro-categories of waste (different colors of glass, PET, HDPE, PVC, LDPE, PP, PS).
Moreover, by collecting data, Nando can obtain information on the typology and quantity of waste produced in a specific area, as well as the carbon dioxide they generate. Furthermore, through the IoT sensors, it is able to monitor its filling level.

Working principle

Initially, the waste is placed in the appropriate insertion compartment on the top of the device. Then, falling by gravity, the object is positioned on a rotating sorter. Subsequently, the waste is rotated using the sorter, photographed with an RGB camera present inside the bin, and once it is recognized, it is dropped into the correct container. Moreover, thanks to the display near the insertion compartment, Nando communicates the type of waste recognized and guides the user in correct waste separation.

Figure 1.1: Nando smart bin structure.

1.2 Thesis objective

The thesis has been carried out in collaboration with ReLearn with the aim of enriching Nando's functionality. Specifically, the main purpose of the thesis is the design of a system to estimate the volume of objects placed inside the bin. Volume measurements can be useful to monitor the bin fill level, adding more information to the IoT sensors already present inside Nando. The real-time monitoring of the fill level allows optimizing the logistic operations of the trash emptying. Moreover, object recognition combined with the estimation of its volume can be helpful to detect possible objects that get stuck falling into the appropriate container. In order to avoid adding additional hardware that would entail additional costs, the volume estimation has been performed exploting only the camera already present inside the bin that is also used to recognize the waste material. The final outcome should be a system capable of estimating the volume of any object starting exclusively from an RGB image that portrays it. This type of algorithm could then be extended so that it can obtain individual volume estimates from a single photo that portrays multiple objects.

1.2.1 Thesis outline

The thesis is structured in seven chapters, organized as follows:

1. **Introduction**, presents the main objectives of the thesis, with a brief overview of its general structure;

2. **Machine Learning**, provides the main tools to understand artificial intelligence terms and concepts that will be used in the following chapters;

3. **Optical 3D Reconstruction**, analyzes and explains the main passive methods of 3D reconstruction;

4. **Monocular Depth Estimation**, classifies monocular depth estimation methods and presents the state-of-the-art CNN architectures able to perform this task;

5. **Volume Estimation Algorithm**, studies the mathematical model of a camera and applies it to define the strategy used to compute the volume of an object starting from its depth image;

6. **MDE Network Training**, shows the steps involved to train a monocular depth estimation network in order to obtain an accurate depth map starting from a single RGB image;

7. **Conclusions**, comments on the results and limitations of the developed system and presents possible future developments of the work.

Chapter 2

Machine Learning

Machine Learning is the science of programming computers so they can learn from data [1]. In particular, the goal of the computer is to find out what relationship binds the data and use this information to predict new instances.
In the classical programming method, the computer is programmed to produce specific outputs in the presence of certain inputs. In the machine learning approach, the computer is provided with input and output data, and it must find the function that links them in order to be able to predict the output when a new input occurs. Machine learning in this way allows solving complex problems for which it would be challenging to develop a specific algorithm. The power of this tool has become progressively greater mainly for three reasons: the availability of new algorithms that advance every day thanks to the scientific community, the rise in data produced and the exponential increase in the computing power of machines.

Tom Mitchel provides a more precise definition of ML in 1997:
"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E" [2].

- **The Task T** can be basically of two types, *regression* or *classification*. A regression problem arises when the output to be predicted is a continuous quantity (e.g. house price). On the other hand, classification occurs when the outcome is a discrete quantity (e.g. the house is cheap or expensive).

- **The Experience E** is the training data. In the ML approach, the available data are divided into training and testing sets. The computer uses training data to learn. Testing data instead are used to evaluate how the computer behaves when it sees new instances.
  The learning process can be essentially of two types, *supervised* if the dataset

contains *labels* (desired output), *unsupervised* if the dataset does not contain them, and it is up to the model to identify commonalities of data.

- **The Performance P** is, in general, the accuracy of the prediction, so the difference between the generated output and the *ground truth* (desired output). The performances are calculated on the testing data. Indeed, it is not so important that the machine learns perfectly how to model the training data; rather, it is important that the machine acquires the ability to generalize, thus producing correct output on data it has never seen.

2.1 Deep Learning

Deep learning is a sub-field of ML which handles more complex problems using algorithms inspired by the structure and function of the brain's neural networks [3]. Indeed, neural networks employed in deep learning, since they have similarities with the actual biological neural networks, are also called artificial neural networks (ANNs).

2.1.1 Artificial Neural Network

The structure of an ANN is shown in Figure 2.1. It consists of a collection of connected neurons (also called nodes). Each neuron processes a received signal, then transmits the results to other neurons through connections, like the synapses in a biological brain. Neurons are organized into *layers*. Layers between the input and the output layer are called *hidden layers*.

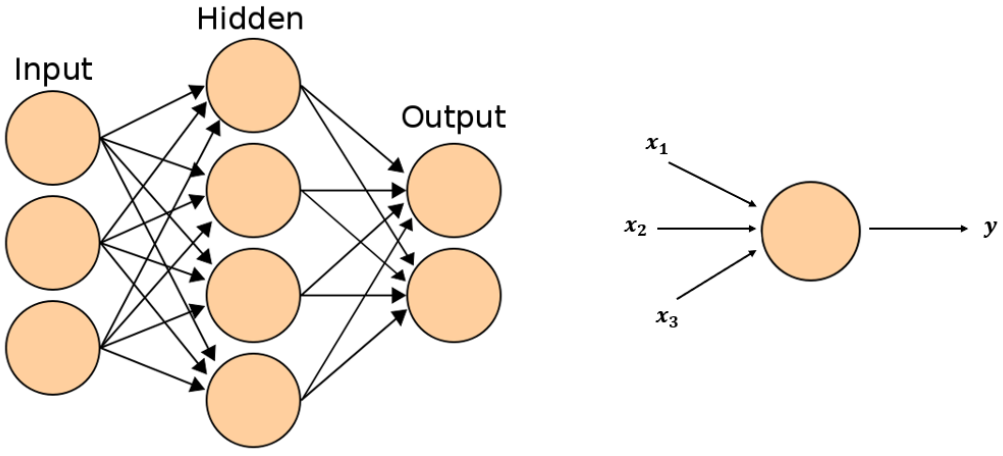


Figure 2.1: Structure of an Artificial Neural Network (left) and of a single neuron (right).

Each connection between two nodes has an associated weight, and each node is described by an activation function (usually non-linear). Therefore, each neuron outputs a value that is the result of a function applied on a linear combination of samples x_i, each one multiplied by a corresponding weight w_i:

$$y = f\left(\sum w_i x_i\right) \tag{2.1}$$

Activation functions

"An activation function is biologically inspired by activity in our brains where different neurons fire (or are activated) by different stimuli" [3].
One of the most adopted activation function is the sigmoid function, shown in Figure 2.2.

$$Sigmoid(x) = \frac{e^x}{e^x + 1} \tag{2.2}$$

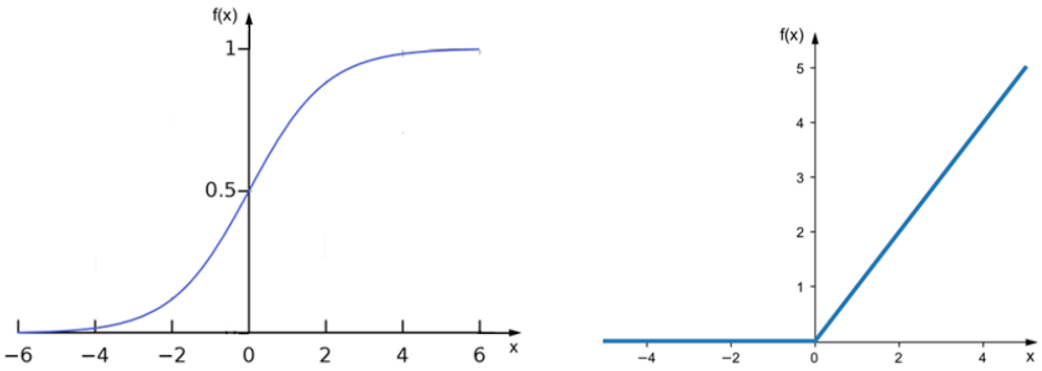
With most positive inputs, the sigmoid function outputs a number very close to 1, and therefore that input activates that neuron. With most negative inputs, instead, this activation function transforms the input into a number very close to 0, and the corresponding neuron is not activated.
Another widely employed activation function is the $ReLU$ (rectified linear unit) function, shown in Figure 2.2. It is defined as follows:

$$ReLU(x) = max(0, x) \tag{2.3}$$

With the rectified liner unit function, the more positive is the input the more activated is the neuron.
Many other types of activation functions apply different transformations to the input. Their choice mainly depends on the type of application and layer for which the function is used.

2.1.2 Learning process

A neural network learns through the training process.
Training a Neural Network consists in resolving an optimization problem.
The problem's variables are the network weights and the objective function, called *loss function* in the ML field, accounts for the error between the correct and the predicted output. Indeed, a loss function typically employed is the mean squared error (MSE).
Therefore, the optimization problem consists of setting the network's weights such that the loss function is minimized.
One of the most adopted optimization algorithms, also called *optimizer*, is the *Stochastic Gradient Descent* (SGD), and it is based on the following steps:



(a) Sigmoid **(b)** ReLU

Figure 2.2: Sigmoid and ReLU activation functions.

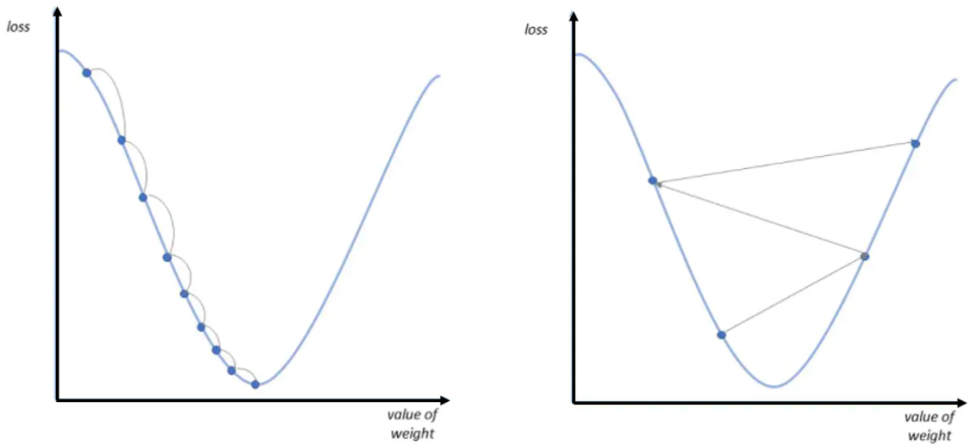
1. The network weights are arbitrarily initialized;
2. The network is supplied with a training sample that propagates from the input to the output (*forward pass*). The computed output depends on the weights that have been assigned to the network connections;
3. Once the output is defined, the loss function can be calculated;
4. The network computes the derivatives of the loss function with respect to each network weight (the gradient), through a process called *Backpropagation*, and multiply them with a fixed number called *Learning Rate*:
5. Each weight is updated as follows:

$$w_i = w_i - (Learning\ Rate \cdot \frac{\partial Loss}{\partial w_i}) \tag{2.4}$$

6. The process repeats starting from the updated weights.

Learning Rate

The Learning Rate parameter (LR) is a value which determines the size of the step through which a parameter is updated.
A too big value of learning rate can lead to a problem called *overshooting*, where too large steps are taken, and the minimum value of the loss function cannot be reached, as shown in Figure 2.3. However, too small LR values cause too slow convergence in the training process that can get stuck. Therefore, the proper choice of this parameter can be challenging, and it is based on a trial and error process.



(a) Small learning rate **(b)** Large learning rate

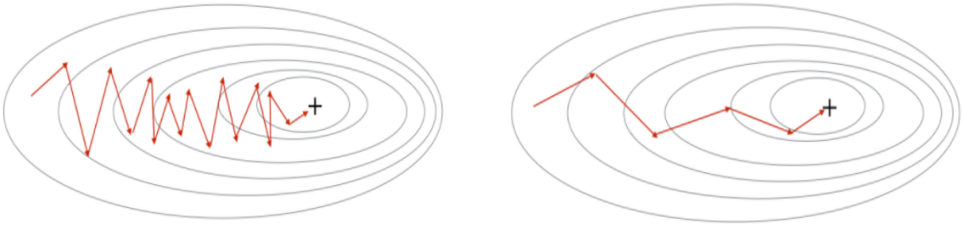
Figure 2.3: Comparison between small and large learning rate.

Optimizers and Batch size

With the Stochastic Gradient Descent optimizer, only one data per step is provided to the network. Therefore, the model is updated as many times as the number of samples contained in the training dataset. Frequently weights updates speed up the training process. In addition, fewer memory resources are required to handle only one sample per step. However, the constant change of the model causes constant fluctuations of the loss function, which can continue to change even if the global minimum has been reached. Therefore, typically in each training step, not only one but multiple data are provided to the network.
When the model parameters are updated in each batch, the optimizer is called *Mini-Batch Gradient Descent*. The batch size defines the number of samples of the training dataset propagated through the network. After that, the error is computed, the weights are updated, and another batch is provided to the network. The more training samples are provided to the network, the more chances arise to update the weights so that the model's performance becomes better. However, the larger the batch size, the more computational resources are required to process the data. A graphical comparison between Stochastic and Mini-Batch Gradient Descent optimizer is illustrated in Figure 2.4

Epochs

The number of epochs defines how many times the network processes the entire training dataset. It must be large enough to allow the algorithm to minimize the loss function.



(a) Stochastic Gradient Descent **(b)** Mini-Batch Gradient Descent

Figure 2.4: Comparison between Stochastic and Mini-Batch Gradient Descent optimizer.

2.2 Convolutional Neural Networks

Convolutional Neural Networks are ANNs, mainly used to analyze images and solves problems such as image classification and segmentation.
Since taking each pixel of the image as input of the neural network would involve a too high computational cost, the *fully-connected layers* used in classical artificial neural networks are replaced with *convolutional layers*.
Convolutional layers consist of a certain number of filters able to detect patterns in the image such as edges, shapes, textures and colors. The deeper the network goes, the more sophisticated are the detected features. So, in later layers, rather than edges and simple shapes, filters may detect higher-level features and specific objects [3].
The typical structure of a CNN, especially for image classification tasks, is illustrated in Figure 2.5. The first part of the network is used to learn and extract increasingly complex features through repeated *convolution* and *pooling* operations. Features are then combined, flattened and inserted one by one within a standard fully-connected network where the image classification takes place.

2.2.1 Covolutional layer

Convolutional layers are the main building block of CNNs and consist of filters capable of capturing increasingly complex features within the image.
Convolutional filters, also called *kernels*, applies a mathematical convolution operation on the input. In particular, they work on a region of the input image at time, called *local receptive field*. The filter matrix convolves a local receptive field of the image and shifts until all the image is covered. This operation consists in the sum of the element-wise multiplication between the kernel and the region of the image processed. Then, the fixed window shifts of a quantity called *stride* and all the results of the convolution operations are stored in another matrix known as

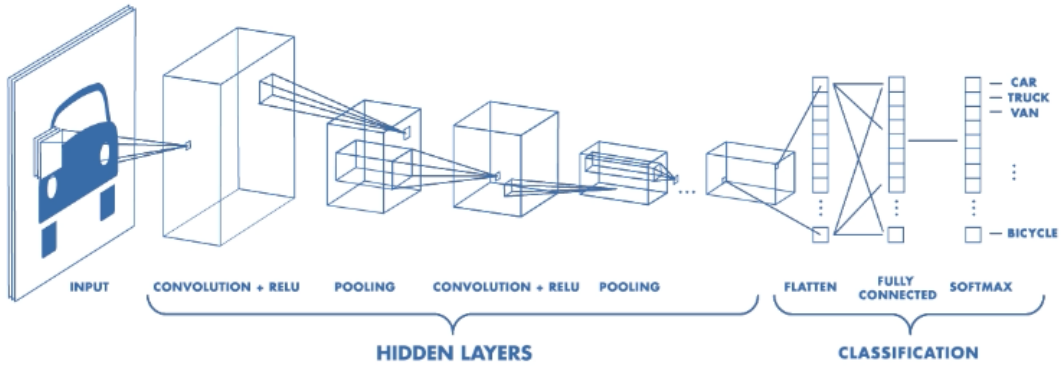


Figure 2.5: Standard CNN architecture used for image classification.

feature map that becomes the input of the next convolutional layer.
Referring to Figure 2.6, the input is a grey-scale image, since it has only one channel, the kernel dimension is 3×3, and the stride is equal to 1. Moreover, to the image is also applied the *zero-padding* technique. It simply consists in adding zeros around the image margin, increasing the image dimension such that the image borders will be convolved the same number of times as the central pixels of the image. Taking into account an RGB image, the complexity of the convolution operation is greater, but the working principle is always the same; the only difference is that the kernel will be composed of three channels.
Convolutional layers are often followed by *pooling layers* used to decrease the spatial resolution of the convolved image. The most adopted one is the *max pooling*, which reduces the size of the feature map (the convolutional layer output) by considering only the most relevant features.
Specifically, the max pooling is a filter with a certain dimension and stride that overlaps the feature map, obtained through the convolution operation, and selects the maximum value within the feature map region covered by its window size. Referring to Figure 2.6, it is used a 2×2 max pooling filter with a stride of 2.
Another widely adopted pooling layer is the *average pooling* which picks the average value of the window instead of the maximum.

Dilated convolution

Besides the classical convolution, there are other types of convolution; one of the most relevant is the *dilated convolution*, also called *atrous convolution*. It is used to capture patterns and features with a wider field of view without adding

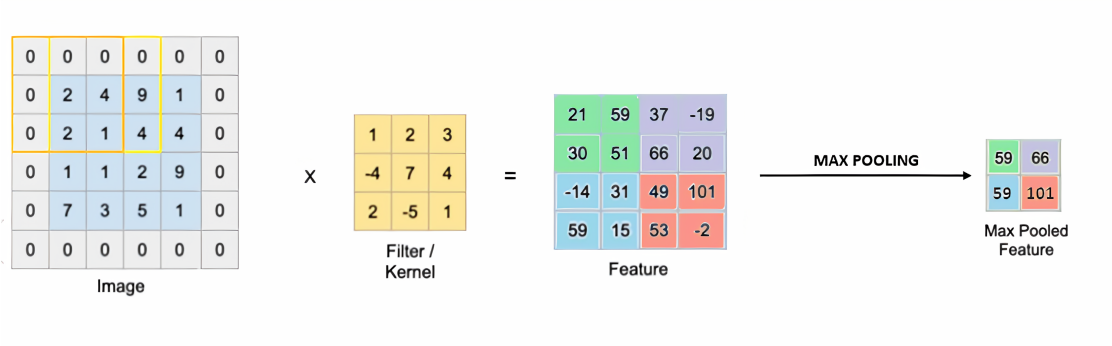
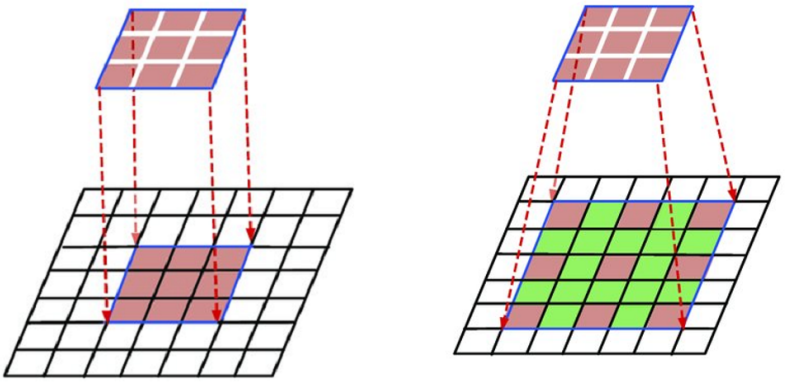


Figure 2.6: Convolution and max pooling operations.

computational costs. A dilated filter is a standard filter in which are added zeros between each kernel element. The amount of spacing between each filter element is a parameter called *dilation rate* d.
In Figure 2.7, is shown as a 3×3 dilated convolution with dilation rate $d = 2$ has the same receptive field of a classical 5×5 convolution while using the same number of parameters of a 3×3 convolution.



(a) 3×3 classical convolution

(b) 3×3 dilated convolution with $d = 2$

Figure 2.7: Comparison between classical and dilated convolution.

Transposed convolution

So far, the analyzed convolution and pooling operations cause a reduction in the spatial resolution of the image called *down-sampling*. In certain applications, it may be useful to map the extracted features in a larger dimensional space or to

recover the initial resolution of the image (*up-sampling*).
An important type of convolution that allows for up-sampling is *transposed convolution*, also called *deconvolution*. Usually, transposed convolutions are performed through classical convolutions adding zeros to the input image.
An example is illustrated in Figure 2.8. The 2×2 input image is padded with a 2×2 border of zeros. After applying transposed convolution with a 3×3 kernel and stride equal to 1, it is possible to obtain a 4×4 output image. In this way, the resolution of the input image is doubled.

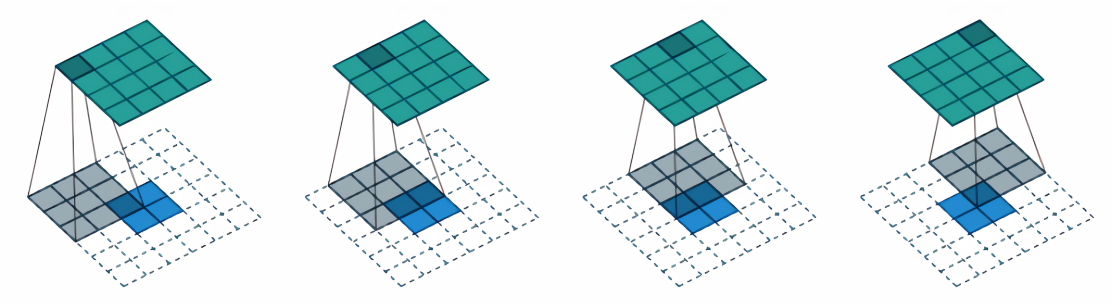


Figure 2.8: Example of a transposed convolution.

2.2.2 CNN architectures

This section briefly analyzes the evolution of the structure and building blocks of the most relevant CNNs.

AlexNet & VGG-16

One of the first networks able to solve complex image classification problems was *AlexNet*. It has a structure very similar to the one illustrated in Figure 2.5, and in particular, it is composed of 8 layers, 5 of which are convolutional and 3 fully-connected. Subsequently, scientific researchers have noticed that to improve CNN's performance it was necessary "to go deeper". For this reason, *VGG-16* was developed, with a structure very similar to AlexNet but made up of 16 layers.

GoogLeNet

GoogLeNet is a 22 layer CNN that, with its fundamental building block called *inception module*, has revolutionized the structure of CNNs, going wider rather than deeper. Indeed, GoogLeNet was the first network to use convolution and pooling operations no longer all in sequence but also in parallel, creating a "network in the network".

In previous works, there was the need to choose whether use the pooling or convolution operation and the size of their filters. Thanks to the introduction of the inception module, it is possible to apply all these operations in parallel, as shown in Figure 2.9. The obtained results are then concatenated. This technique allows to extract features at different levels and fuse them.

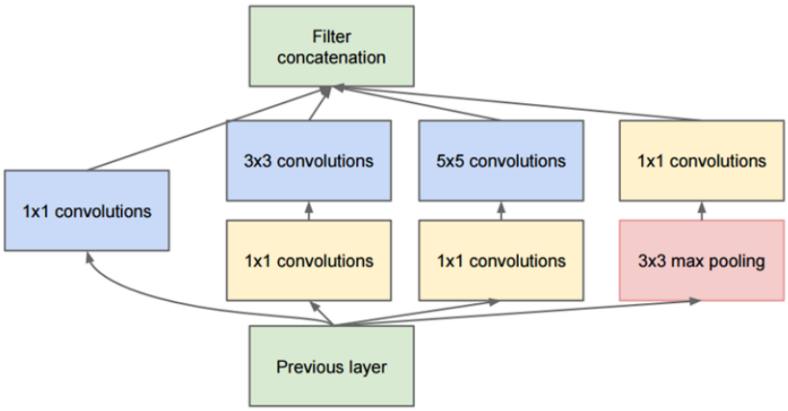


Figure 2.9: Inception module.

However, all these convolution operations are computationally expensive. To solve this problem it has been introduced the 1×1 filters. As shown in Figure 2.10, the 1×1 kernel allows to maintain the same resolution of the input feature map but merging all the information distributed over multiple channels into a single output channel; this effect is called *dimensionality reduction*. In this way, the subsequent 3×3 and 5×5 filters work on an image with fewer channels, which exponentially lowers the required operations.

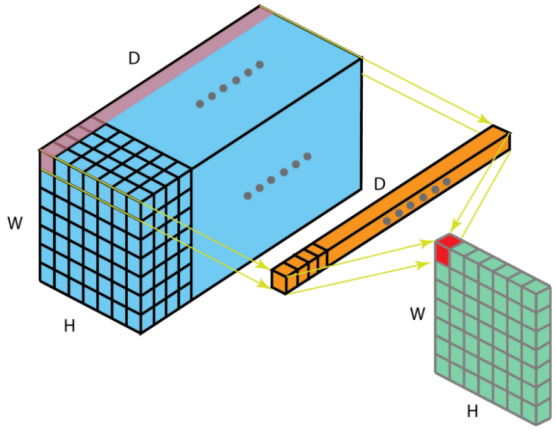


Figure 2.10: 1×1 kernel: dimensionaly reduction.

ResNet

As mentioned earlier, increasing the number of layers within a CNN also increases the accuracy of the prediction. However, once a certain number of layers is reached, the performances saturate and at a certain point exponentially degrade. This problem was solved by introducing the *residual block*, the fundamental block present within *ResNet*. Residual block, also called identity block, uses skip-connection (or shortcut connections, residuals), by which it is possible to insert more than 100 layers within a single CNN without reducing the network efficiency.
Referring to Figure 2.11, in traditional CNNs, the input x propagates through convolutional layers and becomes $H(x)$. Thanks to the residual block, instead of directly calculating the transformation from x to $H(x)$, it is only computed the term that must be added to x in order to obtain $H(x)$. This term is $F(x)$ and is called the residual. "It is easier to optimize the residual mapping than to optimize the original, unreferenced mapping" [4].

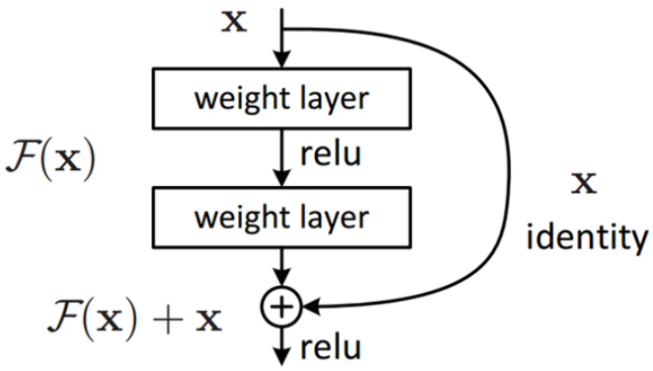


Figure 2.11: Residual block [4].

Chapter 3

Optical 3D Reconstruction

The goal of optical 3D reconstruction is to capture the 3D geometry and structure of objects and scenes. This task is fundamental to many applications such as computer graphics, computer vision, medical diagnosis, and virtual reality.

3.1 Taxonomy

Reconstructing a model implies that all its coordinates in space have to be known to define its profile; this procedure can be achieved either by Passive methods, unlike active ones,

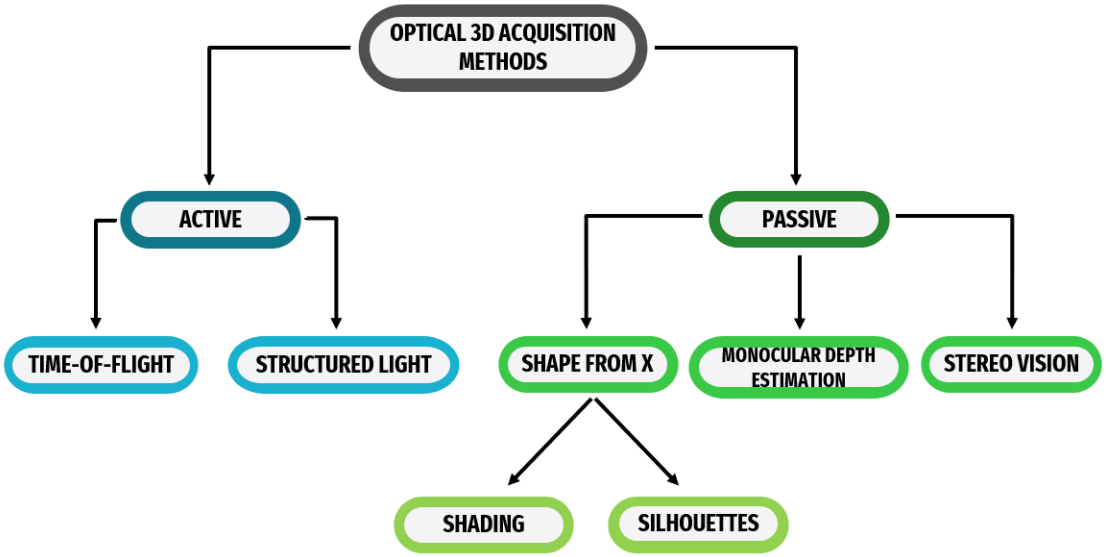


Figure 3.1: 3D shape acquisition techniques taxonomy.

3.1.1 Active methods

Active methods are based on how the object interacts with the light rays emitted by the sensor and usually require the adoption of complex and expensive technologies such as rangefinders or lasers.
In particular, time-of-flight lasers measure the time taken by the wave to travel a certain distance in a specific medium. They calculate the time needed by the wave to go from the emitter to the receiver (Figure 3.2).

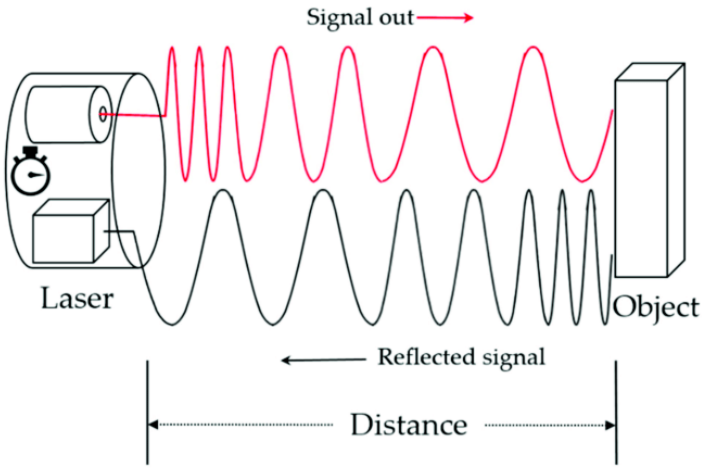


Figure 3.2: TOF sensor working principle.

On the other hand, structured light technologies are based on the projection of known patterns on a scene. The projected image is deformed, as the light rays hit the object, and through this information, the sensor can measure the object's depth and understand the surface features (Figure 3.3).

"Active approaches emit a signal and process how the scene reflects it, while passive approaches rely on changes in the signal emitted by the scene itself" [5].

3.1.2 Passive methods

Passive methods are mostly image-based, so the object shape is recovered from one or multiple 2D images. They do not require expensive sensors, just simple cameras. While it is usually an easy task for a human to understand the 3D structure of the objects shown in an image, it is not as simple for a computer because, when the picture is taken, the projection process involves the loss of one dimension, so estimating the proper 3D geometry becomes complex. For this reason, image-based 3D reconstruction is a so-called ill-posed problem, because many different 3D surfaces may produce the same set of images [6].

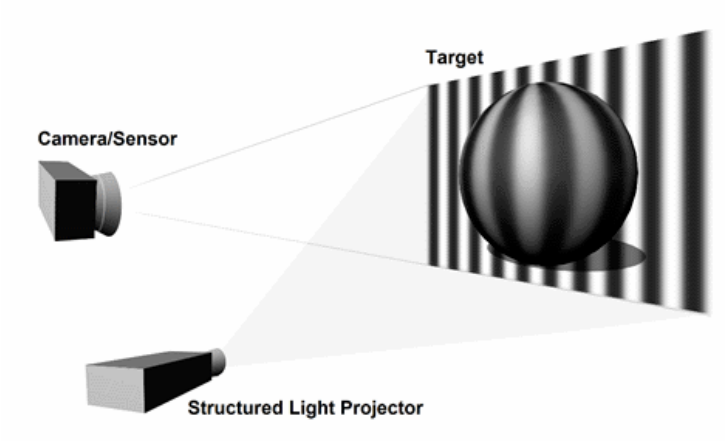


Figure 3.3: Structured light technology.

Passive methods can be divided essentially into two groups. The first generation of methods approached the problem from the geometric perspective; they focus on the projection process from a mathematical point of view and try to reconstruct the object through well-calibrated cameras [7]. As shown in Figure 3.1, stereo vision and shape from X methods belong to this category.
The second generation of methods tries to exploit our ability, as human beings, to estimate the shape and size of an object represented in a picture. This is because, over the years, we have built mental models and stored information that allows us to understand how things look like. Therefore, in the second approach, such methods try to rebuild this previous knowledge through deep learning techniques, and the 3D reconstruction problem is formulated as a ML regression problem. So, the depth estimation task is performed by means of CCNs without requiring a complex camera calibration process [7].

3.2 Shape from X

In computer vision, the methods to recover the shape of the objects from one or multiple images are called shape-from-X techniques, where X can be shading, silhouette, texture, motion, etc.

3.2.1 Shape from Shading

As can be guessed from the method's name, the objective is to infer the object's shape from one or more images with variable levels of darkness.
The Shape from Shading takes advantage of the fact that a grayscale image can

give an idea of the 3D structure of the object, as can be observed from Figure 3.4. However, it is an ill-posed problem since many shapes can generate the same image.

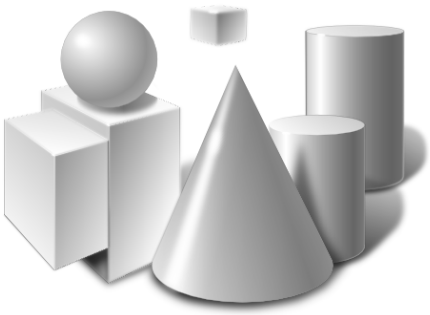


Figure 3.4: Shading gives a cue for the object's 3D shape.

Since a smooth surface has a tangent plane at each point, the object's shape can be easily described by the surface normal at every point. Therefore, the shape determination is based on the surface normals computation.

Surface radiance

There exists a relationship between the radiance (brightness) of a pixel (x, y) and the surface shape:

$$I = \rho \, \boldsymbol{n} \cdot \boldsymbol{s} \tag{3.1}$$

Where:

- I is the observed intensity of a pixel;
- ρ is the surface albedo;
- $\boldsymbol{n} = (n_x, n_y, n_z)^T \in \mathbb{R}^{3x1}$ is the surface normal;
- $\boldsymbol{s} = (s_x, s_y, s_z)^T \in \mathbb{R}^{3x1}$ is the source direction (illumination direction), its module represents the amount of light that falls on the surface.

As seen from the camera, the brightness of the surface is linearly correlated to the amount of light falling on the surface.

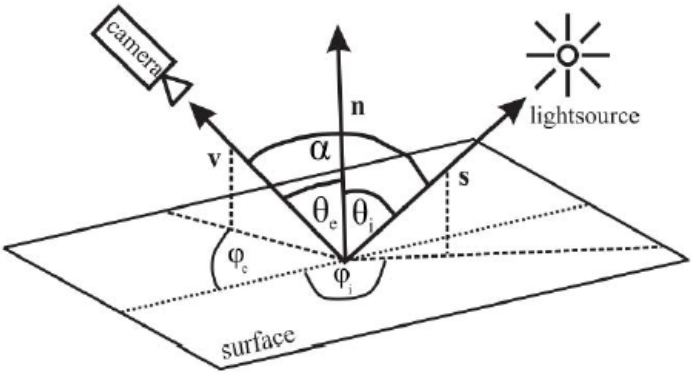


Figure 3.5: Shape from shading principle: determination of surface normals from their appearance on the image [8].

Surface Albedo

The albedo is the fraction of light radiation that is reflected from the surface. It gives an idea of the reflective power of the material, and, considering the same material, it depends on the wavelength of the light radiation. The maximum albedo is 1 when all the amount of incident light is reflected; on the other hand, when all the amount of incident light is absorbed, the albedo is equal to 0.

Assumptions

To apply Equation 3.1 some assumptions must occurs:

- The light source $\boldsymbol{s}$ direction and intensity are known;
- Surface characteristics ρ are known;
- The surface is Lambertian and the albedo is uniform over the entire surface.

Lambertian surface

There are two types of reflection models, the specular reflection and the diffuse one. In the first case, the light that affects the object is reflected in all directions of space, and surfaces behaving in this way are called Lambertian.
On the contrary, in the specular reflection model, the incident light is reflected in a single outgoing direction, symmetrical to the surface normal (Figure 3.6). This behaviour is typical of specular surfaces (mirror-like).
So, a Lambertian surface appears equally bright from all viewing directions, and

the observed intensity I is independent of the viewing position, as can be seen from Equation 3.1.

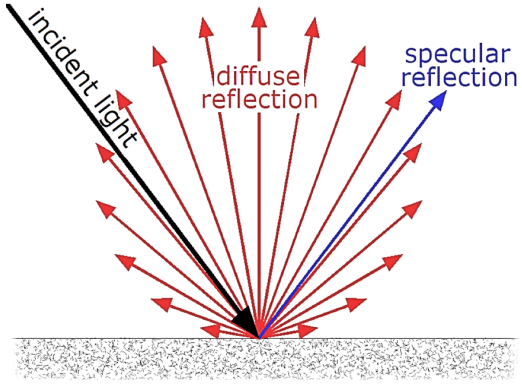


Figure 3.6: Comparison between diffuse and specular reflection.

Surface normal

By describing the surface shape in terms of the surface normal, Equation 3.1 has three unknowns (n_x, n_y, n_z). So, in order to simplify the problem, the surface shape can be described in terms of the surface gradient [9].
Surface orientation can be parameterized by the first partial derivatives of z:

$$p = \frac{\partial z}{\partial x} \quad , \quad q = \frac{\partial z}{\partial y} \tag{3.2}$$

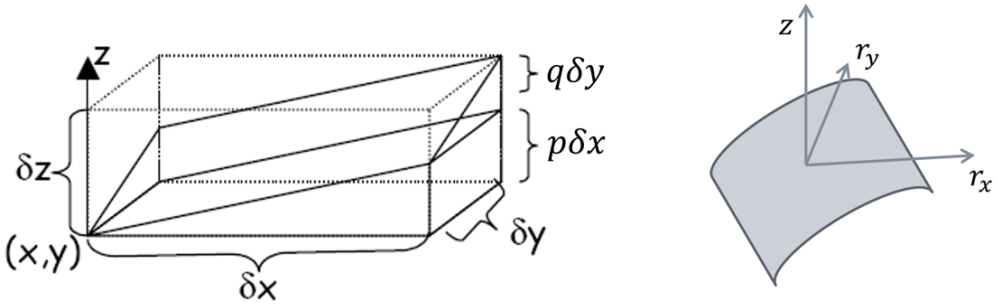


Figure 3.7: Surface parameterization.

Then, it is possible to express the surface normal as a function of the two normal

directions p and q:

$$\boldsymbol{r_x} = (p,0,1) \quad , \quad \boldsymbol{r_y} = (0,q,1) \tag{3.3}$$

And finally:

$$\boldsymbol{n} = \boldsymbol{r_x} \times \boldsymbol{r_y} = (p,q,-1) \tag{3.4}$$

Nevertheless, the problem is not yet solved because Equation 3.1 becomes a nonlinear equation with two unknowns. "Only one intensity value $I(x,y)$ is available per pixel, but two independent normal directions p and q have to be determined" [8].

Reflectance map

The reflectance map is defined as a function $R(p,q)$ proportional to the image intensity I:

$$R(p,q) = \boldsymbol{n} \cdot \boldsymbol{s} = \frac{1 + p_s p + q_s q}{\sqrt{1+p^2+q^2}\sqrt{1+p_s^2+q_s^2}} \tag{3.5}$$

Where $(p_s, q_s, -1)$ is the illumination direction.
The reflectance map is a tool used to visualize the scene radiance as a function of the surface orientation, and it based on a set of nested iso-contours corresponding to the same observed brightness.

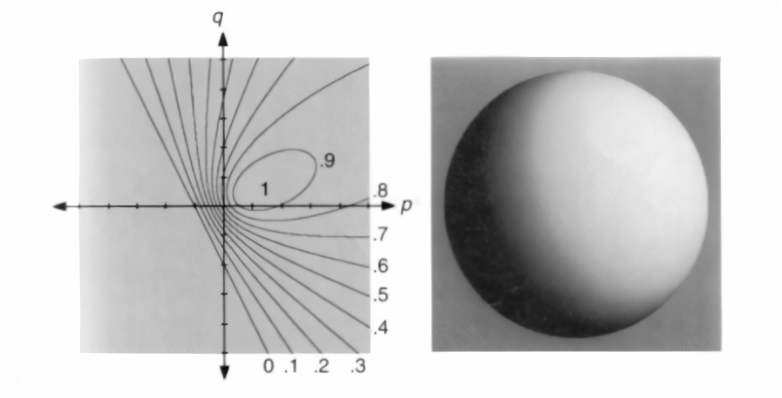


Figure 3.8: Reflectance map of a Lambertian surface.

Knowing the object features, the reflectance map provide, for each gray level, all the compatible orientation $(p,q,-1)$ of the surface.

Solution of the problem

As mentioned before, there are two unknowns, p and q, but only one relationship, so the only way to solve the SFS problem is to introduce additional constraints or to use more images.
In the last case, the problem is also known as *Photometric stereo*, and it is based on taking multiple images of the same object but with different lighting conditions. In this manner, drawing the iso-brightness contour in the reflectance map for a particular point of the surface and changing the illumination conditions, it is possible to obtain an intersection of lines that defines the normal vector of the surface in that point:

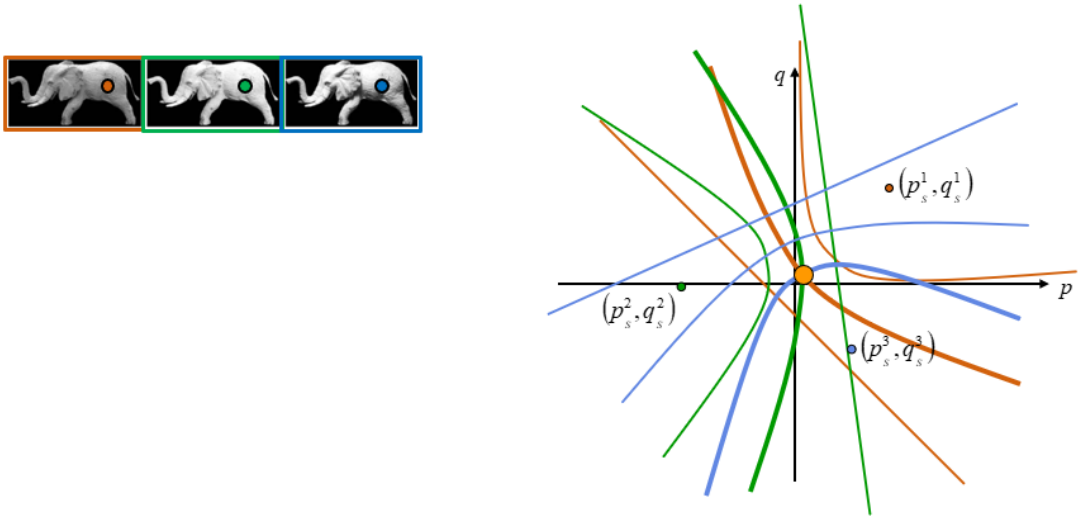


Figure 3.9: Photometric stereo approach: intersection of iso-brightness contours define the surface normal.

3.2.2 Shape from Silhouette

The Shape from Silhouette problem consists in reconstructing the shape of an object through silhouette images.
As illustrated in Figure 3.10, a silhouette image is a binary image where the background is removed from the foreground, and it can be easily obtainable through a segmentation mask.

The foreground mask, known as a silhouette or occluding contour, is the 2D projection of the corresponding 3D foreground object.
The Shape from Silhouette method is based on multiple silhouette images, also called

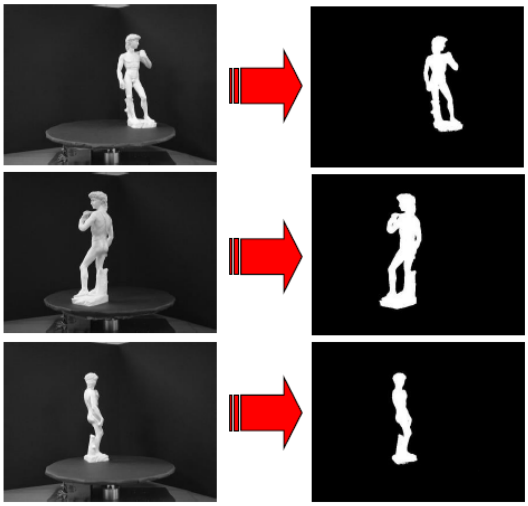


Figure 3.10: Comparison between the image (left) and the corresponding silhouette image (right).

reference images, of the same object from different points of view. Considering the camera intrinsic parameters and back-projecting the silhouette area, it is possible to obtain a cone, the *Visual Cone*, in which it is known that the real object will lay. Repeating the process for each reference image, Visual Cones intersect each other obtaining a volume that contains the object to be reconstructed. It is an upper bound of the volume of the actual object, and it is called *Visual Hull*.

Referring to Figure 3.11, suppose there are K fixed cameras C^k with $k = (1, ..., K)$ positioned around an head-shaped object O. Using the silhouette images S_j^k with $k = (1..., K)$ taken at time t_j, an upper bound of the object's shape, called Visual Hull, can be built by intersecting the Visual Cones defined by the camera intrinsic parameters. As K increases, the Visual Hull converges to the real shape of the object.

As mentioned before, the Visual Hull always contains the object, and it provides an upper bound on the shape of the actual thing. This conservative property is advantageous in applications such as obstacle avoidance and visibility analysis. "On the other hand, if there are only a few cameras, the Visual Hulls obtained using Shape from Silhouette can be a very coarse approximation to the shape of the actual object. This poses a big disadvantage for Shape from Silhouette in applications such as detailed shape acquisition and realistic re-rendering of objects" [10].
As previously stated, the number of cameras used is essential for good results; however, the intersection of many volumes can be a high computational cost operation. As a result, Shape from Silhouette algorithms rely primarily on quantizing the 3D

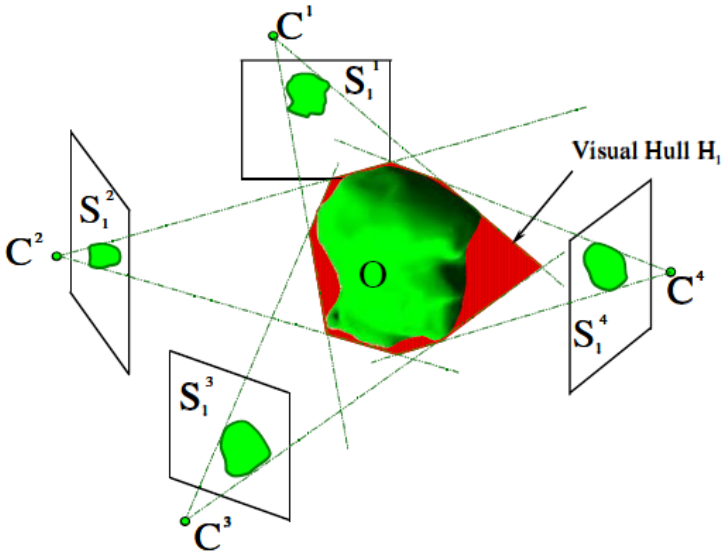


Figure 3.11: Shape from Silhouette working principle: intersection of Visual Cones define a volume that certainly contains the object, the Visual Hull [10].

area or performing intersections in a 2D space to have a lower computational cost. As an example, one of the most popular algorithms is the standard *voxel-based* method. A voxel is the 3D equivalent of a pixel. As shown in Figure 3.12, the algorithm is based on the following steps:

1. The space taken into consideration is discretized in many small voxels;

2. Back-projecting the silhouette images, Visual Cones intersect with voxels;

3. Only the voxels that lie inside all the Visual Cones will be part of the final shape; the others are eliminated from the final volume .

The result, as can be observed in Figure 3.13, it is a quantized representation of the Visual Hull according to the given volumetric grid.

Many more efficient algorithms are based on this method, such as *Space-Carving* or *Marching Intersections*. These methods usually have large memory requirements since they rely on a volumetric representation of the object. Other algorithms are image-based, where all the computations are made in the "image space" coordinates of the reference images. They reduce computation complexity and do not suffer from limited resolution and quantization artifacts of the volumetric approaches [11].

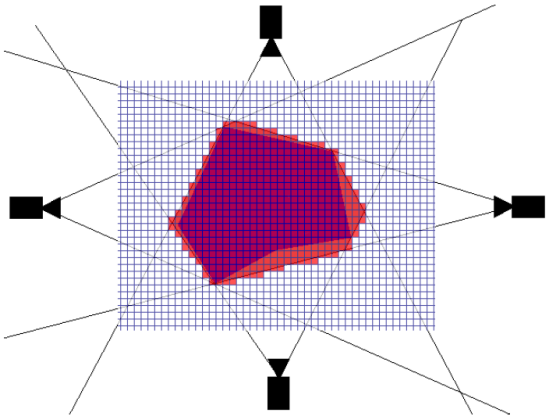


Figure 3.12: Voxel-based algorithm: in purple the object to be reconstructed, in red the corresponding reconstructed volume.

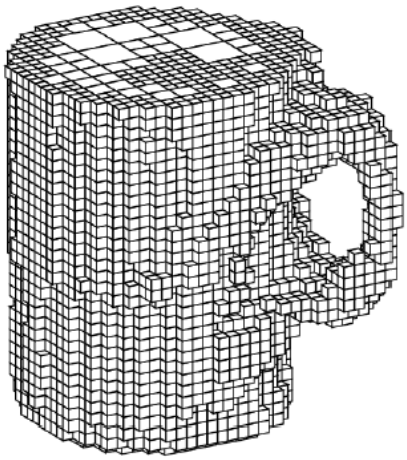


Figure 3.13: Voxel representation example.

3.3 Stereo Vision

Stereo vision is a passive 3D reconstruction method; it relies on simultaneously taking two photos with two different well-calibrated cameras separated by a baseline, obtaining two views of the same observed scene. The object's shape will be recovered through geometric relationships between the right and the left picture. This process takes inspiration from the human visual system, where images from the two eyes combine and provide information on the depth of the observed scene.

3.3.1 Epipolar geometry

Stereo Vision leverages the epipolar geometry to estimate the depth of the image, the task of inferring the distance of every point in a scene with respect to the camera.
Referring to Figure 3.14, the goal is to understand how far object X is from the camera. However, it is impossible to perceive depth with a single camera because other objects belonging to the same line of sight, such as X_1, X_2, or X_3, are projected into the same left image point X_L. Consequently, it is necessary to add another point of view, the right camera O_R, which, like the left camera O_L, points towards object X. The problem of estimating the depth turns into the correspondence problem: finding which pixel that belongs to the right image more looks like the X_L pixel. In this way, object X is triangulated, and its distance can be calculated. If the search for the pixel were to be done on the entire plane of

the right image, it would be an operation with a too expensive cost. Thanks to epipolar geometry, all the points belonging to the line of sight of X (X_1, X_2, X_3, ...) are projected on the right image in a particular area of the plane. All points belong to a specific line, the epipolar line (the red line in Figure 3.14). In this way, the search for correspondence becomes more straightforward, from a 2D search to a 1D one.

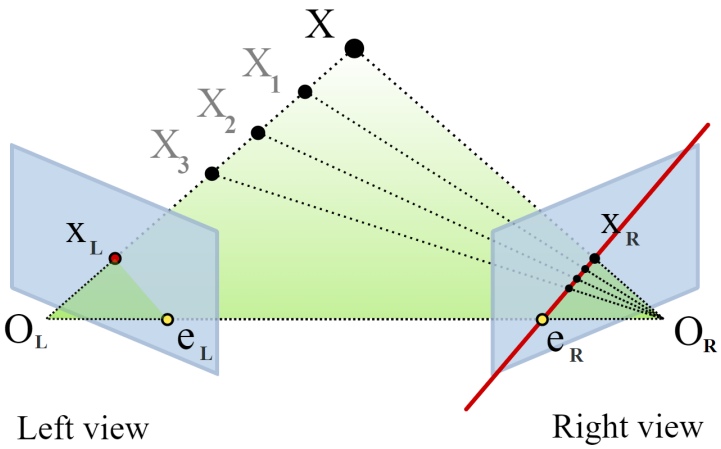


Figure 3.14: Epipolar geometry principle: all points belonging to the line of sight of X are projected into the same line of pixel of the right image, the epipolar line.

3.3.2 Steps in a Stereo Vision process

As illustrated in Figure 3.15, the Stereo Vision process can be divided into the following phases:

1. Calibration;
2. Rectification;
3. Stereo Correspondence;
4. Triangulation.

Calibration

Camera calibration is the process of relating the ideal camera model to the actual physical device (internal calibration) and retrieving the cameras' relative position and orientation (external calibration) [13].

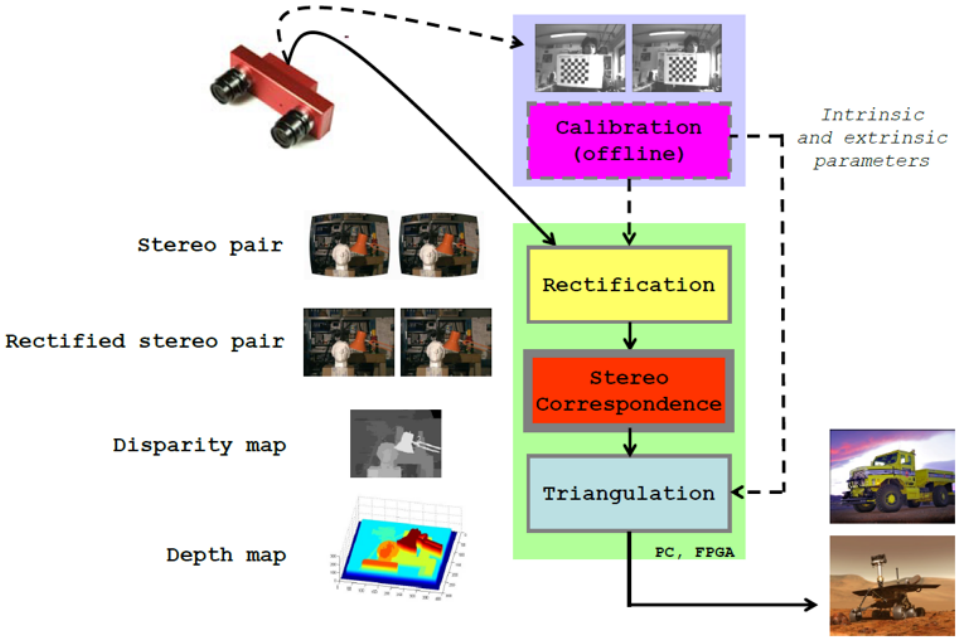


Figure 3.15: Steps in a Stereo Vision process [12].

"This is a fundamental step for 3D reconstruction and, in particular, for stereoscopic vision analysis. It allows not only for determining the geometry of the stereo setting needed for triangulation but also for removing radial distortions provided by common lenses" [13]. Moreover, both the correspondence problem and triangulation assume to deal with an ideal model of the camera (pinhole model).
As mentioned before, the objective of the camera calibration process is to find the camera calibration parameters, which can be divided into:

- Intrinsic (or internal) parameters, which describe the features of the camera, such as focal length, parameters of lenses distortion and image center;
- Extrinsic (or external) parameters, which determine the position and orientation of the camera with respect to a world reference system.

Usually, the calibration of a stereo camera consists of calibrating the two cameras separately and then applying the geometric transformations defined by the extrinsic parameters to understand the geometry of the stereo settings.
OpenCV and MATLAB provide several algorithms to calibrate the stereo rig, and they are based on acquiring and processing between 10 and 20 stereo images of a known pattern (typically a checkerboard).
The pinhole camera model and the mathematical aspects of the camera calibration process will be better investigated in Section 5.1.

Rectification

As explained before, the objective of stereo vision algorithms is to infer the distance of every point in a scene with respect to the camera.
In reference to Figure 3.16, the goal is to understand how far the P point is from the baseline of the stereo rig. In order to triangulate the point P, the problem of inferring the distance turns into the problem of searching on the right image (target image) the pixel that more looks like the pixel p (the projection of the object P on the left image plane) in the left image (reference image). Thanks to the epipolar geometry, the pixel looked for belongs to a particular line in the target image, the epipolar line. Once this information is known, the stereo rig can be virtually placed in a more convenient configuration, the standard form, in such a way that the two images are projected into a common image plane parallel to the line between optical centers, and corresponding points will belong to the same image scan line y.

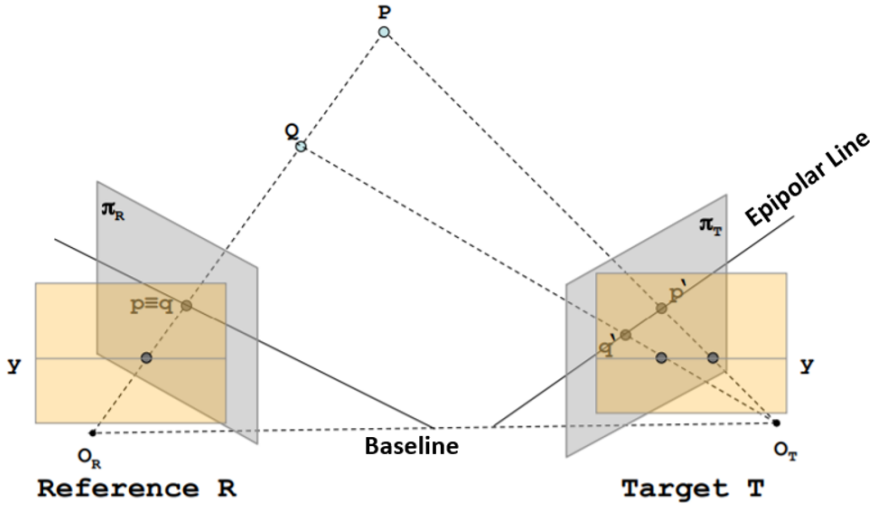


Figure 3.16: Stereo camera in the standard form [12].

Therefore, through the rectification process, epipolar lines of the images become parallel and horizontal, conjugated points will lie on the same line, and the search for the correspondent point in the other image can be limited only to points belonging to the same row of pixels (Figure 3.17). In this way, the correspondence problem becomes easier to solve.
Moreover, the rectification process can compensate for camera imperfections removing lens distortions.

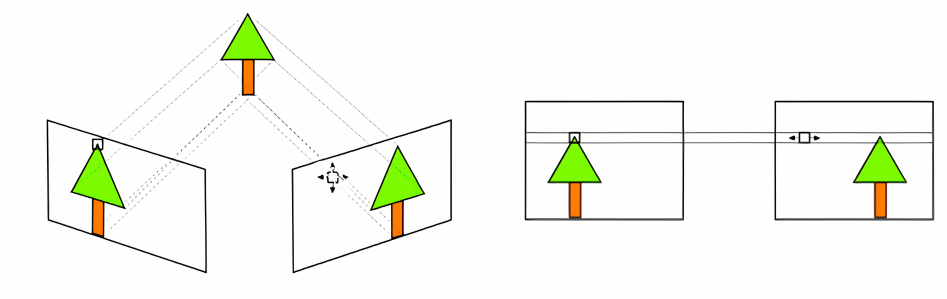


Figure 3.17: Images before and after the rectification process.

Stereo Correspondence

The correspondence problem involves finding, for each pixel of the reference image, the corresponding pixel in the target image.
The algorithms developed to solve the correspondence problem take the rectified image pair as input and produce a disparity map as output. The disparity map contains a disparity value for each pixel of the image. This value is simply the difference between the x coordinate of the analyzed pixel in the reference image x_R with the x coordinate of the corresponding pixel in the target image x_T. The disparity is typically encoded with a gray-scale image. Points near the camera are brighter and have a higher disparity; on the contrary, points far from the camera are darker and have a lower disparity (Figure 3.18).

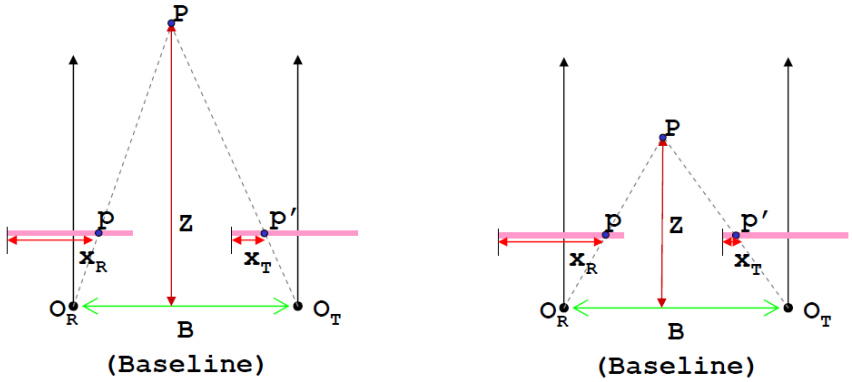


Figure 3.18: Disparity is higher for points closer to the camera.

To determine corresponding points, it is crucial to measure the similarity of those points. Taken any pixel in the reference image, the corresponding pixel in the target image will be the one that most looks like the first and belongs to the same image scanline.

Sometimes the search of conjugate points can be very challenging, and in the

following cases, it does not produce correct results:

- Specular surfaces: a non-Lambertian surface does not reflect light uniformly. Therefore, if the point of view changes, the surface looks different. Moreover, specular surfaces are mirror-like, so when they are illuminated, some bright spot of light appears on the surface.

- Occlusions and discontinuities: looking at the scene from different points of view, the visibility of the objects inside it changes.

- Uniform regions: the scene may contain objects with uniform textures, such as a white wall, or repetitive patterns, like in a chessboard. In these cases finding the conjugate points may be very tricky.

An example of these unlucky situations is shown in Figure 3.19.

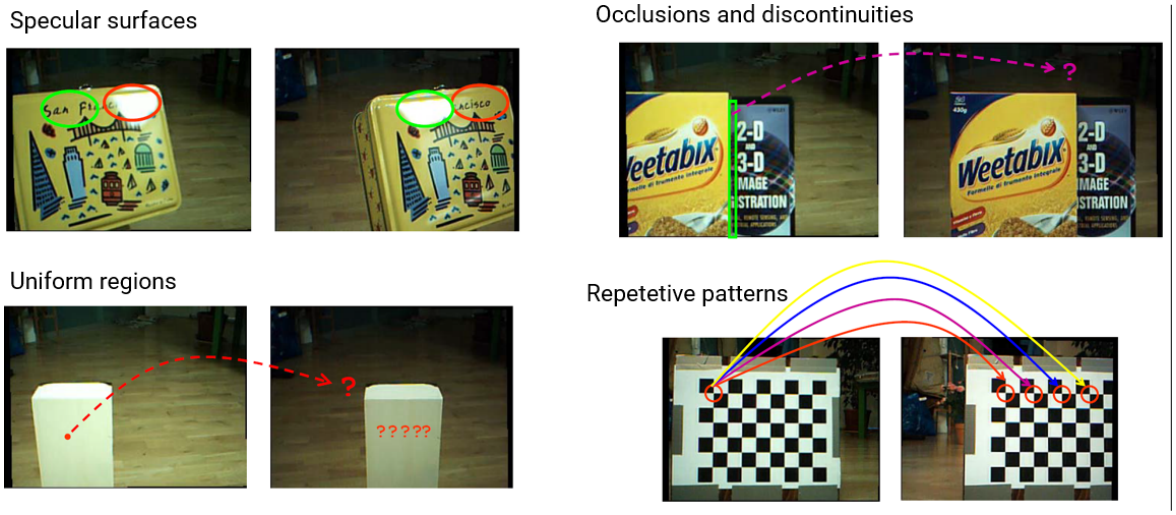


Figure 3.19: Difficulties in the correspondence problem [12].

Several stereo matching algorithms have been developed to limit one or more of these problems.
They can be divided into two general classes, depending on the way they assign disparities to pixels:

- Local algorithms: decide the disparity of each pixel according to the information provided by its local, neighboring pixels [14]. They do not produce particularly accurate disparity maps, but they are time efficient, do not require high memory resources, and consequently can be applied in real-time.

- Global algorithms: assign disparity values to each pixel depending on information derived from the whole image [14]. On the contrary, global methods are time-consuming but very accurate.

Moreover, stereo correspondence algorithms usually perform the following steps:

1. **Matching cost computation**
   For each pixel taken into consideration in the reference image, a "cost" is calculated. This value measures how much the pixel is different from the candidates in the target image. Examples of pixel-based matching cost functions are:

   - Absolute differences (AD)

   $$AD(x, y, d) = |I_R(x, y) - I_T(x + d, y)| \tag{3.6}$$

   - Squared differences (SD)

   $$SD(x, y, d) = (I_R(x, y) - I_T(x + d, y))^2 \tag{3.7}$$

   When costs are aggregated over a support region (window), the matching functions become:

   - Sum of Absolute differences (SAD)

   $$SAD(x, y, d) = \sum_{x \,\epsilon\, S} |I_R(x, y) - I_T(x + d, y)| \tag{3.8}$$

   - Sum of Squared differences (SSD)

   $$SSD(x, y, d) = \sum_{x \,\epsilon\, S} (I_R(x, y) - I_T(x + d, y))^2 \tag{3.9}$$

   Where S is the support window.

2. **Cost aggregation**
   Once costs have been defined for each point, they are aggregated. Usually, to each cost is added those in the surrounding of its pixel (fixed windows). Costs within the window are simply added up, or a weighted sum is applied where different weights are assigned to the central pixels.
   However, almost all state-of-the-art cost aggregation strategies assume that all the points belonging to the support share the same disparity [12]. Consequently, it is implicitly assumed that the surface is flat over the entire window and that there are no depth discontinuities within the support. Windows are usually square and small for performance reasons; nevertheless, other methods

like *adaptive algorithms* change the shape and size of the window based on the properties of the analyzed region. An ideal cost aggregation strategy should include in the support only points with similar disparities, reducing the support near discontinuities and expanding it near regions of similar depth [12].

3. **Disparity computation**
   Once a set of possible candidates for each pixel is built, a corresponding pixel is chosen, and the disparity is computed.
   Local methods usually use the "winner takes all" (WTA) selection, where the pixel with the lowest cost is selected as the corresponding pixel.

4. **Disparity refinement**
   Filters can be applied to improve the disparity map and, sometimes, a segmentation process is applied to obtain more defined edges.

Local methods are area-based, whereas global ones are energy-based. Local algorithms use the simple WTA disparity selection strategy and reduce ambiguity by aggregating matching costs over a support window. On the other hand, global algorithms search for disparity assignments that minimize an energy function which combines data and smoothness terms, considering the whole image [12]. The goal is to find the optimum disparity function $d = d(x, y)$ which minimizes a global cost function E:

$$E(d) = E_{data}(d) + \lambda \cdot E_{smooth}(d) \tag{3.10}$$

The smoothness hypothesis (no discontinuities in the analyzed region) is implicitly assumed by local approaches while it is explicitly modeled by the global ones.

In order to better understand the different steps of the stereo matching algorithms, it is possible to analyze the simplest local algorithm. This naive and unused algorithm is based on the following steps:

Algorithm Naive approach

```
1: for each epipolar line do
2:     for each each pixel in the left image do
3:         ▷ compare with every pixel on same epipolar line in right image
4:         ▷ pick pixel with minimum match cost
5:     end for
6: end for
```

To limit resources and increase performance, the disparity is not calculated for all physically possible values in the epipolar line but within a narrower range (Figure 3.20).

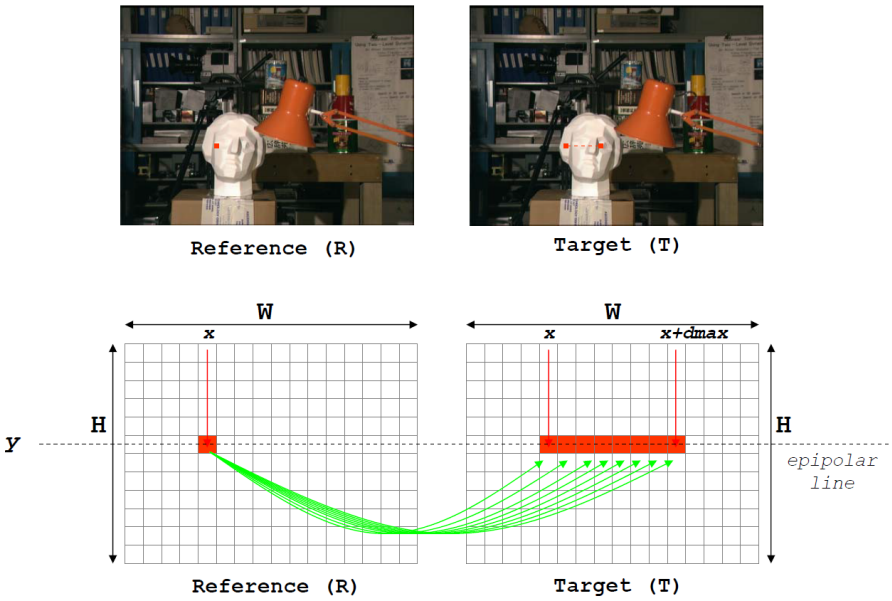


Figure 3.20: The simplest stereo matching algorithm [12].

As shown in Figure 3.21, the performed steps are:

1. Matching cost: is a simple pixel-based absolute difference between pixel intensities;
2. Cost aggregation: is not performed;
3. Disparity computation: is a winner takes all (WTA), so, the pixel that minimizes the matching cost will be considered as the corresponding pixel, and the disparity can be calculated.

Once the stereo matching phase is completed, the pixel taken into consideration is triangulated and its distance from the camera can be computed.

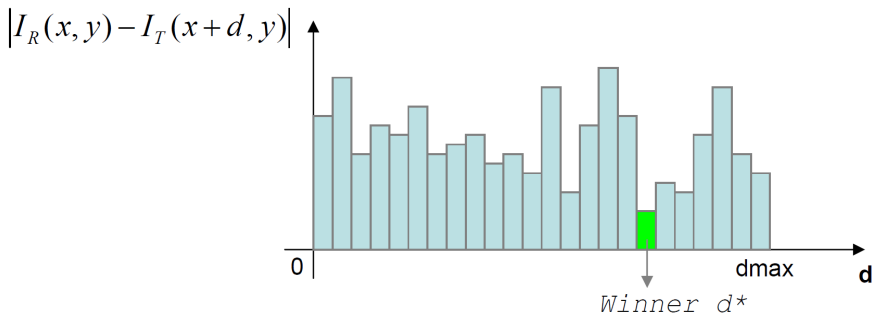


Figure 3.21: WTA selection with an absolute difference matching function [12].

Triangulation

As it is possible to see from Figure 3.18, the PO_RO_T triangle is similar to the Ppp' one. Consequently, it is possible to state that:

$$\frac{B}{Z} = \frac{(B + x_T) - x_R}{Z - f} \tag{3.11}$$

And finally:

$$Z = \frac{B \cdot f}{x_R - x_T} = \frac{B \cdot f}{d} \tag{3.12}$$

Similarly:

$$X = Z \cdot \frac{x_R}{f} \qquad Y = Z \cdot \frac{y_R}{f} \tag{3.13}$$

Where:

- $x_R - x_T$ is the disparity d;
- f is the focal length of the camera;
- B is the baseline of the stereo rig;
- Z is the depth of the point P, its distance from the. baseline

Therefore, given the disparity map, the focal length and the baseline of the stereo camera, the triangulation process allows computing the position of the correspondence in the 3D space (X, Y, Z). Repeating the procedure for each pixel of the disparity map, it is possible to obtain the corresponding depth map (Figure 3.22).

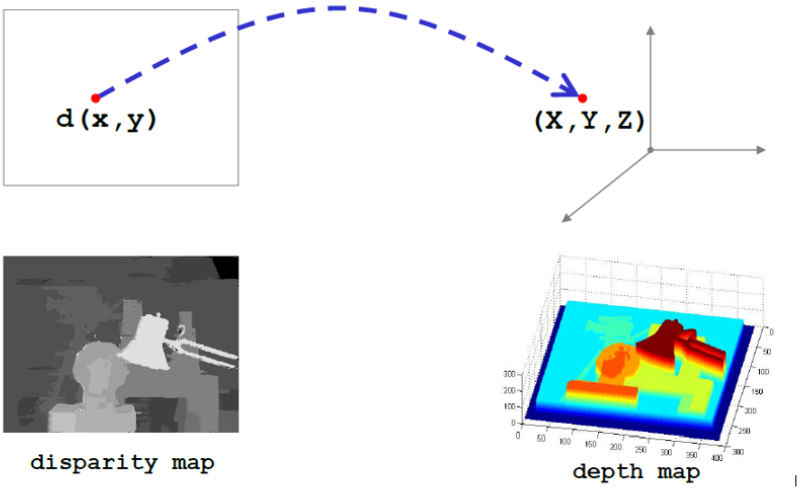


Figure 3.22: Triangulation process [12].

3.4 Monocular depth estimation

Monocular depth estimation (MDE) is the last analyzed passive reconstruction method and consists of inferring scene depth from a single RGB image.
The next chapter will be dedicated exclusively to the MDE approach. There will be a classification and a state-of-the-art analysis, with a relative explanation of the most used algorithms.

Chapter 4

Monocular Depth Estimation

Monocular depth estimation (MDE) is the task of inferring scene depth from a single RGB image. Clearly, it is an ill-posed problem since an infinite number of 3D scenes may have the same 2D projection in the image plane, obtaining the same RGB image.
The previously analyzed approaches require several observations of the scene of interest, either in the form of multiple viewpoints (Stereo Vision, Shape from Silhouette) or in the form of different lighting conditions (Shape from Shading). On the contrary, MDE exploits only one cue to predict the depth.
As explained before, scene depth can be perceived through at least two viewpoints of the scene. By employing only one camera, it is possible to exploit only one point of view; consequently, artificial intelligence algorithms must be used to overcome this limitation. Thanks to the rapid improvement of deep learning algorithms, the field of image processing has achieved extraordinary progress. Scientific research in the 3D reconstruction field has increasingly focused on MDE methods as they only involve using a single camera and do not require a complex camera calibration process.

4.1 Classification

Like classical machine learning techniques, depth estimation methods based on deep learning can be classified according to the degree of supervision assigned to the network during the training phase.
Deep learning MDE models can be divided in the following way:

- Supervised methods

- Unsupervised methods
- Semi-supervised methods

4.1.1 Supervised MDE

"These methods attempt to directly predict the depth of each pixel in an image using models that have been trained offline on large collections of ground truth depth data" [15]. In the scientific community, several large datasets have been created to meet the high demand for data, such as the *KITTI* dataset, which contains RGB-D depth maps for outdoor environments, or the *NYU-V2* dataset, which accounts for indoor environments. So, supervised MDE techniques require the creation of wide datasets where for each RGB image is also present the corresponding aligned depth map used as ground truth during the training. The objective is to penalize the network when the predicted depth map deviates from the ground truth through an appropriate loss function.
Since depth is a continuous quantity, the monocular depth estimation problem can be seen as a regression problem. The differences between the predicted and the correct depth map are utilized as loss functions to supervise the network's training. In particular, since this is a regression problem, it is usually employed a standard mean squared error (MSE) loss:

$$L_2(d, d^*) = \frac{1}{N} \sum_{i}^{N} \|d_i - d_i^*\|_2^2 \tag{4.1}$$

Where:

- d_i is the predicted depth value for the pixel i and d_i^* the corresponding ground truth value;
- N indicates the total number of pixel of the image.

Therefore, "depth networks learn the depth information of scenes by approximating the ground truth" [16].
However, the disadvantage of this method is the need to build a dataset with depth images acquired by a high-level depth sensor. As explained above, there exist datasets for indoor and outdoor environments. Nevertheless, if the scene is different from those present in the existing datasets, it is necessary to acquire the data personally and build a specific dataset for the application to obtain better results in estimating depth.

4.1.2 Unsupervised MDE

Since one of the biggest challenges of deep learning is the lack of enough datasets with ground truth, especially in the field of MDE, a possible solution is to approach the estimation problem in an unsupervised way.
In the unsupervised MDE, the data scarcity problem is no longer present since the dataset used for training the network does not require depth images but only monocular video.
Like this, the geometric constraints between video frames are regarded as the supervisory signal during the training process instead of using ground truths, which are expensive to acquire [16].
In particular, the training process consists of synthesizing a frame (target frame) from the point of view of a neighboring frame (source frame).

In order to better understand this concept, it is possible to consider the frame at the time instant $n-1$ as the source frame and frame n as the target frame.
The objective is to obtain a synthesized frame $\hat{I}_n$ starting from the target frame I_n such that the difference between the real and the synthesized frame is minimized. So, this difference will be used as loss function to supervise the training:

$$L_{vs} = \frac{1}{N} \sum_{p}^{N} \left| I_n(p) - \hat{I}_n(p) \right| \tag{4.2}$$

Where:

- N indicates the total number of pixels p in the image;
- $I_n(p)$ represents a pixel p of the target view I_n;
- $\hat{I}_n(p)$ is a pixel p of the reconstructed target view $\hat{I}_n$.

This problem is also known as novel view synthesis: given one input view of a scene, synthesize a new image of the scene seen from a different camera pose [17]. However, to synthesize a target frame from a neighboring frame, it is necessary to know the geometric transformations between the two frames in consecutive time instants. Consequently, in addition to predicting depth, the network has also to estimate the camera pose between frames. "This estimated camera pose is only needed during training to help constrain the depth estimation network" [18].

The general framework of the unsupervised MDE is illustrated in Figure 4.1. The depth network takes only the target view I_n as input, and outputs a per-pixel depth map $\hat{D}_n$. The pose network takes both the target view I_n and the nearby source views (I_{n-1} and I_{n+1}) as input, and outputs the relative camera poses (

$T_{n\to n-1}$ and $T_{n\to n+1}$).

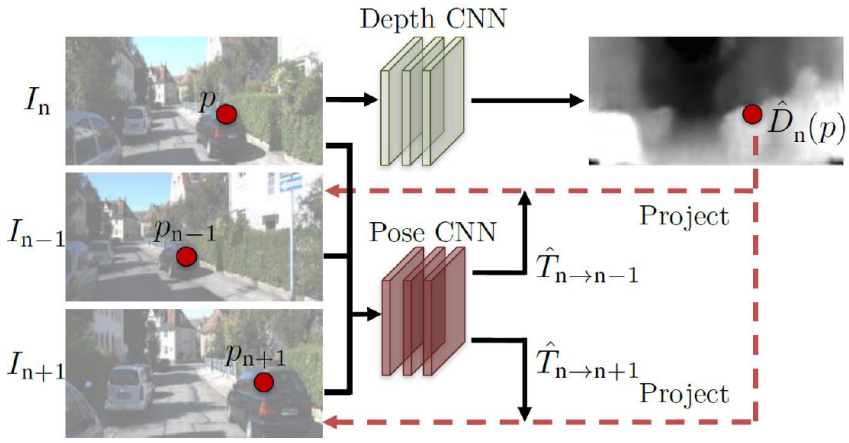


Figure 4.1: Unsupervised MDE: general framework [16].

As can be observed from Figure 4.2, the frame I_n (target frame) can be synthesized starting from the nearby frame I_{n-1} (source frame), and the reconstructed frame will be $\hat{I}_n$.

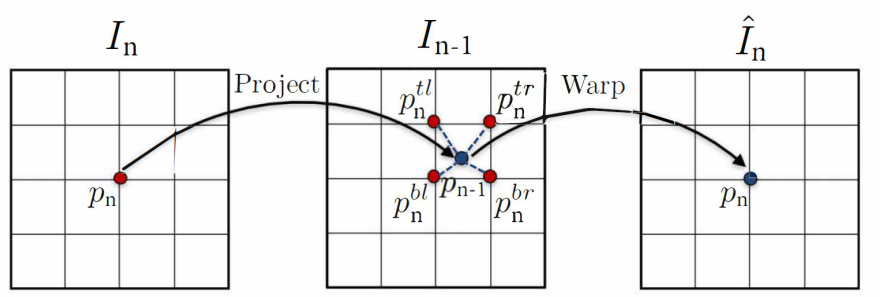


Figure 4.2: Image warping process [16].

There exist a relationship to understand, for each pixel of the target image I_n, what is the corresponding pixel in the source image I_{n-1}:

$$p_{n-1} \sim K\hat{T}_{n\to n-1}\hat{D}_n(p_n)K^{-1}p_n \tag{4.3}$$

Where :

- p_n is a pixel of the of the target image I_n, and p_{n-1} refers to the corresponding pixel in the source image I_{n-1};

- K indicates the camera intrinsic matrix, accounting for the camera intrinsic parameters;
- $\hat{D}_n(p_n)$ is the depth value of the pixel p_n, predicted from the depth network;
- $\hat{T}_{n \to n-1}$ represents the geometric transformation between I_n and I_{n-1}. It is the relative transformation matrix predicted by the pose estimation network.

Hence, once $\hat{D}_n$ and $\hat{T}_{n \to n-1}$ are estimated, the correspondence between pixels on different images (I_n and I_{n-1}) is established by projection function [16].

Once the projection relationships between I_n and I_{n-1} are established, then I_n is reconstructed by the image warping process, by sampling pixels from the source view I_{n-1}. In particular, the warping consists in a bilinear sampling mechanism that linearly interpolates the values of the four neighboring pixels (top-left, top-right, bottom-left, and bottom-right) [17].
The whole process is illustrated in Figure 4.2; each point p_t of the target view is projected onto the source view through the estimated depth and camera pose. After that, the point is reconstructed by bilinear interpolation.
Once the target view is synthesized, it is finally possible to compute the reconstruction loss to supervise the training (Equation 4.2). It is an L1 distance in the pixel space between the target and the reconstructed image, also known as photometric reconstruction error.
The overall goal is to predict a depth map such that the synthesized image minimizes the photometric error.

A possible problem of the unsupervised MDE occurs when the camera transformation matrices between frames ($T_{n \to n-1}$ and $T_{n \to n+1}$) are equal to zero.
This event, when it occurs, causes the presence of holes of infinite depth in the predicted depth map.
Hence, the estimation method fails when the assumption of moving camera and static scene is not respected. In this way, the only video that can be used during the network training are those in which there is sufficient motion between frames and which portray static scenes.
Godard *et al.* [18] solve the problem by introducing a simple auto-masking method that filters out pixels which do not change appearance from one frame to the next in the sequence, such that the loss function will not be contaminated.

4.1.3 Semi-supervised MDE

As explained before, it is hard to collect high-quality depth datasets accounting for all the possible background conditions. Semi-supervised MDE methods, as well as unsupervised ones, were developed to solve this problem.

Unlike unsupervised approaches, semi-supervised ones exploit rectified stereo images captured simultaneously to build the dataset used for training the network. However, the two methods are very similar. The main difference is that the geometric transformation between the two frames (left-right images) is known since the stereo camera has to be well-calibrated, so there is no need to insert a pose estimation network in the system.

As illustrated in Figure 4.3, the goal is to reconstruct the stereo pair's left image (target image) starting from the scene viewpoint of the right image (source image). In particular, the training process is developed through the following steps:

1. The left image I_L is taken as input;
2. The disparity map d (inverse depth map) of the left image is predicted through the depth estimation network;
3. The predicted inverse depth map is used to reconstruct the left image from the right image I_R by the inverse warping algorithm;
4. The error between the real image I_L and the synthesized one I_W is used to supervise the training:

$$L_{recons} = \sum_p \|I_L(p) - I_W(p)\|^2 = \sum_p \|I_L(p) - I_R(p + d(p))\|^2 \tag{4.4}$$

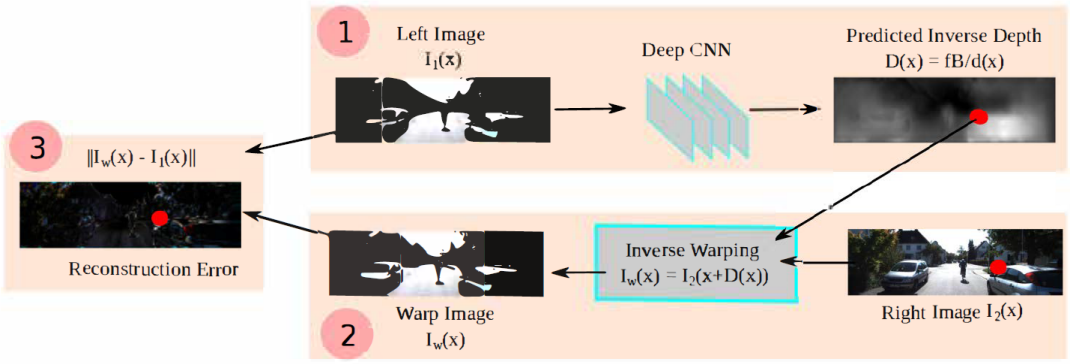


Figure 4.3: Semi-supervised MDE pipeline [16].

A significant improvement was achieved by Godard *et al.* [15], where the disparity map is computed both for the left image and the right image. Then the depth is calculated by considering both values. They introduce a novel training loss that emphasizes left-right depth consistency inside the network achieving better results on object boundaries.

4.1.4 Comparison between MDE methods

Since training the depth estimation network on stereo image pairs is very similar to the case of monocular video, it is important to point out that some studies regard both methods as either unsupervised or semi-supervised (also called self-supervised).
The most important thing to underline is that all three methods work in the same way during the inference phase: they take an RGB image as input and output a depth image. What differentiates them is the training phase: supervised methods need depth maps as ground truth, self-supervised ones require stereo images and unsupervised approaches involve collecting monocular video, enabling the gathering of training data through regular camera sensors.
Obviously, since there is no need for ground truth during training, the performance of unsupervised methods is still far from the supervised ones. Moreover, unsupervised methods also suffer from various problems, like scale ambiguity and scale inconsistency [16]. Semi-supervised methods are born to overcome these problems, to get higher estimation accuracy while reducing the dependence on the expensive ground truth. Besides, since the stereo camera must be well-calibrated, the scale information can be learned from the semi-supervised signals.
However, the unsupervised and self-supervised methods offer a better generalization, whereas the performance of the supervised ones depends a lot on the matching between the scenes present in the training dataset and those used in the inference phase.

4.2 Depth estimation network

The following section presents the most relevant architectures used as depth estimation networks. Specifically, it will be showed CNN-based structures employed mainly in the supervised monocular depth estimation problem. The exposition will be done chronologically, from the beginning methods (2014) to the state-of-the-art methods (2021).

4.2.1 State-of-the-art

Humans, over the years, have built mental models and stored information that allows them to perceive the depth information from a single image, exploiting features such as perspective, scale relative to known objects, appearance in lighting or occlusion, and more. Inspired by this, initial works perform single image depth estimation by combining some a-priori information, like the relationship between geometric structures, object sizes, and texture information [16].
With the popularity of deep learning, especially in the field of image processing,

some works try to solve the depth estimation problem using deep convolutional networks and achieved outstanding performance.
Eigen *et al.* [19] have been the first to exploit CNNs for regressing dense depth maps from a single image, expiring the modern trends of depth estimation. They introduce the approach of using multi-scale information and features captured at different levels of the deep neural network. Combining feature maps of different layers or aggregating them allows fusing information from multiple scales obtaining a better prediction of depth maps. They propose a two-scale architecture; the first stage estimates the global structure of the scene, the second stage refines the original prediction using local information [20]. The depth estimation network consists of the following blocks:

1. The Global Coarse-Scale Network: which produces a coarse depth map using a global view of the scene. The network focuses on important features such as vanishing points, object locations, and room alignment.

2. The Local Fine-Scale Network: which has the task of refining the coarse depth map produced from the first stage of the depth estimation network. The coarse prediction is aligned with local details such as objects and wall edges.

The architecture of the multi-scale network is illustrated in Figure 4.4.
The coarse stage contains five feature extraction layers of convolution and max pooling, followed by two fully connected layers. The resolution of the network output image is reduced by 1/4 with respect to the input image. This procedure is called down-sampling and is used to capture features at different levels.
The fine-scale network consists of convolutional layers with one pooling stage for the first layer, and a coarse feature concatenation to restore the resolution of the input image.

The important thing to remark is that the refinement from low spatial resolution to high spatial resolution is done in a network independent from the global coarse-scale one, which is trained separately at first.
Future works combine together the down-sampling stage with the recovering phase of the input resolution of the image (up-sampling) through an encoder-decoder network.
The classical framework of monocular depth estimation based on deep learning is an encoder-decoder network, with the RGB image taken as input and depth map produced as output, as shown in Figure 4.5.
The encoder part has to capture depth features, and it is usually composed of convolutional blocks and pooling layers. In this phase, the resolution of the input image is progressively decreased, through the convolution and pooling blocks, giving to the higher-level neurons large receptive fields, thus capturing more global

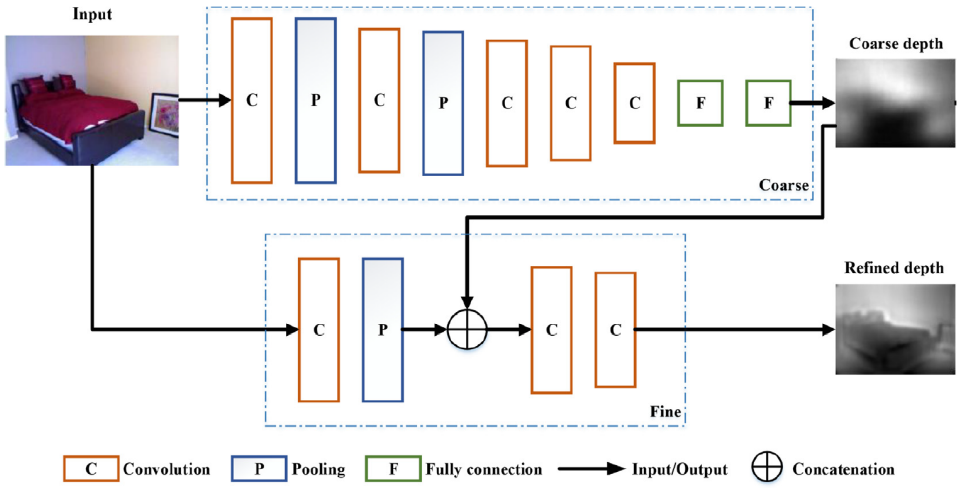


Figure 4.4: The architecture of the multi-scale network for MDE proposed by Eigen *et al.* [19].

information. Instead, the decoder network consists of deconvolutional layers, and it has to restore the estimated pixel-level depth map to the same spatial size of the input image. Additionally, in order to exploit features at each scale, the corresponding encoder and decoder layers are concatenated with skip-connections [21]. Skip-connections allow combining low-spatial resolution depth maps in deeper layers with high-spatial resolution depth map in lower layers.

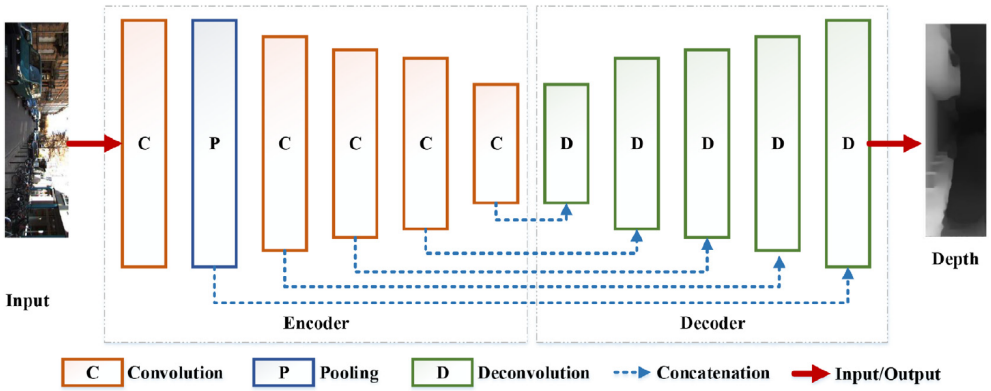


Figure 4.5: The general pipeline of the depth estimation network based on deep learning [21].

Laina *et al.* [22] are the first to adopt this kind of architecture, where the fully-connected layers used by [19], typically employed in classification networks,

are replaced by up-sampling blocks to improve the resolution of the predicted depth map, obtaining a fully-convolutional residual network. Moreover, fully-connected layers are very expensive with respect to the number of parameters. So, if the network allows input images with higher resolution (networks with higher receptive field) the fully-connected layer would cause the need for a very high number of parameters, and the network would be unfeasible with the current hardware.
In particular, they decide to employ *ResNet*, a deeper network than the ones employed in previous works, having a larger receptive field. By introducing skip-layers and batch normalization after every convolution it is possible build much deeper networks avoiding problems like vanishing gradient or degradation, which hamper convergence during training. Moreover, fully-connected layers in ResNet are replaced by their novel up-sampling blocks. In this way, the network contains fewer weights, and in addition, it generates more accurate depth maps.

Also Cao *et al.* [23] decide to follow the recent success of the deep residual network ResNet in a fully-convolutional manner (removing fully-connected layers). Previous existing algorithms formulate depth estimation as a regression problem due to the continuous property of the depth. However, it is very complex to regress the depth value of input data to be precisely the ground-truth value [23]. For humans, it is more difficult to estimate the exact distance of a specific object in an image than to give a rough distance range of that object. In general, it seems to be easier to estimate the depth range of a point rather than to estimate its exact depth value. Motivated by this, they formulate the depth estimation problem as a pixel-wise classification task by discretizing the continuous depth values into several discrete bins labeled according to their depth ranges [23]. So, the depth estimation network is trained to predict the depth range rather than the exact depth value.
Moreover, by reformulating the depth estimation problem as a classification problem, they can obtain a confidence level of the prediction. In a standard classification problem, if the predicted label is different from the ground truth label, the prediction is considered wrong and does not update the model parameters. In this type of classification problem, instead, if the predicted label is wrong but close to the ground-truth one and with a high level of confidence, it can still be used to update the model parameters.
Besides, the confidence level in the form of probability distribution is also useful for a potential post-processing. In particular, as shown in Figure 4.6, Cao *et al.* employ the fully-connected conditional random fields (CRF) through which pixel depth estimation with low confidence can be improved by other pixels that are connected to it. Thereby, the estimated depth maps are refined by considering the depth information of neighboring pixels, but the network cannot be trained in an end-to-end fashion.

The discretization of the depth space into several bins is also adopted by Fu *et al.* [24]. When the depth estimation task is addressed as a standard regression

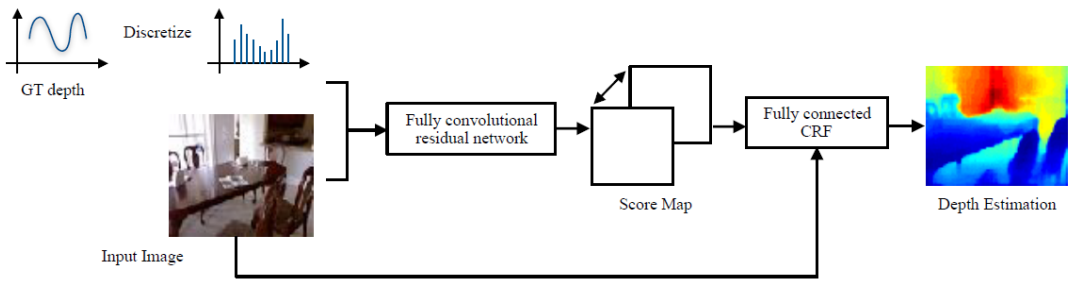


Figure 4.6: Depth estimation network as classification problem with post-processing [23].

problem, loss functions like mean squared error (MSE) are usually chosen. However, the minimization of MSE during training suffers from slow convergence and local optima solutions.
Moreover, previous works usually employ deep CNN in a fully-convolutional manner to extract features. In this stage, repeated pooling operations significantly reduce the spatial resolution of feature maps, which is an undesirable effect. To recover back the original resolution of the image and incorporate multi-scale information, networks combine higher-resolution feature maps through multi-scale networks [19], decoder networks [22], or skip-connections [19] [23], which complicate network training and consume much more computations.
Motivated by this, they decide to discretize continuous depth into several intervals and address the monocular depth estimation as an ordinal regression problem.
Since the uncertainty of the predicted depth values increases along with the ground truth depth values, it is convenient to allow larger estimation errors when predicting larger depth values. So, the training loss is down-weighted in regions with larger depth values to avoid that these predictions significantly influence the training process. To realize this kind of concept, they adopt a space-increasing discretization (SID) strategy instead of the classical uniform discretization (UD) strategy. In this way, as depth increases, the intervals become wider.
The designed architecture, as illustrated in Figure 4.7, basically consists of three main parts:

- The dense feature extractor, which is basically a deep CNN without the last few down-sampling operators that decreases the spatial resolution of depth maps. Specifically, down-sampling operators are replaced by subsequent dilatated convolutional layers to enlarge the field-of-view of filters without reducing the spatial resolution or increasing the number of parameters.

- The scene understanding modular, which is composed of three parallel components:
  - An atrous spatial pyramid pooling (ASPP) module, which extracts features from multiple large receptive fields through three dilated convolutional layers (with dilatation rates 6, 12 and 18, respectively);
  - A 1 x 1 convolutional branch to learn complex cross-channel interactions;
  - A full-image encoder that captures global features, similar to the ones seen in previous works but with fewer parameters.

  Subsequently, the obtained features are concatenated to achieve a comprehensive understanding of the input image. A 1x1 convolutional layer is inserted at the end to reduce the dimension of the features and learn complex cross-channel interactions.

- An ordinal regression optimizer that transforms features into multi-channel dense ordinal labels.

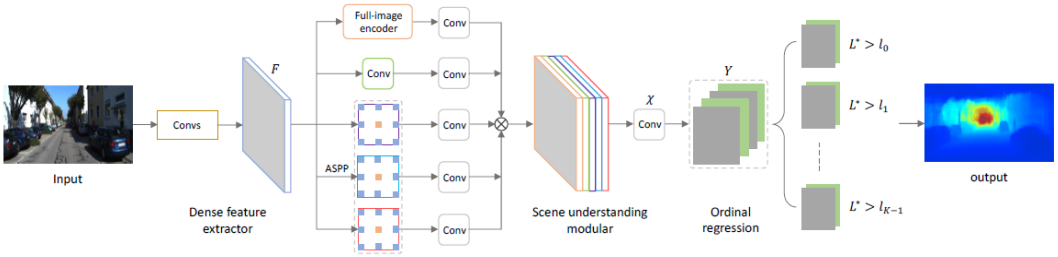


Figure 4.7: The depth estimation architecture proposed by Fu *et al.* [24].

Lee *et al.* [25] take up the latest architecture presented but apply some changes in the decoding phase. The main idea of their work is to define direct and explicit relations in recovering back the full resolution of the image for more effective guidance of densely encoded features to the desired depth prediction. The initial structure composed by the dense feature extractor is maintained but in the decoding phase, where the resolution of the input image has to be restored, up-sampling layers and skip connections are replaced by novel local planar guidance layers which guide features to the full resolution with the local planar assumption.
Fu *et al.* [24], in all the decoding phase, work with a fixed resolution of the image (1/8 of the input image resolution) and, at the end, recover the final resolution through a simple up-sampling operation. Instead, Lee *et al.* [25] work with different spatial resolutions of the image (1/8, 1/4, and 1/2 of the input image) and, at

each decoding stage, a local planar guidance layer is placed to recover the final resolution of the image, as can be observed in Figure. So, the main difference with [24] is that the final resolution of the image H is obtained at each decoding stage, not only at the end, and all these predictions are combined together in full resolution. This process allows a direct relation between internal features and the final prediction. The final architecture is composed by:

- A dense feature extractor that reduces feature map resolution to $H/8$;
- A denser version of atrous spatial pyramid pooling layer (ASPP) with various dilation rates (3,6,12,18,24) as contextual information extractor;
- The decoding phase previously explained.

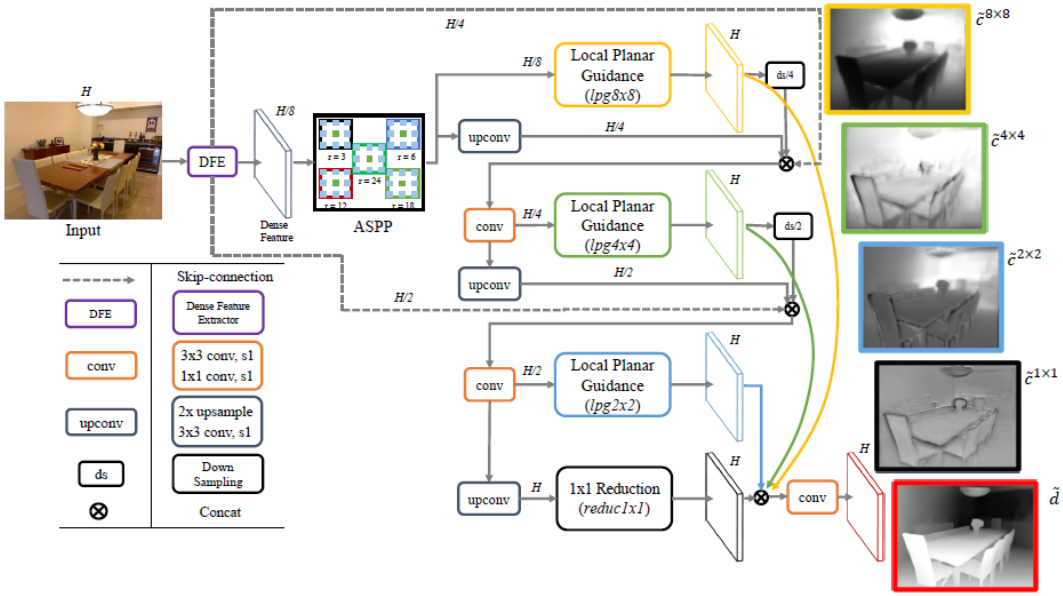


Figure 4.8: The depth estimation architecture proposed by Lee *et al.* [25].

4.2.2 Training Loss

For a depth map with N pixels, where d_i indicates the depth value of the pixel i and d_i^* is the corresponding ground-truth depth value, the most used loss functions for the supervised monocular depth estimation task are the following:

- L1 Loss

$$L_1(d, d^*) = \frac{1}{N} \sum_{i=1}^{N} \|d_i - d_i^*\|_1 \tag{4.5}$$

- L2 Loss (mean squared error)

$$L_2(d, d^*) = \frac{1}{N}\sum_{i=1}^{N} \|d_i - d_i^*\|_2^2 \tag{4.6}$$

- Scale-invariant mean squared error

$$L(d, d^*) = \frac{1}{N}\sum_{i=1}^{N} y_i^2 - \frac{\lambda}{N}(\sum_{i=1}^{N} y_i)^2 \tag{4.7}$$

  where $y_i^2 = log(d_i) - log(d_i^*)$, and λ is a balance factor.
  It is introduced by Eigen *et al.* [19] and it measures the relationship between points in the scene, irrespective of the absolute global scale.

- Berhu Loss

$$L_{Berhu}(d, d^*) = \begin{cases} |d - d^*| & if \ |d - d^*| \leq c \\ \frac{|d-d^*|^2+c^2}{2c} & if \ |d - d^*| > c \end{cases} \tag{4.8}$$

  where c is a threshold.
  It is introduced by Laina *et al.* [22].
  If $|d - d^*| \leq c$ the Berhu loss is equal to L1, otherwise, if $|d - d^*|$ is outside the range, the Berhu function is equal to L2.
  It puts high weight towards samples (pixels) with a high residual because of the L2 term, and at the same time, L1 accounts for a greater impact of smaller residuals than L2 would.

4.3 Datasets and evaluation metrics

This section introduces some common datasets and evaluation metrics used in deep learning methods for monocular deep estimation, especially for the supervised training.

4.3.1 Datasets

There exist various datasets for supervised monocular depth estimation. Some of these can also be used for unsupervised and semi-supervised monocular depth estimation, if they collect RGB images through monocular video or with a stereo camera.
The objective of the supervised training is to estimate depth maps that approximate ground truth depth images as closely as possible. For this reason, the presented datasets consist of both RGB and depth images. Specifically, each RGB image is associated with the corresponding aligned depth map, which is used as ground-truth

during the training process.
They differ in several aspects, such as the captured environment (indoor and outdoor scenes), the depth range of the images, the depth map accuracy, the camera settings and the dataset size.
The most used datasets are certainly NYU Depth V2 for indoor environments and KITTI for outdoor environments.

NYU Depth V2

The NYU Depth dataset is provided by Silbereman *et al.* at the New York University. It is composed of 464 indoor scenes captured by a Microsoft Kinect camera (RGB-D camera). These indoor scenes are split into 249 ones for training and 215 ones for testing. It is mostly used for supervised monocular depth estimation. The depth of the dataset ranges from 0.5 m to 10 m.
The images are acquired through monocular video, and in total, the dataset contains 407024 video frames with a resolution of 640 x 480 pixels.
The RGB-D camera is composed of an RGB sensor and a depth camera, respectively acquiring RGB and depth images. Since the depth and RGB cameras operate at different variable frame rates, there is no one-to-one correspondence between depth maps and RGB images. So, each depth image is associated with its closest RGB image in time. Then, the camera projections provided by the dataset are used to align depth and RGB pairs. Frames, where one RGB image is associated with more than one depth, are not considered, obtaining a total of 1449 aligned RGB and depth images. These last selected images are pre-processed: missing depth values, always present in the images acquired by the depth camera, are filled with a colorization algorithm and are manually labeled with the semantic information.
In summary, the dataset is made up of:

- The Raw Dataset: 407024 unlabeled frames directly coming from the Microsoft Kinect camera, without any processing;
- The Labeled Dataset: 1449 densely labeled pairs of aligned RGB and depth images pre-processed to fill in missing depth labels. It is a subset of the Raw Dataset.

KITTI

The KITTI dataset is composed of 56 outdoor scenes, and in particular, it includes five categories of environments: "Road", "City", "Residential", "Campus", and "Person". These outdoor scenes are split into 28 ones for training and 28 ones for testing. The maximum measuring distance is 120 m. It is the largest and most commonly used dataset, especially in the unsupervised and semi-supervised

monocular depth estimation, but also in other tasks like visual odometry, object detection, and tracking.
Unlike the NYU Depth V2 dataset, which collects ground-truths with an RGB-D camera, the KITTI dataset takes sparse depth maps with a rotating LIDAR scanner. RGB images instead are acquired by a stereo pair with e a resolution of 1224 x 368. All the scenes are captured through a moving car equipped with RGB cameras, the LIDAR depth sensor, and a GPS localization system.

Make3D

Make3D dataset is another outdoor dataset. It is provided by Saxena *et al.* in Stanford University and includes daytime city and natural scenery. It contains a total of 534 depth images, 400 of which are used for training, and 134 are used for testing.
The depth ranges from 5 m to 81 m, and depth maps are collected through a laser scanner. RGB images are acquired with a resolution of 2272 x 1704. They are not in the form of stereo images or monocular video frames, so, this dataset cannot be applied for unsupervised or semi-supervised monocular depth estimation.

4.3.2 Evaluation metrics

Evaluation metrics are tools used to evaluate the performance of a model, hence the accuracy of the prediction.
During the training phase, a specific loss function has to be minimized, whereas, during the testing phase, the results obtained through the training process are evaluated with the following metrics.
The most common indicators are:

- Root Mean Squared Error

$$RMSE = \sqrt{\frac{1}{|N|} \sum_{i \in N} \|d_i - d_i^*\|^2} \tag{4.9}$$

- Root Mean Squared Logarithmic Error

$$RMSLE = \sqrt{\frac{1}{|N|} \sum_{i \in N} \|log(d_i) - log(d_i^*)\|^2} \tag{4.10}$$

- Absolute Relative Error

$$AbsRel = \frac{1}{|N|} \sum_{i \in N} \frac{|d_i - d_i^*|}{d_i^*} \tag{4.11}$$

- Squared Relative Error

$$SqRel = \frac{1}{|N|} \sum_{i \in N} \frac{\|d_i - d_i^*\|^2}{d_i^*} \tag{4.12}$$

- Threshold Accuracy

$$Accuracy = \%\ of\ d_i\ s.t.\ max(\frac{d_i}{d_i^*}, \frac{d_i^*}{d_i}) = \delta < thr \tag{4.13}$$

Where N indicates the total number of pixels in the depth image, d_i is the predicted depth value of the pixel i, and d_i^* the corresponding ground-truth depth value.
For the accuracy rate metrics, the only one for which bigger values are better, the term thr denotes a predefined threshold. Usually it is used $\delta < 1.25^n$ where $n = 1,2,3$.

Chapter 5

Volume Estimation Algorithm

This thesis project aims to design a system for estimating the volume of objects placed inside the smart bin Nando using only a simple RGB camera. Taking advantage of the recent progress in the field of depth estimation achieved by deep learning methods, it is possible, starting from a single RGB image of the object captured inside the bin, to predict the corresponding depth image.
The following chapter initially introduces the ideal model of the camera used to map real-world quantities into camera sensor quantities. Subsequently, the camera model is applied to transform the number of pixels covered by the object in the image into world units. Then, two estimation strategies are presented and compared to calculate the volume of an object starting from its depth image. Finally, volume estimation algorithms are tested through different procedures.

5.1 Pinhole camera model

Human vision begins when a source emits light rays that strike a particular object. Some of the rays are absorbed, the others are reflected. The reflected light rays hit the eye, go through the pupil and then project onto the retina, where the image is reduced and upside down. Finally, the image is processed by the brain, which straightens the image. A simple model of how this happens is the pinhole camera model.
The pinhole camera can be seen as a barrier with a small hole in the center. Only light rays that pass through the tiny hole hit the photographic film and are projected onto the image plane, as it is shown in Figure 5.1.

Where:

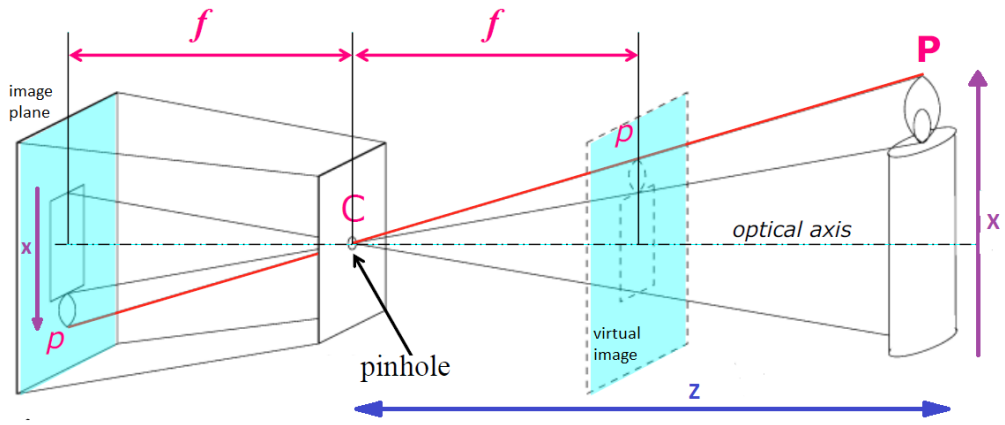


Figure 5.1: Pinhole camera working principle.

- f is the focal length of the camera and represents the distance between the image plane (also called screen, sensor, film) and the pinhole aperture C that behaves as the center of projection;
- X indicates the length of the object in the real world, and x is the corresponding length on the image plane;
- Z denotes the distance of the object from the camera.

Moreover, the intersection between the optical axis and the image plane is called principal point.
As can be observed form Figure 5.1, thanks to the presence of similar triangles, the following relation holds: $-x/f = X/Z$.
Usually, it is common to consider the virtual image plane instead of the real one. Since it is geometrically equivalent to consider the real image (inverted) or the virtual one (not inverted), symmetrical with respect to the pinhole, it is possible to state that:

$$x = f\frac{X}{Z} \tag{5.1}$$

And similarly:

$$y = f\frac{Y}{Z} \tag{5.2}$$

The minus sign is no longer present since the image on the virtual plane is no longer upside down.

The objective of the pinhole camera model is to describe the mapping between coordinates of a point in 3D space and its projection onto the image plane.
The problem, as it is shown in Figure 5.2, can be divided as follows [26]:

- World coordinates are transformed into camera coordinates through the extrinsic matrix (rigid transformation);
- Camera coordinates are mapped into the image plane through the camera intrinsic matrix (projective transformation).

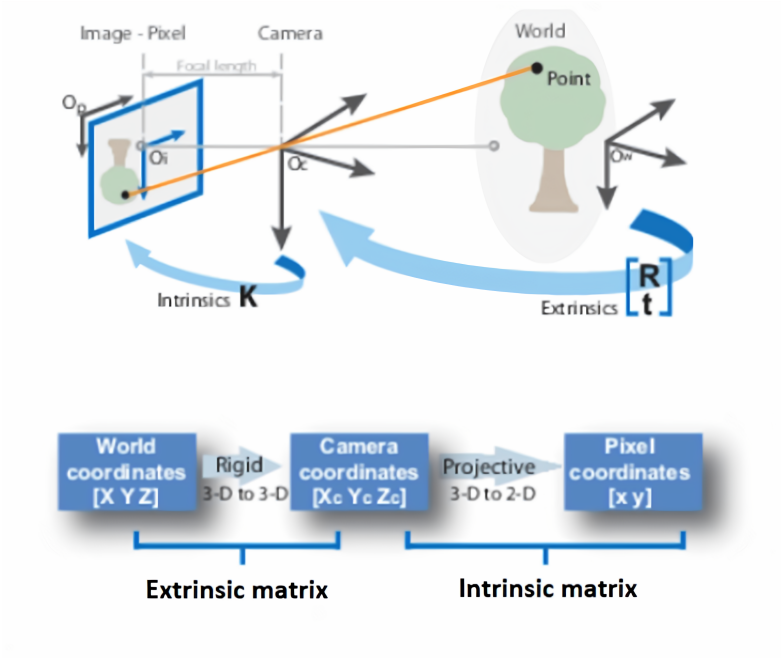


Figure 5.2: Pinhole camera model pipeline [26].

Intrinsic camera matrix

The goal is to compute the matrix connecting the camera points to the image points. In Figure 5.3 are illustrated all the reference systems that need to be considered for this problem. The camera reference system $CXYZ$ is positioned in the pinhole aperture with the Z axis along the optical axis, pointing towards the image plane and intersecting with the principal point. The image reference system $Oxyz$ has its origin on the principal point, and its z axis coincides with the Z axis of the camera reference system.

By means of Equations 5.1 and 5.2, 3D points expressed with respect to the camera reference system are mapped on the image plane through the following

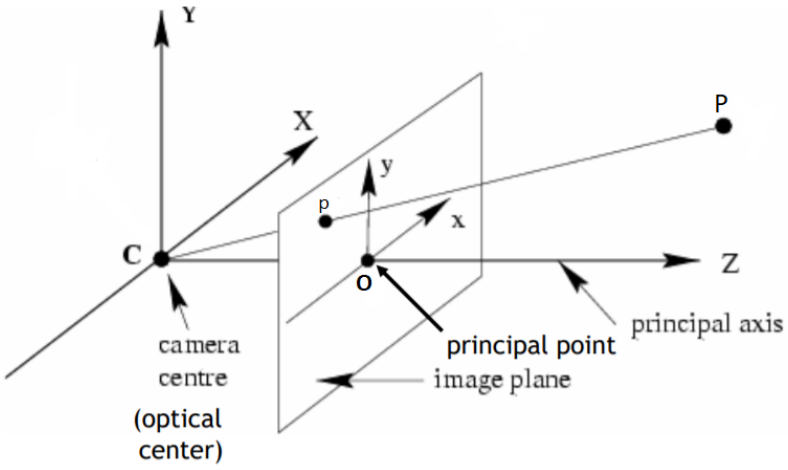


Figure 5.3: Reference systems in the pinhole camera model.

relationship:

$$(X_C, Y_C, Z_C) \rightarrow (f\,\frac{X_C}{Z_C}, f\,\frac{Y_C}{Z_C}) \tag{5.3}$$

However, in most cameras, the pixel with coordinates (0,0) is at the top left of the image and not at the center. Thus, 2D points in the image plane are offset by a translation vector $[c_x, c_y]$. Moreover, pixels on a typical low-cost imager are rectangular rather than square, so two focal length values are present.
The result is that a point P_C in the real world with coordinates (X_C, Y_C, Z_C) with respect to the camera reference system, is projected onto the screen at some pixel location given by (x, y), according to the following equations [27]:

$$\begin{aligned} x &= f_x \left(\frac{X_C}{Z_C}\right) + c_x \\ y &= f_y \left(\frac{Y_C}{Z_C}\right) + c_y \end{aligned} \tag{5.4}$$

As it is possible to notice, the function mapping real-world points onto the screen is nonlinear due to the division by Z. A possible solution to the problem is to transform the euclidean coordinates into homogeneous coordinates. In homogeneous coordinates, the space is augmented introducing a new coordinate. Point $p = (x, y)$ becomes $(x, y,1)$ and similarly $P_C = (X_C, Y_C, Z_C)$ becomes $(X_C, Y_C, Z_C,1)$. Moreover, all points having proportional values in the projective space are equivalent,

so the point p can be expressed in homogeneous coordinates as follows:

$$p = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x \left(\frac{X_C}{Z_C}\right) + c_x \\ f_y \left(\frac{Y_C}{Z_C}\right) + c_y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x X_C + c_x Z_C \\ f_y Y_C + c_y Z_C \\ Z_C \end{bmatrix} \tag{5.5}$$

3D points in camera coordinates are mapped into the image plane as follows:

$$p = \begin{bmatrix} f_x X + c_x Z \\ f_y Y + c_y Z \\ Z \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix} \longrightarrow p = MP_C \tag{5.6}$$

Matrix M can be decomposed as:

$$M = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} K & 0 \end{bmatrix} = K \begin{bmatrix} I & 0 \end{bmatrix} \tag{5.7}$$

where I is a 3×3 identity matrix and K is the camera intrinsic matrix:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.8}$$

Extrinsic camera matrix

The objective is to relate points from an arbitrary world reference system to the camera reference system. This transformation can be achieved by a simple rotation matrix R and a translation vector T.
Therefore, given a point in a world reference system $P = [X, Y, Z]^T$, its camera coordinates $P_C = [X_C, Y_C, Z_C]^T$ can be computed as follows:

$$P_C = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} P \tag{5.9}$$

where the term $[R, T; 0,1]$ is the 4×4 rototranslation matrix.
Substituting this relationship into Equation 5.6, it is possible to find the final mapping between coordinates of a point in 3D space and its projection onto the image plane:

$$p = MP_C = K \begin{bmatrix} I & 0 \end{bmatrix} P_C = K \begin{bmatrix} R & T \end{bmatrix} P = CP \tag{5.10}$$

Where:

$$C = K \begin{bmatrix} R & T \end{bmatrix} \tag{5.11}$$

is the final 3×4 camera matrix linking intrinsic to extrinsic parameters.

5.1.1 Lens distortions

However, the pinhole camera model is an ideal camera model that can only be used as a first approximation of the mapping from a 3D scene to a 2D image.
Due to the very narrow aperture of the pinhole camera, not enough light is collected to take the picture quickly. Real cameras manage to collect enough light through the use of a lens. The lens gathers light rays and bends them so that they all converge at the same point. However, lenses introduce distortions, in particular, radial and tangential distortions.
The cheaper the camera is used, the more its behavior differs from the pinhole model.
Radial distortion arises from the shape of the lens and occurs when light rays bend more near the edges of a lens than they do at its optical center. The effects of radial distortion can be seen in Figure 5.4, and it can be modeled as follows:

$$\begin{aligned} x_D &= x\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \\ y_D &= y\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \end{aligned} \tag{5.12}$$

where (x_D, y_D) are the real coordinates of a point in the image (distorted), (x, y) are the ideal coordinates (not distorted), $r^2 = x^2 + y^2$ represents the distance of the point from the center of the image and k_1, k_2, k_3 are the the radial distortion coefficients of the lens.

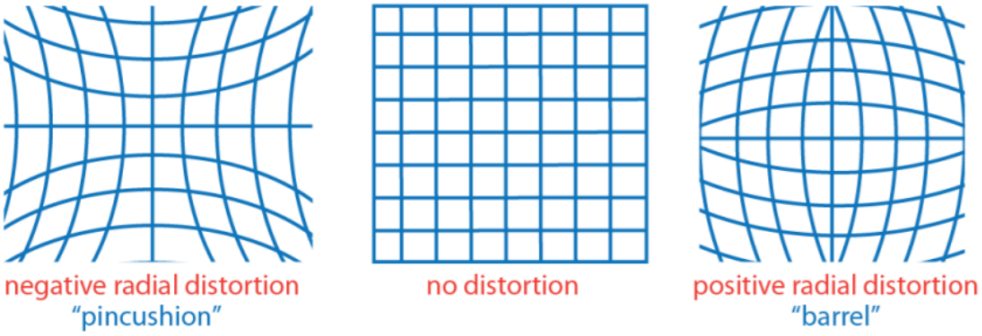


Figure 5.4: Radial distortion introduced by the lens [26].

Tangential distortion, instead, arises from the assembly process of the camera and occurs when the lens and the image plane are not exactly parallel. It can be characterized as follows:

$$\begin{aligned} x_D &= x + \left[2p_1 y + p_2(r^2 + 2x^2)\right] \\ y_D &= y + \left[p_1(r^2 + 2y^2) + 2p_2 x\right] \end{aligned} \tag{5.13}$$

where p_1, p_2 are the coefficients accounting for the tangential distortion, and the other quantities are the same described above.
Taking into account also the distortion effects, the camera intrinsic matrix becomes more complicated.

5.1.2 Camera calibration

The camera calibration process aims to estimate the extrinsic and intrinsic camera parameters, therefore the camera matrix of the Equation 5.11. It consists in pointing the camera towards a known structure with many identifiable points, like a chessboard. By viewing this structure from different angles, it is possible to compute the (relative) location and orientation of the camera at the time of each image as well as the intrinsic and lens parameters of the camera [27].
The knowledge of these parameters is important for several reasons; they can be used, for example, to correct the camera's behavior from the distortions introduced by the lens or to estimate the size of an object in world units.

5.2 Volume estimation

In this section, firstly, the strategy used to recover the object's height from the depth map is deeply analyzed. Then, two different volume estimation formulas to be applied to the reconstructed object are compared. Finally, the pinhole camera model is employed to transform the number of pixels covered by the object in the image along the x and y direction into world units.

5.2.1 Object reconstruction strategy

The strategy developed to reconstruct the object within the depth map is simple and essentially consists of three fundamental steps. The following reasoning is done assuming:

- To have a neural network capable of estimating depth from a single RGB image. The network takes the RGB image as input and produces a depth map as output, where each pixel of the depth image contains the distance in meters between the camera and the scene depicted in that pixel.
- To take into account static scenes with a fixed working environment.
  In particular, the considered working environment is the scenario inside the bin, and it is based on:
  - A stationary camera with a top-view of the object;
  - A fixed background that is the same in all the photos taken by Nando

Therefore, the distance between the camera and the background is always constant. The only thing that changes from one photo to another is the object that is present inside it.

The basic idea used to isolate the object within the image is shown in Figure 5.5 and is based on the following three steps:

1. Take a picture of the background and estimate the depth.
   If a flat background is considered, such as a table, and the distance between the table and camera is 10 meters, the depth map produced by the network should contain a value of 10 in each pixel.
   The depth image just described is the "background depth image" illustrated in Figure 5.5.

2. Take a photo of the object whose volume has to be estimated, positioned above the background previously selected, and estimate the depth.
   Supposing to estimate the volume of a cuboid object with a height of 2 meters, the computed depth map should contain:

   - A value of 10 in each pixel not covered by the object; since the background is at a distance of 10 meters from the camera.
   - A value of 8 in each pixel covered by the object; since the object will be 10 minus 2 meters away from the camera, where 2 meters represents the object's height.

   The depth image obtained in this step will be called the "back+obj depth image", as illustrated in Figure 5.5.

3. Make the difference between the "background depth image" and the "back+obj depth image".
   Therefore, to each depth value in the background depth image is subtracted the corresponding depth value in the "back+obj depth image". In the following way, the resulting depth map should contain:

   - A value of 0 in all the pixels that have not changed depth once the object was inserted into the background;
   - A value of 2 (the object's height in meters) in all the pixels that have changed depth once the object was inserted into the background.

   The depth image resulting from the subtraction operation will be called the "object depth image", as shown in Figure 5.5.

Thanks to the object reconstruction strategy, it is possible to estimate the object's height in each pixel covered by the object in the image.

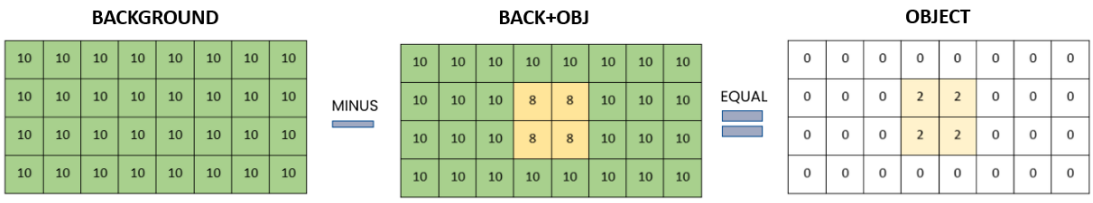


Figure 5.5: Object reconstruction strategy.

5.2.2 Volume estimation algorithms

Once the object is reconstructed within the image, its volume can be computed by adding the product between each depth value in the "object depth map", which represents the object's height in the various points of the image, and the area of the corresponding pixel:

$$V = \sum_{i=1}^{N} (d_i A_i) \tag{5.14}$$

where:

- V represents the volume of the object;
- d_i is the depth value of the pixel i in the "object" depth map (the object's height);
- N indicates the total number of pixels in the image;
- A_i denotes the area of the pixel i expressed in world units.

In the particular example of the cuboid object (Figure 5.5) the volume will be:

$$V = 2A_i + 2A_i + 2A_i + 2A_i \tag{5.15}$$

To verify the correctness of the first proposed method, another possible approach is developed to estimate the volume of an object.
In order to compare the two techniques, it is possible to consider the same example of the cuboid object presented before, but in 2D. So, instead of the volume, the area has to be estimated. In particular, the second row of the "object" depth map is taken into account, and it is converted into an xz profile, as shown in Figure 5.6. Where x_i terms represent the pixels of the examined row with fixed length L, and $z = f(x)$ values indicate the height of the object.

The true area defined under the graph is the integral of the profile and is equal to:

$$A_{TRUE} = \int_{x_1}^{x_8} f(x)dx = 4L \tag{5.16}$$

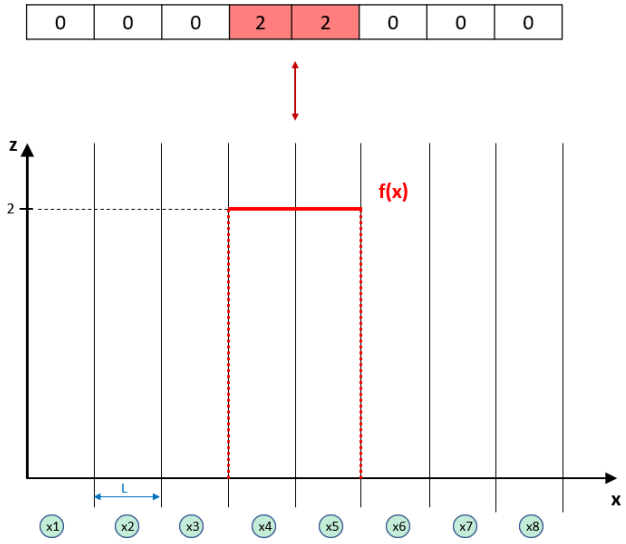


Figure 5.6: From pixel raw to xz profile.

The algorithm explained before approximates the area by adding the product between each depth value in the "object" depth map and the area of the corresponding pixel. In 2D, the area of the pixel becomes its length; the depth values, instead, can be seen as the function values in the discrete intervals representing the pixels. So, the integral is approximated as follows:

$$\int_{x_1}^{x_8} f(x)dx \approx \sum_{i=1}^{8} x_i f(x_i) \tag{5.17}$$

Substituting the values provided by the example, it is possible to obtain:

$$A_1 = x_4 f(x_4) + x_5 f(x_5) = 2L + 2L = 4L \tag{5.18}$$

The second algorithm, instead of considering the height of the pixel understudy, takes the mean value between that pixel and its adjacent. Therefore, in this second proposed approach, the integral is approximated as follows:

$$\int_{x_1}^{x_8} f(x)dx \approx \sum_{i=1}^{7} x_i \left(\frac{f(x_i) + f(x_{i+1})}{2} \right) \tag{5.19}$$

Applying the estimation algorithm to the example, the correct result can be

achieved:

$$A_2 = x_3\left(\frac{f(x_4)+f(x_3)}{2}\right) + x_4\left(\frac{f(x_5)+f(x_4)}{2}\right) + x_5\left(\frac{f(x_6)+f(x_5)}{2}\right) = \frac{2+0}{2}L + \frac{2+2}{2}L + \frac{0+2}{2}L = 4L \tag{5.20}$$

The graphical representation of the two volume estimation methods is illustrated in Figure 5.7.

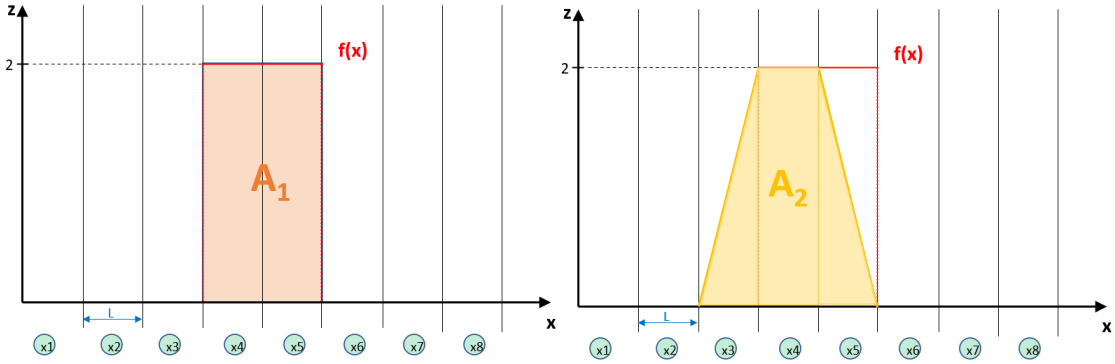


Figure 5.7: Comparison of the two estimation methods on a flat surface.

As demonstrated in previous computations, both algorithms provide the correct area value. However, it seems that the first evaluation method provides a better estimation of the area. This is because a flat surface object is considered. The real advantage in the second approach can be observed when a not flat surface is considered, as it is illustrated in Figure 5.8.

When dealing with curved surfaces, the second estimation algorithm seems to approximate better the area defined under the function, thus providing improving performance. However, both the estimation methods converge to the proper area when the total number of pixels N goes to infinity.
The depth maps that will be obtained from the neural network are made up of many pixels. Therefore, as it will be possible to see in the chapter dedicated to the algorithm test, the two methods will provide similar volume estimates and close to the correct volume value.

5.2.3 From pixels to world units

In Equation 5.14, representing the volume estimation formula of the first approach, d_i indicates the depth value of the pixel i in the "object depth map", which

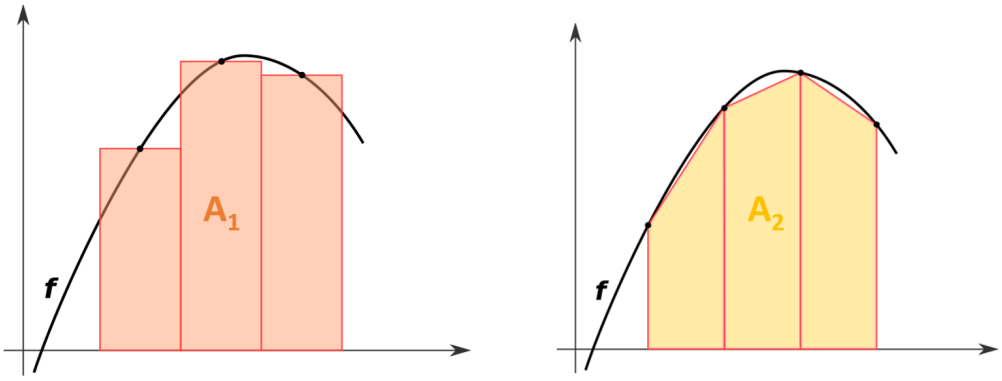


Figure 5.8: Comparison of the two estimation methods on a curved surface.

coincides with the object's height. This quantity is known because it is obtained by subtracting depth maps produced by the neural network. The term A_i instead indicates the area of the pixel i expressed in world units, and for now, it is not yet known.
As explained in Section 5.1, in most cameras, the shape of the pixel is rectangular rather than square. Therefore, Equation 5.14 can be rewritten as follows:

$$Volume = \sum_{i=1}^{N} (mmPerPix_x(i) \cdot mmPerPix_y(i) \cdot ObjHeight(i)) \tag{5.21}$$

where $mmPerPix_x(i)$ is the transformation in millimeters, along the x direction, of what is represented in the pixel i. Similarly, $mmPerPix_y(i)$ is the transformation in millimeters, along the y direction, of what is represented in the pixel i.
This concept is of fundamental importance, and to understand it better, it is possible to think that, within a photo, objects closest to the camera are represented larger than objects further away with the same real size. In other words, the further an object is from the camera, the fewer pixels it will cover in the photo.
Therefore, the more the distance from the camera increases, the greater will be the conversion into millimeters of what is represented in pixel i. For example, a 5 millimeters long object is 4 meters away from the camera, and it covers 1 pixel in the photo. It is wrong to think that a pixel is 5 millimeters long because the same object, if placed 2 meters from the camera instead of 4, occupies 2 pixels. Therefore, according to this reasoning, it should be 10 millimeters long.
Hence, the conversion into millimeters of what is inside the pixel i will be directly proportional to the distance from the camera of what is represented inside the pixel i. So, the quantity $mmPerPix(i)$ is directly proportional to d_i, the depth value of the pixel i.

In order to recover the complete relation between these two quantities, it is possible to consider the pinhole camera model. Objects in the real world are mapped into the image plane according to the Equations 5.1 and 5.2 that can be rewritten as follows:

$$x = f\,\frac{X}{d} \qquad y = f\,\frac{Y}{d} \tag{5.22}$$

where:

- X, Y are respectively the sizes of the 3D object in the world along the x and y directions;
- x, y are respectively the sizes of the 3D object projected in 2D on the image plane along the x and y directions;
- f indicates the focal length of the camera;
- d represents the distance of the object from the camera, previously indicated as Z.

The size of the 3D object projected in 2D on the image plane can be calculated through the camera specifications. Multiplying the ratio between the sensor length and the image resolution with the number of pixels covered by the object, it is possible to compute the length of the object in world units on the image plane.
For example, the iPhone 11 camera sensor size is: $5.6mm \times 4.2mm$ and it takes photos with a resolution of 4032×3024 pixels. Considering the y direction, $4.2mm/3024px \approx 1.389 \cdot 10^{-3} mm/px$ is the size of a pixel in millimeters.
Therefore, an object 1000 pixels long has a length of $1.389mm$ on the sensor.
It is important to underline that the value $1.389 \cdot 10^{-3} mm/px$ is the size of the pixel in millimeters. This quantity is constant and should not be confused with the term $mmPerPix(i)$ in Equation 5.21, which is not constant and represents the transformation in millimeters of what is inside the pixel i.
Hence, the sizes of the object on the image sensor can be rewritten as:

$$\begin{aligned} x &= \frac{SensorLength_x(mm)}{ImageLength_x(pix)} \cdot ObjLength_x(pix) \\ y &= \frac{SensorLength_y(mm)}{ImageLength_y(pix)} \cdot ObjLength_y(pix) \end{aligned} \tag{5.23}$$

Substituting this result into Equation 5.22, and rewriting the sizes of the object in the real world X, Y respectively as $ObjLength_x(mm)$ and $ObjLength_y(mm)$, it

is possible to obtain:

$$\begin{aligned} \frac{ObjLength_x(mm)}{ObjLength_x(pix)} &= \frac{d \cdot SensorLength_x}{f_x \cdot ImageLength_x} \\ \frac{ObjLength_y(mm)}{ObjLength_y(pix)} &= \frac{d \cdot SensorLength_y}{f_y \cdot ImageLength_y} \end{aligned} \tag{5.24}$$

Where the ratio in the first term of the equality defines the correspondence between the size of the object in millimeters in the real world and its size in the image plane. Moreover, it is directly proportional to the distance of the object d. Multiplying this ratio by the number of pixels covered by the object in the image, the real size of the object can be found. Therefore, it is possible to state that:

$$\begin{aligned} \frac{ObjLength_x(mm)}{ObjLength_x(pix)} &= mmPerPix_x \\ \frac{ObjLength_y(mm)}{ObjLength_y(pix)} &= mmPerPix_y \end{aligned} \tag{5.25}$$

Since in general, the distance of the object from the camera changes in each pixel of the image, if a not flat surface is considered, the quantity d becomes $d(i)$ and $mmPerPix$ becomes $mmPerPix(i)$, the transformation in real world units of what is inside the pixel i.

Therefore the complete volume estimation algorithm of the first method is:

$$\begin{aligned} Volume &= \sum_{i=1}^{N} (mmPerPix_x(i) \cdot mmPerPix_y(i) \cdot ObjHeight(i)) \\ mmPerPix_x(i) &= \frac{d(i) \cdot SensorLength_x}{f_x \cdot ImageLength_x} \\ mmPerPix_y(i) &= \frac{d(i) \cdot SensorLength_y}{f_y \cdot ImageLength_y} \end{aligned} \tag{5.26}$$

5.3 Algorithm tests

In the following section will be presented different types of tests applied to the volume estimation algorithm. In particular, three distinct kinds of procedures have been developed, taking into account less and less ideal situations.
Initially, the behavior of the volume algorithm is analyzed by constructing ideal depth maps representing the object whose volume has to be estimated. Assuming

to know the real sizes of the object under consideration, it is possible to manually build the "object depth image" (Figure 5.5). After that, manually constructed depth images are replaced with real depth maps acquired with a depth camera. Finally, the volume estimation algorithm is tested with depth images predicted by a monocular depth estimation network. In this way, it will be possible to estimate the volume of an object starting from a single RGB image that portrays it.

5.3.1 Test with ideal depth maps

The volume estimation algorithm is firstly tested with matrices representing ideal depth maps. The constructed matrices would ideally be those obtained by subtracting the depth map of the background from the depth map of the object placed above the background, as explained in Section 5.2.1. Therefore, each matrix is an ideal "object depth map" where in each pixel of the image is stored the height of the object whose volume has to be estimated. Hence, each matrix represents a certain shape in the 3D space. For example, supposing to have depth maps in millimeters, the matrix representing a pyramid 90 millimeters high is illustrated in the Figure 5.9.

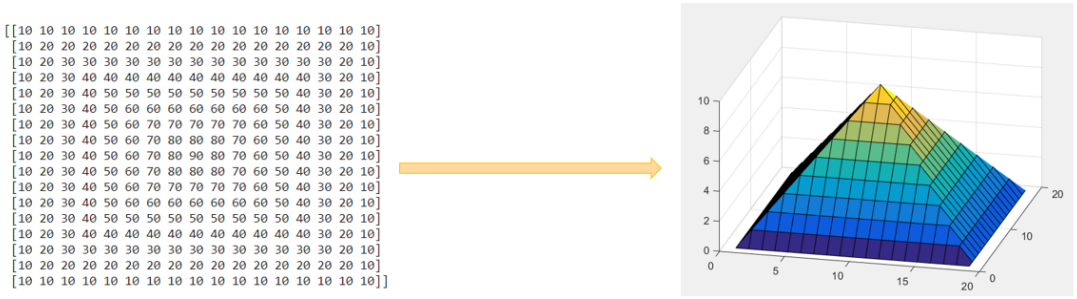


Figure 5.9: Ideal depth map of a pyramid 90mm high.

Construction of the ideal depth map

In order to construct ideal depth maps some assumptions have to be done. The tests are performed assuming to have:

- An iPhone 11 camera sensor, which has:
  - $f_x \approx f_y \approx 4.25mm$
  - $SensorLength_x \approx 5.6mm$
  - $SensorLength_y \approx 4.2mm$
- Depth maps with:

  - $ImageLength_x = 160pix$
  - $ImageLength_y = 128pix$

- A flat background at a distance of $470mm$ from the camera

Choosing sizes of a shape in world units, it is possible to compute the corresponding sizes in the image plane.
For example, taking into account a cuboid object with the following sizes:

- $ObjLength_x(mm) = 150mm$
- $ObjLength_y(mm) = 120mm$
- $ObjHeight = 80mm$

The corresponding sizes in the image plane can be computed as follows:

$$\begin{aligned} ObjLength_x(pix) &= \frac{f_x \cdot ImageLength_x \cdot ObjLength_x(mm)}{d \cdot SensorLength_x} \approx 47pix \\ ObjLength_y(pix) &= \frac{f_y \cdot ImageLength_y \cdot ObjLength_y(mm)}{d \cdot SensorLength_y} \approx 40pix \end{aligned} \tag{5.27}$$

In each of these 47×40 pixels is stored the height of the object, equal to $80mm$. Taking into account all these information, it is possible to build the three depth maps of the object reconstruction strategy that are illustrated with all values in centimeters in Figure 5.10.

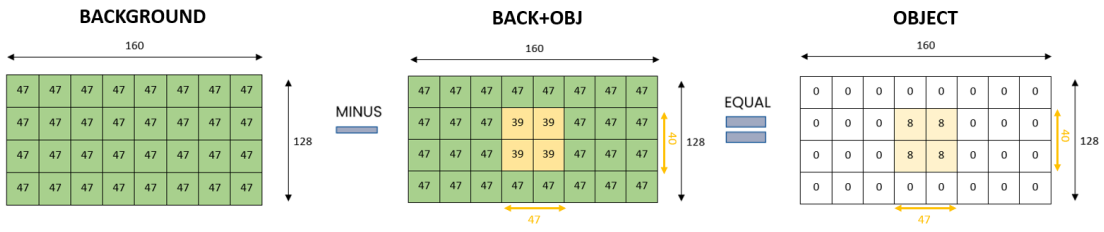


Figure 5.10: Ideal depth maps of a cuboid object $8cm$ high placed on a flat background $47cm$ away from the camera.

Test Results

Once the object is reconstructed, it is possible to apply the two algorithms described in Section 5.2.2.

By running the Python code reported in Appendix A, the following numerical results can be obtained:

$$\begin{aligned} V_1 &= 1454.47cm^3 \\ V_2 &= 1454.47cm^3 \end{aligned} \tag{5.28}$$

where V_1 and V_2 are the estimated volume computed through the first and second presented algorithm.
And the correct volume of a cuboid is:

$$V = ObjLength_x \cdot ObjLength_y \cdot ObjHeight = 1440cm^3 \tag{5.29}$$

Following this reasoning, the algorithm is tested with other 3D shapes and some results are reported in Table 5.1.

Shape	**Size**$[cm]$	**Estimated V1**$[cm^3]$	**Estimated V2**$[cm^3]$	**Correct V**$[cm^3]$
Cube	$l = 6.00$	219.32	219.32	216.00
Cuboid	$l_x = 15.00$ $l_y = 12.00$ $h = 8.00$	1454.47	1454.47	1440.00
Pyramid	$l_x = 6.58$ $l_y = 6.17$ $h = 9.00$	111.31	111.49	121.79
Cylinder	$r_x = 7.90$ $l_y = 7.41$ $h = 15.00$	2755.01	2755.01	2758.58

Table 5.1: Test of the volume estimation algorithm with ideal depth maps.

As can be noticed, the results obtained by the two volume algorithms are different only when the surface considered is not flat. However, even when a curved surface is taken into account, the two outcomes are very similar. Increasing the number of pixels towards infinity, they will converge to the same number. For this reason, only the results obtained through the first volume estimation method will be analyzed in the following sections.

5.3.2 Test with acquired depth maps

In this type of test, manually constructed depth images are replaced with real depth maps acquired with an RGB-D camera. The RGB-D camera, also called

depth camera, is a camera capable of acquiring not only the RGB image of the depicted scene but also its corresponding depth image.
In particular, the volume estimation algorithm is tested with the Intel Realsense D435i camera. Through its D430 module based on stereoscopic technology, this RGB-D camera can detect depth in a range from 0.3 meters to over 10 meters, generating very accurate depth maps up to 3 meters away. The left and right cameras of the D430 module capture the scene and send data to the processor, which calculates depth values for each pixel in the image by correlating points on the left image to the right image [28]. Moreover, there is also an Infrared (IR) Laser Projector System to improve depth accuracy in scenes with low texture.
The tests are carried out using through following steps, visible in Figure 5.11. Initially, the depth map of the background is acquired with the Intel camera. Also in this case a flat surface is taken as the background. After that, the background depth image is saved since it is always the same; what changes is the object placed on top of it. Once the background depth image is stored, several images are captured with different objects placed one at a time over the selected background. By making the difference between the depth map of the background and the one of the object positioned above it, it is possible to reconstruct the object. So, it is possible to obtain the "object depth map", where each pixel contains the object's height in the various points of the image.

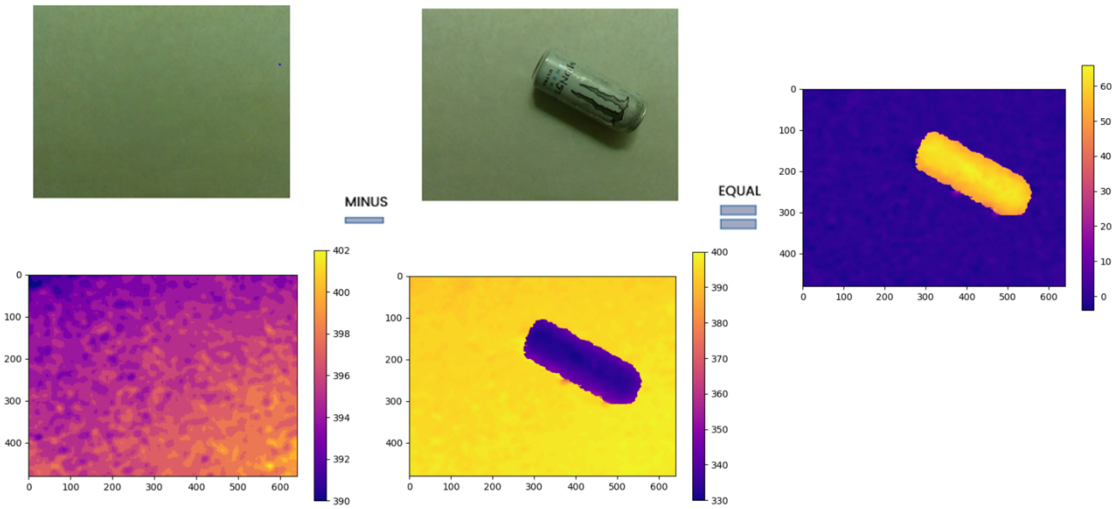


Figure 5.11: Object reconstruction strategy by acquiring the RGB image (on top) and the corresponding aligned depth image (on bottom) with the Intel Realsense D435i camera. Depth values are expressed in millimeters.

Once the object is reconstructed, it is possible to apply the volume estimation

algorithm (Equation 5.26) by taking into account the Intel Realsense D435i camera intrinsic parameters.
Some results of the volume estimation applied to different objects are illustrated in Table 5.2. The column "object depth image" represents the depth image obtained through the object reconstruction strategy, with depth values expressed in millimeters according to the depth range shown in Figure 5.12. The correct volume V is approximated to the volume of liquid contained in the object; otherwise, for items that do not contain liquids, it is computed using the ruler.

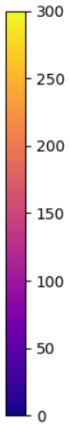


Figure 5.12: Fixed depth range with values expressed in millimeters.

The performed simulations show a good behaviour of the algorithm and a nice consistency between computed volumes of the same object placed in different positions. Some volume estimation errors occur when infrared rays emitted by the Intel Realsense camera are parallel to the object's surface, and due to this, the surface is not detected by the camera.
Since the final goal is to obtain a depth map using only an RGB image, the next step is to test the algorithm with a depth estimation network able to perform this task.

5.3.3 Test with estimated depth maps

The following test aims to understand how the algorithm behaves in the final system, where there is not a depth camera but a simple RGB camera. Consequently, monocular depth estimation networks must be used to obtain a predicted depth image by means of a single RGB image.
To test the algorithm with estimated depth maps, it is used the state-of-the-art convolutional neural network of Lee *et al.* [25], in order to take advantage of recent improvements in the depth estimation. The model is called *BTS*, and among

the different backbone structures proposed to extract the features, the *DenseNet-161* network is chosen, since it represents a good trade-off between accuracy and complexity. The estimation network takes an RGB image of resolution 640×480 and produces as output the corresponding predicted aligned depth map with the same resolution of the input image. In Figure 5.13, is shown the result of the object reconstruction strategy, capturing the RGB image with the Intel Realsense D435i camera and estimating the corresponding depth map through the BTS model.

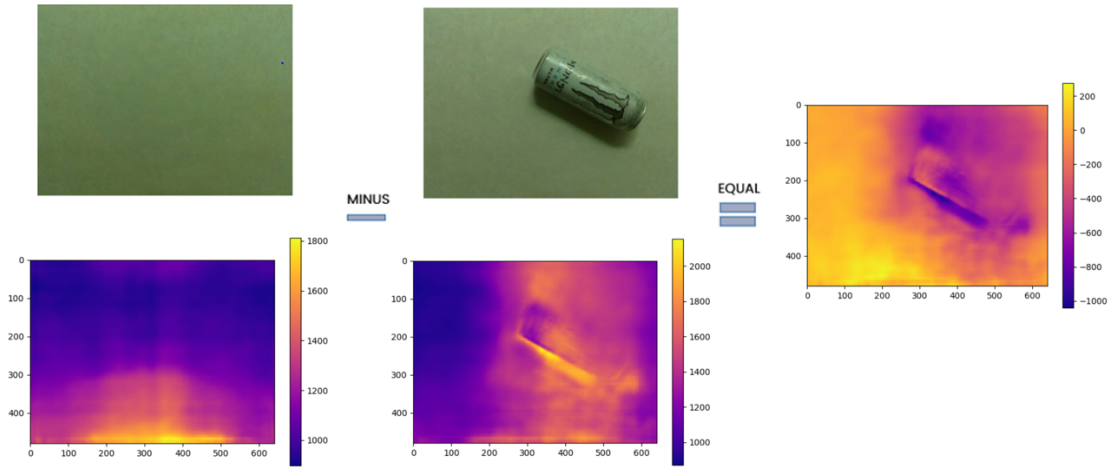


Figure 5.13: Object reconstruction strategy by acquiring the RGB image with the Intel Realsense D435i camera (on top) and estimating the corresponding aligned depth image (on bottom) with the BTS model already trained on NYU Depth Dataset V2. Depth values are expressed in millimeters.

As it is possible to see from Figure 5.13, the prediction of the depth map of the background and of the object placed above the background are completely wrong. The correct estimation would be the one represented in Figure 5.11.
Since by using the already trained BTS model (on *NYU Depth Dataset V2*), the object cannot be reconstructed properly, the network must be retrained on a specific dataset built for this type of application.

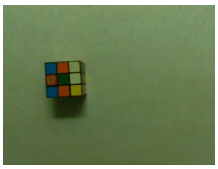
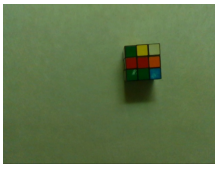
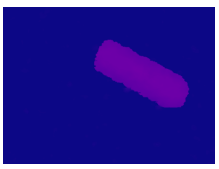
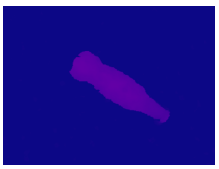
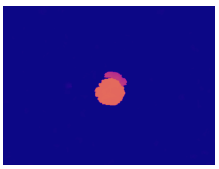
Object	**Estimated V**$[cm^3]$	**Correct V**$[cm^3]$	**RGB image**	**Object depth image**
Rubik's cube	216.17	216.00		
Rubik's cube	210.78	216.00		
Monster	504.09	500.00		
Monster	433.33	500.00		
Coca-Cola	264.62	250.00		
Coca-Cola	231.54	250.00		

Table 5.2: Test of the volume estimation algorithm with depth maps acquired through the Intel Realsense D435i depth camera.

Object	**Estimated V**$[cm^3]$	**Correct V**$[cm^3]$	**RGB image**	**Object depth image**
Beer	334.13	330.00		
Beer	325.28	330.00		
Red Bull	262.89	250.00		
Red Bull	242.48	250.00		

Table 5.3: Test of the volume estimation algorithm with depth maps acquired through the Intel Realsense D435i depth camera.

Chapter 6

MDE Network Training

The volume estimation algorithm previously described can be applied starting from the depth map of the reconstructed object, which contains in each pixel the actual height of the object. The algorithm has to be run inside the intelligent bin Nando, which does not contain an RGB-D camera able to produce dense depth maps, but only a simple RGB camera. The only way to obtain a depth map starting from a single RGB image is to perform monocular depth estimation. As mentioned before, among all the MDE networks created by the scientific community, it is adopted the one proposed by Lee *et al.* [25] called BTS and previously explained in Section 4.2.1. Among all the various base networks proposed, DenseNet-161 has been chosen since it provides a good trade-off between complexity and performance. As shown in Section 5.3.3, the BTS model already trained on NYU Depth Dataset V2 does not achieve satisfactory results, and as a consequence, the object cannot be reconstructed properly. Therefore, it is necessary to repeat the training process on a specific dataset built for this type of application. It means to train the network with images very similar to those used in the inference phase, with similar portrayed scenes and depth values. In order to obtain better results, the training of the MDE network has been performed in a supervised way, acquiring both RGB and depth images with the Intel Realsense D435i depth camera.

6.1 Training parameters

The code used to train the BTS model on the PyTorch framework [29], an open-source machine learning framework, needs some arguments to be run, which are passed to the training function in file format. The adopted training parameters are reported in Figure 6.2, and the most important ones are discussed below. The customize dataset specifically built for the volume estimation application will be called "My dataset".

```
--mode train
--model_name bts_nyu_v2_pytorch_test
--encoder densenet161_bts
--dataset nyu
--data_path ../../dataset/My_dataset/train/
--gt_path ../../dataset/My_dataset/train/
--filenames_file ../train_test_inputs/My_train_file.txt
--batch_size 4
--num_epochs 100
--learning_rate 1e-4
--weight_decay 1e-2
--adam_eps 1e-3
--num_threads 1
--input_height 416
--input_width 544
--do_random_rotate
--degree 2.5
--log_directory ./models/
--multiprocessing_distributed
--dist_url tcp://127.0.0.1:2345

--log_freq 100
--do_online_eval
--eval_freq 500
--data_path_eval ../../dataset/My_dataset/test/
--gt_path_eval ../../dataset/My_dataset/test/
--filenames_file_eval ../train_test_inputs/My_test_file.txt
--eval_summary_directory ./models/eval/
--eigen_crop
```

Figure 6.1: Training code arguments.

- The total number of epochs is set to 100 with batch size equal to 4.
- It is adopted an Adam optimizer with $\epsilon = 10^{-3}$, and a base learning rate of 10^{-4}. It is used to update network weights and, unlike classical methods, it computes individual learning rates for different parameters. The Adam optimizer achieves good results faster than the classical stochastic gradient descent method.
- The weight decay technique is applied to avoid the exploding gradient problem. By means of this method the final loss function is:

$$Loss = Loss + weigth\ decay\ parameter \cdot L_2\ norm\ of\ the\ weights \quad (6.1)$$

  This technique allows minimizing not only the loss function but also the parameter weights. As loss function, it is employed the scale-invariant error described in Section 4.2.2 and as loss decay parameter is used the value 10^{-2}.
- Images before input to the network are randomly cropped to a resolution of 544×416. Moreover, to avoid over-fitting, input images are augmented

by using random contrast, brightness and randomly rotated in a range of [−2.5,2.5] degree.

- The model is evaluated every 500 steps, and the best model for each evaluation metric is saved.
- In the dataset path there must be a directory with the training samples and a directory with the testing ones. Both training and testing samples have to be listed in a separate *txt* file according to the following template:

```
/color1.jpg /depth1.png 1.93
/color2.jpg /depth2.png 1.93
/color3.jpg /depth3.png 1.93
/color4.jpg /depth4.png 1.93
/color5.jpg /depth5.png 1.93
/color6.jpg /depth6.png 1.93
/color7.jpg /depth7.png 1.93
/color8.jpg /depth8.png 1.93
/color9.jpg /depth9.png 1.93
```

Figure 6.2: Txt file snippet used to list the training/testing samples.

Both training and testing *txt* files must contain in the same line:

1. The file name of the RGB image in *JPG* extension;
2. The file name of the corresponding depth image in *PNG* extension;
3. The focal length of the camera used to capture the images, in this case, the Intel Realsense D435i depth camera.

Software setup

Once all the directories are created and the paths are defined, the training code is run on Google Colab, an online service allowing to write Python code using the computational power provided by Google's virtual GPUs. This service is especially useful in machine learning, where the computational power provided by the physical GPU of the computer would not be enough to train very large datasets.
PyTorch and its related libraries are installed from *pip* following the compatibility indications provided on the PyTorch website [30]. In particular, the following library versions are chosen and are installed on Colab through the line of code reported below:

```
1 !pip install torch==1.7.0+cu110 torchvision==0.8.1+cu110 torchaudio
  ==0.7.0 -f https://download.pytorch.org/whl/torch_stable.html
```

6.2 Data acquisition

RGB images and the corresponding aligned depth data are acquired with the Intel Realsense D435i depth camera. In order to obtain depth maps as accurate as possible for the type of environment in which the dataset is built, the following actions are applied:

- The ShortRangePreset in *JSON* format is loaded into the camera since the dataset will contain images such that the scene depicted is very close to the camera. It is available in the Intel Realsense documentation [31], and it is recommended to optimize depth data accuracy when acquiring depth maps with a depth range lower than 1 meter.
  Subsequently, the cross-platform library Intel *RealSense SDK 2.0* is downloaded, and by means of the*Intel Realsense Viewer* application, some advanced settings have been fine-tuned. So, the ShortRangePreset has been slightly modified to obtain even better performance, and before starting the data acquisition phase, the customized preset is loaded into the camera.

- Both RGB and depth images are captured with a resolution of 640×480 according to the demands of the BTS model.

- Since the color and the depth sensor have different fields of view in the Intel Realsense D435i camera, the depth frame is aligned to the color frame in order to obtain depth maps aligned to the corresponding RGB images.

- After each data acquisition, depth maps are post-processed to enhance depth data quality and reduce noise levels by using the following filters suggested by the Intel Realsense documentation [32]:

  1. Depth maps are transformed into disparity maps;
  2. Spatial Edge-Preserving filter is used to improve the smoothness of the reconstructed data;
  3. Temporal filter is employed to enhance consistency between frames;
  4. Disparity maps are transformed into depth maps.

The Python code related to the data acquisition and storage is reported in Appendix B.
Once RGB and depth data are acquired, the dataset can be divided into training and testing samples, which must be listed in two separate *txt* files as described in Section 6.1. The Python code related to the data split and the *txt* generation is reported in Appendix C.

6.3 Dataset

In order to obtain satisfactory results from the training process, the dataset must contain images as similar as possible to the images that will be acquired during the inference time, therefore to the images taken from the Nando smart bin. The camera inside Nando takes photos from the top, with a distance of about 40 centimeters from the background. Moreover, the captured images mainly depict waste (e.g. cans, bottles or pieces of paper)
Taking into account these information, two datasets has been built with the Intel Realsense D435i depth camera. The first one has been constructed to see if the training process would have improved the accuracy of the depth maps predicted from the network. Instead, the second dataset has been created taking photos inside a Nando prototype, simulating the working environment inside the real Nando.

6.3.1 First working environment

The first dataset is built with a flat and white background in which objects are placed once a time, with the Intel Realsense camera positioned at the top, at a distance of 40 centimeters from the background. This background has been chosen to make objects stand out and avoid light reflections since lighting spots create problems in predicting depth. The lighting conditions are controlled and kept constant for all the acquisition time.
Once RGB and depth data are acquired, the BTS model is trained on the customize dataset. The estimation network is tested with the same images used in Figure 5.13, and the results are shown in Figure 6.3.

As it is possible to notice, the depth maps estimated by the BTS model trained on the customized dataset are much more accurate than those estimated by the BTS model already trained on NYU Depth Dataset V2, and the object can be reconstructed properly. This means that the training process on the customize dataset is efficient, and, to obtain even better results, an even more application-specific dataset can be built.

6.3.2 Second working environment

The second dataset is built acquiring RGB and depth data with the Intel Realsense camera in an environment as similar as possible to the one present inside Nando. In particular, it is used a Nando prototype simulating the working environment inside the real Nando. Both systems and the corresponding images taken inside them, are illustrated in Figure 6.4.

The final dataset is obtained appending the data captured in the first working

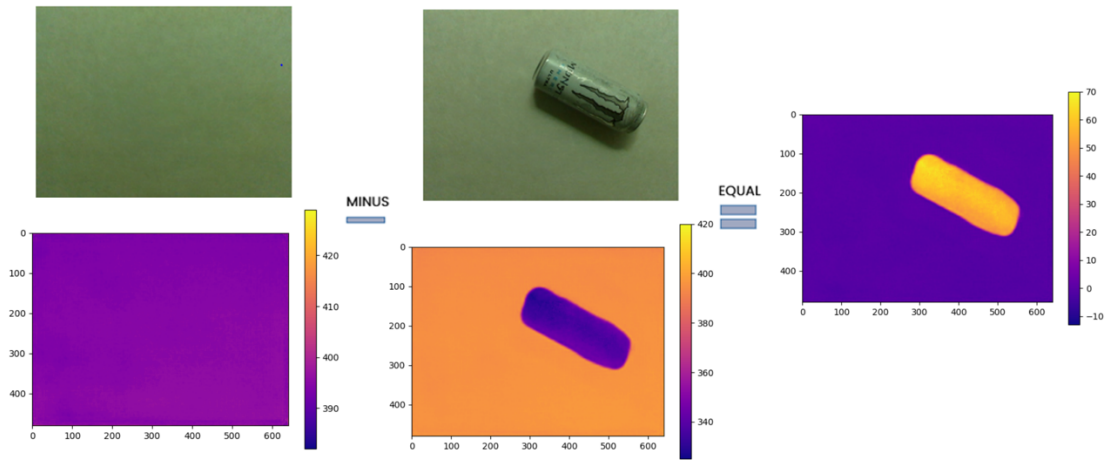


Figure 6.3: Object reconstruction strategy by acquiring the RGB image with the Intel Realsense D435i camera (on top) and estimating the corresponding aligned depth image (on bottom) with the BTS model trained on the customize dataset. Depth values are expressed in millimeters.

Figure 6.4: Comparison between the real Nando (left) and the Nando prototype (right).

environment with the ones acquired inside the Nando prototype. It consists of about one thousand and five hundred RGB photos with the corresponding aligned depth maps.
Laslty, the training process of the BTS model is repeated on the final dataset, and the evaluation results are reported in Figure 6.5. Every 500 steps, the best model for each evaluation metric is saved. Once the training process ends, the model with the best RMSE is chosen to test the final system. Similar volume estimation results can be obtained using the best models on the other evaluation metrics.

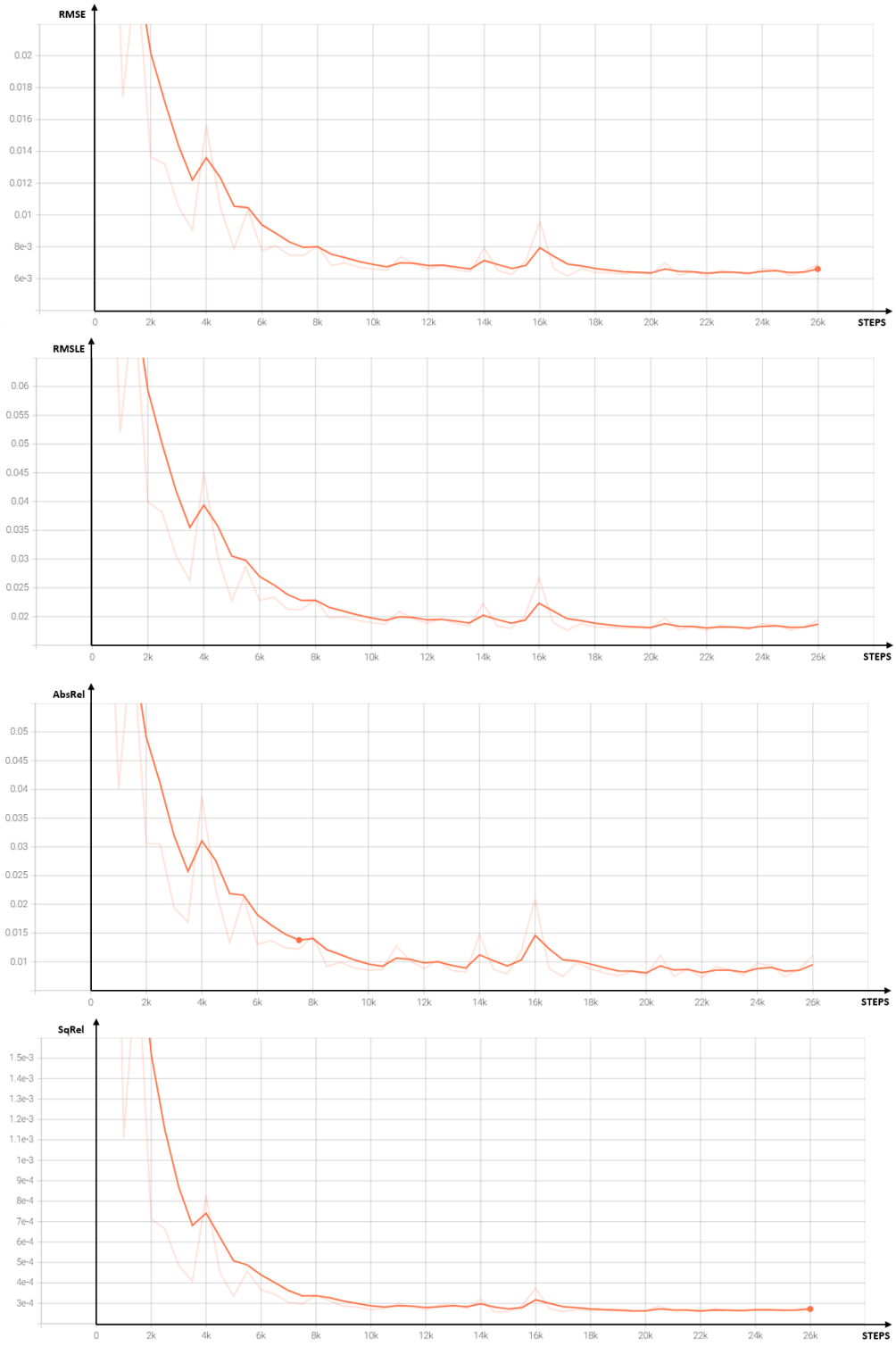


Figure 6.5: Evaluation metrics evolution during the training process.

6.4 System test

The final system is tested by taking images with the Raspberry Pi Camera since it is the hardware used by Nando to recognize the material of which the waste is composed.
As it is possible to notice from Figure 6.6, the object depth map obtained through the reconstruction strategy has some problems. In particular, around the object, there are some depth values different from zero. Even if they do not belong to the object, they make a contribution to the computation of its volume, and therefore they must be removed.
Since the diameter of the bin is constant and known, it is possible to create a simple filter to eliminate depth values outside the diameter of the bin.

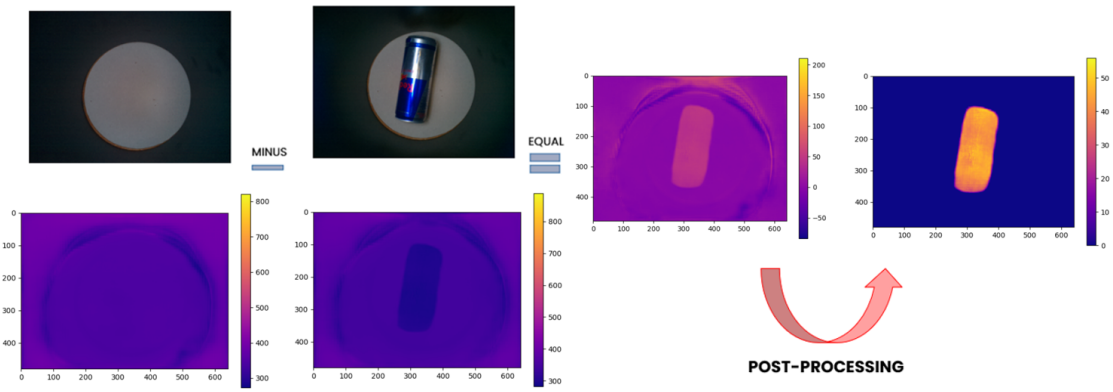


Figure 6.6: Object reconstruction strategy by acquiring the RGB image with the Raspberry Pi Camera (on top) and estimating the corresponding aligned depth image (on bottom) with the BTS model trained on the customize dataset. Depth values are expressed in millimeters.

Once the object is reconstructed, it is possible to apply the volume algorithm by taking into account the Raspberry Pi Camera intrinsic parameters. The corresponding Python code is reported in Appendix D.
Some results of the volume estimation applied to different objects are illustrated in Table 6.1. The column "object depth image" represents the depth image obtained through the object reconstruction strategy (after post-processing), with depth values expressed in millimeters, according to the depth range shown in Figure 5.12. The correct volume V is approximated to the volume of liquid contained in the object; otherwise, for items that do not contain liquids, it is computed using the ruler.

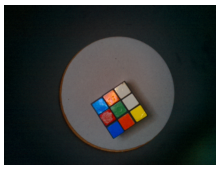
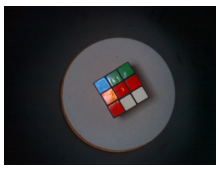
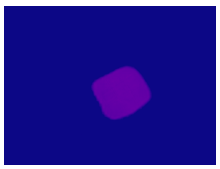
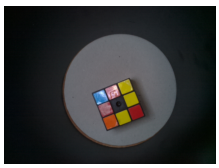
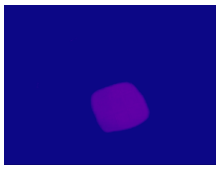
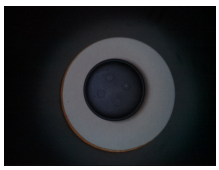
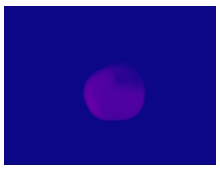
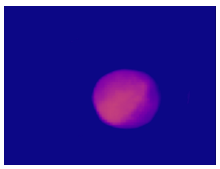
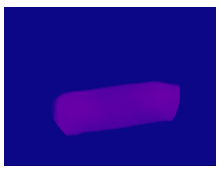
Object	$\mathbf{V_E}[cm^3]$	$\mathbf{V}[cm^3]$	**Error**(%)	**RGB image**	**Object depth image**
Rubik's cube	220.41	216.00	2.04		
Rubik's cube	226.29	216.00	4.76		
Rubik's cube	205.93	216.00	4.66		
Alexa	209.82	223.00	5.91		
Alexa	402.99	223.00	80.71		
Monster	475.14	500.00	4.97		
Monster	496.82	500.00	0.64		

Table 6.1: System test results, where V represents the correct volume and V_E the estimated one.

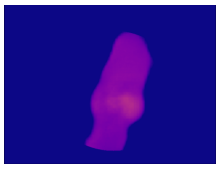
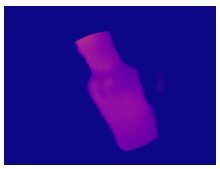
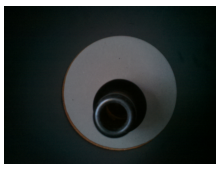
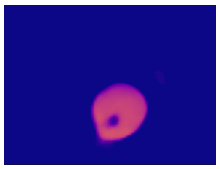
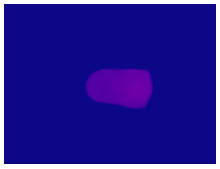
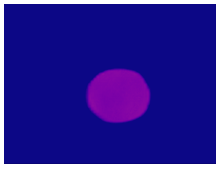
Object	$\mathbf{V_E}[cm^3]$	$\mathbf{V}[cm^3]$	**Error**(%)	**RGB image**	**Object depth image**
Monster	420.87	500.00	15.82		
Beer	621.25	330.00	88.26		
Beer	646.07	330.00	95.78		
Beer	337.19	330.00	2.18		
Estathé	243.07	200.00	21.53		
Estathé	215.06	200.00	7.53		
Estathé	344.81	200.00	72.40		

Table 6.2: System test results, where V represents the correct volume and V_E the estimated one.

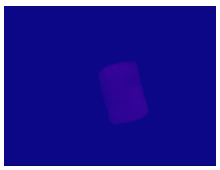
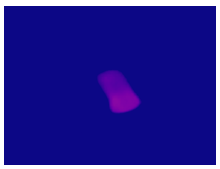
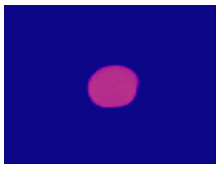
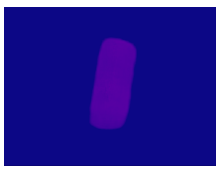
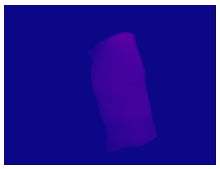
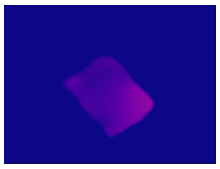
Object	$\mathbf{V_E}[cm^3]$	$\mathbf{V}[cm^3]$	**Error**(%)	**RGB image**	**Object depth image**
Pills	102.65	128.00	19.80		
Pills	117.83	128.00	7.94		
Pills	117.29	128.00	8.37		
Red Bull	247.89	250.00	0.84		
Red Bull	281.93	250.00	12.77		
Box	342.27	550.00	37.77		
Box	356.47	550.00	35.19		

Table 6.3: System test results, where V represents the correct volume and V_E the estimated one.

Chapter 7

Conclusions

The presented work aims to design a system able to estimate the volume of an object placed inside the smart bin Nando, starting exclusively from an RGB image that portrays it. The performed simulations show a fine consistency in the estimates obtained on the same object positioned in different ways. The estimation results are satisfactory, even though the only hardware tool needed to obtain volumetric estimates is a simple RGB camera. Moreover, the developed algorithm is scalable since it can be applied with any camera by only knowing the intrinsic parameters of the sensor. However, predicting the depth of a scene, having only one point of view, thus having only one camera, is a very difficult task, especially if it does not exist in the scientific community large datasets containing the scenes specific to your application. In addition, the volume algorithm is very sensitive to the predicted depth maps, and therefore its effectiveness depends significantly on the accuracy with which the depth is estimated. So, as can be seen from the tests conducted, some estimation results are not particularly accurate. Specifically, errors on volume estimation can essentially be divided into two types:

- Errors due to the depth estimation
- Errors due to the volume algorithm

Errors due to the depth estimation

A critical situation arises when trying to calculate the volume of objects not present in the training dataset. In this case, it is unknown whether the monocular depth estimation network will predict the correct depth, and the final result may be right or wrong. A wrong result on the volume calculation occurs when the depth map obtained from the object reconstruction process is wrong, so when the object is reconstructed with a height lower or higher than the real one. Furthermore, the depth map obtained from the object reconstruction process is wrong when

the depth map of the object positioned above the background is not accurately estimated since the neural network cannot recognize the object in the photo.
For example, *Alexa* is an object that has not been inserted inside the dataset. Indeed, as can be seen in Table 6.1, the first volume estimate of *Alexa* is quite good, but when it is upside-down, in the second sample, the estimated volume is wrong. The object is reconstructed with a height much greater than the real one. This issue occurs because the estimation network has never seen this type of object, and maybe it thinks that *Alexa* is another object with a greater height present within the dataset. The same phenomenon occurs in calculating the volume of the *Box* in Table 6.3 since it is not an object that has been inserted into the dataset. In this case, however, the object's height is underestimated, probably because it is mistaken for another box present within the dataset but with a lower height.

Another problem occurs when transparent objects, such as plastic bottles are placed inside the bin. In this case, the depth estimation is not accurate. Nevertheless, this kind of problem arises in all passive methods of 3D reconstruction and it is even difficult to accurately acquire depth with RGB-D cameras.

Errors due to the volume algorithm

The volume computation also fails whenever the camera, positioned at the top of the bin, cannot see that there are some void spaces between the surface of the object and the background. As a result, the camera thinks these empty spaces are part of the object, and the volume is overestimated.
This kind of problem is visible in Table 6.2 when trying to estimate the volume of the *Estathè*. In the third sample, the *Estathè* is standing upright. The camera does not see empty spaces between the upper surface of the object and the background. As a consequence, the volume is overestimated.
The same type of problem occurs when large objects are placed inside the bin. These objects, having a height greater than the diameter of the basket, when they fall inside it, do not go entirely into contact with the background and are positioned obliquely, generating empty spaces. Indeed, as can be seen in Table 6.2, in the first two *Beer* samples, the estimated volume is greater than the correct one. A 2D representation of the problem is illustrated in Figure 7.1.

Estimating the volume of large objects that fall obliquely involves another type of error. As explained in Section 6.4, only depth values within the bin diameter are taken into account; the others are filtered out. Large objects, having a height greater than the diameter of the bin, are cut by the filter, and therefore the object is not entirely reconstructed. A possible solution to this problem is that the elliptical filter does not completely eliminate depth contributions outside the diameter of the bin but gives them a smaller weight. In this way, the object would be completely

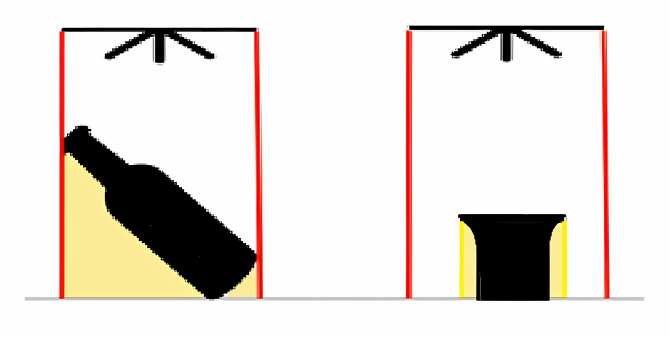


Figure 7.1: Volume estimation problems due to the algorithm. The camera on the top does not see empty spaces (in yellow) between the object and the background. As a consequence the volume is overestimated.

reconstructed. However, the algorithm would go even further to overestimate the object's volume because of the empty spaces problem. Consequently, it is accepted that the object is cut so that this kind of error goes a little bit to compensate for the error due to the void spaces.

Moreover, it is important to remark that, for objects which are drinks, the correct volume V is approximated to the volume of liquid contained in the object. However, sometimes the volume of liquid is less than the volume of the object that contains it. For this reason, the algorithm may overestimate the computed volume.

7.1 Future works

Some significant improvements that can be made in this work could be:

- To obtain a more precise depth estimation, inserting further images within the dataset or exploring the results achieved through other state-of-the-art MDE networks;
- To get a more accurate object reconstruction, transforming depth maps into point clouds. After that, once outliers are removed, it is possible to partition the surface into Delaunay triangles or create a simple convex-hull and then compute the volume.

Possible development of this work could be to extend the algorithm so that it can obtain volume estimates starting from an RGB image that contains several objects. For example, it would be possible to insert a segmentation network to isolate all the objects in the image and calculate their volume separately.

Appendix A

Standard volume estimation algorithm

Below is reported the Python code of the volume estimation algorithm applied to the example presented in Section 5.3.1.

```
import numpy as np

rows=128        #coincides with ImageLength_y
columns=160     #coincides with ImageLength_x

#background matrix
background=[ [ 470 for i in range(columns) ] for j in range(rows) ]
background=np.array(background)
background=background.astype('float64')

#back+obj matrix
back_plus_obj=background.copy()
ObjHeight=80      #80mm
for i in range(rows):
  for j in range(columns):
    if i<47 and j<40:  #object covers 47 pixel along x and 40 along y
      back_plus_obj[i][j] = back_plus_obj[i][j] - ObjHeight



#ALGORITHM ------------------------------------------------------------

#Object matrix
obj = [ [ 0 for i in range(columns) ] for j in range(rows) ]
obj=np.array(obj)
obj=obj.astype('float64')
```

```
27 obj=np.subtract(background, back_plus_obj)
28
29 #IPhone 11 camera sensor parameters
30 f=4.25       #4.25 mm
31 SensorLength_x=5.6       #5.6mm
32 SensorsLength_y=4.2       #4.2mm
33
34 #mmPerPix
35 mm_pix_x = [ [ 0 for i in range(columns) ] for j in range(rows) ]
36 mm_pix_x=np.array(mm_pix_x)
37 mm_pix_x=mm_pix_x.astype('float64')
38
39 mm_pix_y = [ [ 0 for i in range(columns) ] for j in range(rows) ]
40 mm_pix_y=np.array(mm_pix_y)
41 mm_pix_y=mm_pix_y.astype('float64')
42
43 for i in range(rows):
44   for j in range(columns):
45     if obj[i][j]==0:
46       mm_pix_x[i][j]=0  #to avoid useless computations
47     else:
48       mm_pix_x[i][j]=(back_plus_obj[i][j]*SensorLength_x)/(f*columns)
49
50 for i in range(rows):
51   for j in range(columns):
52     if obj[i][j]==0:
53       mm_pix_y[i][j]=0  #to avoid useless computations
54     else:
55       mm_pix_y[i][j]=(back_plus_obj[i][j]*SensorLength_y)/(f*rows)
56
57 #Transformation in centimeters
58 obj=obj/10
59 cm_pix_x=mm_pix_x/10
60 cm_pix_y=mm_pix_y/10
61
62 #Volume estimation algorithm 1
63 volume1=0
64 for i in range(rows):
65   for j in range(columns):
66     volume1 += cm_pix_x[i][j]*cm_pix_y[i][j]*obj[i][j]
67 print(volume1)
68
69 #Volume estimation algorithm 2 (applied only in the central pixels
       covered by the object)
70 volume2=0
71 for i in range(rows):
72   for j in range(columns):
73     if obj[i][j]!=0 and obj[i+1][j]!=0:
```

```
74          volume2 += cm_pix_x[i][j]*cm_pix_y[i][j]*(obj[i][j]+obj[i+1][j
    ])/2
75      elif obj[i][j]!=0 and obj[i+1][j]==0:
76          volume2 += cm_pix_x[i][j]*cm_pix_y[i][j]*obj[i][j]
77  print(volume2)
```

Appendix B

Intel Realsense D435i data acquisition

Below is reported the Python code relative to the data acquisition process described in Section 6.2.

```
import cv2
import pyrealsense2 as rs
import numpy as np
from PIL import Image as im
import matplotlib.pyplot as plt
import os
import json

# Setup:
pipe = rs.pipeline()
cfg = rs.config()
pipeline_wrapper = rs.pipeline_wrapper(pipe)
pipeline_profile = cfg.resolve(pipeline_wrapper)
device = pipeline_profile.get_device()
device_product_line = str(device.get_info(rs.camera_info.product_line
    ))

# Customize preset
jsonObj = json.load(open("My_preset.json"))
json_string= str(jsonObj).replace("'", '\"')
advnc_mode = rs.rs400_advanced_mode(device)
advnc_mode.load_json(json_string)

# Streaming resolution
cfg.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)
cfg.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)
```

```
26
27 # Start streaming
28 pipe.start(cfg)
29
30 # Create an align object
31 # The "align_to" is the stream type to which we plan to align depth
       frames.
32 align_to = rs.stream.color
33 align = rs.align(align_to)
34
35 # Skip 10 first frames to give the Auto-Exposure time to adjust
36 for x in range(10):
37   pipe.wait_for_frames()
38
39 # Store next frameset for later processing:
40 frameset = pipe.wait_for_frames()
41
42 # Align the depth frame to color frame
43 aligned_frames = align.process(frameset)
44 color_frame = aligned_frames.get_color_frame()
45 depth_frame = aligned_frames.get_depth_frame()
46
47 # Cleanup:
48 pipe.stop()
49 print("Frames Captured")
50
51
52 #POST-PROCESSING ------------------------------------------------
53 #Filters parameters are chosen through a fine-tuning process
54
55 # Dept2Disparity
56 depth_to_disparity = rs.disparity_transform(True)
57 depth_frame=depth_to_disparity.process(depth_frame)
58
59 # Spatial filter
60 spatial = rs.spatial_filter()
61 spatial.set_option(rs.option.filter_magnitude, 5)
62 spatial.set_option(rs.option.filter_smooth_alpha, 1)
63 spatial.set_option(rs.option.filter_smooth_delta, 50)
64 spatial.set_option(rs.option.holes_fill, 4)
65 depth_frame = spatial.process(depth_frame)
66
67 # Temporal filter
68 temp=rs.temporal_filter()
69 depth_frame=temp.process(depth_frame)
70
71 # Disparity2Depth
72 disparity_to_depth = rs.disparity_transform(False)
73 depth_frame=disparity_to_depth.process(depth_frame)
```

```
74
75
76 #PLOTS ------------------------------------------------------------------
77
78 depth_array=np.asanyarray(depth_frame.get_data())
79 color_array=np.asanyarray(color_frame.get_data())
80
81 rows=depth_array.shape[0]
82 columns=depth_array.shape[1]
83
84 #First depth frame plot
85 plt.figure(1)
86 plt.imshow(depth_array)
87
88 # Invalid depth values are set to zero
89 for i in range(rows):
90   for j in range(columns):
91     if depth_array[i][j]>400:
92        depth_array[i][j]=0
93        #since the distance between the camera and the bottom of the
     bin is maximum 400mm
94
95 #Second depth frame plot
96 plt.figure(2)
97 plt.imshow(depth_array)
98
99 # RGB frame plot
100 plt.figure(3)
101 plt.imshow(cv2.cvtColor(color_array, cv2.COLOR_BGR2RGB)) #convert
     image from BGR to RGB
102 plt.show()
103
104
105 #DATA BACKUP ------------------------------------------------------------
106
107 n_color=0
108 n_depth=0
109
110 dataset_dir= r'C:\Users\Gabriele\Desktop\My_dataset'
111 path, subdirs, files = next(os.walk(dataset_dir))
112 for i in range(len(files)):
113     if files[i].endswith('.jpg'):
114         n_color=n_color+1
115     elif files[i].endswith('.png'):
116         n_depth=n_depth+1
117     else:
118         print('ERROR: Some data are not in jpg or png format')
119
120
```

```
121 os.chdir(dataset_dir)
122 #rgb
123 cv2.imwrite('color'+str(n_color+1)+'.jpg', color_array)
124 #depth
125 cv2.imwrite('depth'+str(n_depth+1)+'.png', depth_array)
126
127 #working dir
128 os.chdir(r'C:\Users\Gabriele\Desktop\Project')
```

Appendix C

Data split and TXT generation

Once the dataset is built, data can be divided randomly into training and testing samples, as described in Section 6.1.

```
import os
from sklearn.model_selection import train_test_split
import natsort
import shutil

dataset_dir= r'C:\Users\Gabriele\Desktop\My_dataset'
train_dir=r'C:\Users\Gabriele\Desktop\train'
test_dir=r'C:\Users\Gabriele\Desktop\test'

focal='1.93' #Intel Realsense focal length in millimeters

dataset_list=[]
color_list=[]
depth_list=[]

for path, subdirs, files in os.walk(dataset_dir):
    for filename in sorted(files):
      dataset_list.append(filename)

for i in range(len(dataset_list)):
  if dataset_list[i].endswith('.jpg'):
    color_list.append(dataset_list[i])
  elif dataset_list[i].endswith('.png'):
    depth_list.append(dataset_list[i])
  else:
    print('ERROR: Some data are not in jpg or png format')
```

```
27
28 color_list=natsort.natsorted(color_list)
29 depth_list=natsort.natsorted(depth_list)
30
31 if len(color_list) != len(depth_list):
32   print('ERROR: number of data mismatch')
33
34
35 # TRAIN/TEST SPLIT ----------------------------------------------------
36
37 X_train, X_test, y_train, y_test = train_test_split(color_list,
       depth_list, test_size=0.25)
38 #test size represents the proportion of the dataset to include in the
       test split
39
40 X_train=natsort.natsorted(X_train)
41 X_test=natsort.natsorted(X_test)
42 y_train=natsort.natsorted(y_train)
43 y_test=natsort.natsorted(y_test)
44
45 for path, subdirs, files in os.walk(dataset_dir):
46     for filename in natsort.natsorted(files):
47         if any(filename in s for s in X_train):
48             shutil.copyfile(os.path.join(dataset_dir,filename),os.path
      .join(train_dir,filename))
49         elif any(filename in s for s in y_train):
50             shutil.copyfile(os.path.join(dataset_dir,filename),os.path
      .join(train_dir,filename))
51         elif any(filename in s for s in X_test):
52             shutil.copyfile(os.path.join(dataset_dir,filename),os.path
      .join(test_dir,filename))
53         elif any(filename in s for s in y_test):
54             shutil.copyfile(os.path.join(dataset_dir,filename),os.path
      .join(test_dir,filename))
55         else:
56             print('ERROR: Selected file in dataset_dir does not
      belong to train set or test set')
57
58
59 # TXT GENERATION ------------------------------------------------------
60
61 train_file = open("My_train_file.txt", "w")
62 test_file = open("My_test_file.txt", "w")
63
64 for i in range(len(X_train)):
65   if i==len(X_train)-1:
66     train_file.write(str('/'+X_train[i]+" "+'/'+y_train[i]+" "+focal)
      )
67   else:
```

```
68      train_file.write(str('/'+X_train[i]+" "+'/'+y_train[i]+" "+focal)
        + '\n')
69
70
71  for i in range(len(X_test)):
72    if i==len(X_test)-1:
73      test_file.write(str(X_test[i]+" "+y_test[i]+" "+focal))
74    else:
75      test_file.write(str(X_test[i]+" "+y_test[i]+" "+focal) + '\n')
76
77  train_file.close()
78  test_file.close()
```

Appendix D

Extended volume estimation algorithm

Below is reported the volume estimation algorithm applied to depth maps estimated by the BTS model trained on a customized dataset. RGB images captured by the Raspberry Pi Camera inside the smart bin Nando are sent to the monocular depth estimation network that predicts the corresponding aligned depth maps.

```
import cv2
from numpy.core.defchararray import asarray
import pyrealsense2 as rs
import numpy as np
from PIL import Image as im
import matplotlib.pyplot as plt
import os

# Open Predicted depth images
directory= r'C:\Users\Gabriele\Desktop\Project\Predicted_depth'
os.chdir(directory)
back_image=im.open('back.png')
back_plus_obj_image=im.open('prediction19.png')
# Open RGB images
directory= r'C:\Users\Gabriele\Desktop\Project\Rgb_images'
os.chdir(directory)
rgb=im.open('rgb19.jpg')

# Image2Matrix
background=np.asarray(back_image,dtype=np.float32)
back_plus_obj=np.asarray(back_plus_obj_image,dtype=np.float32)

rows=back_plus_obj.shape[0]
columns=back_plus_obj.shape[1]
```

```
25
26 # Object matrix
27 obj = [ [ 0 for i in range(columns) ] for j in range(rows) ]
28 obj=np.array(obj)
29 obj=obj.astype('float64')
30 obj=np.subtract(background, back_plus_obj)  #per-pixel object's
       height
31
32 # Depth image of the background
33 fig=plt.figure(1)
34 plt.title('Estimated Background')
35 plt.imshow(background,cmap='plasma')
36 # Depth image pf the object positioned above the background
37 plt.figure(2)
38 plt.title('Estimated Back+obj')
39 plt.imshow(back_plus_obj,cmap='plasma')
40 # Depth map of the object without the background
41 plt.figure(3)
42 plt.title('Estimated Object')
43 plt.imshow(obj,cmap='plasma')
44
45
46 #ALGORITHM-----------------------------------------------------
47
48 # Invalid depth values and depth noises are set to zero
49 for i in range(rows):
50   for j in range(columns):
51     if obj[i][j]<5 or obj[i][j]>400:
52        obj[i][j]=0
53
54 # Ellipsoidal filter to isolate the object
55 x0=330
56 y0=260
57 rx=200
58 ry=180
59 for i in range(rows):
60     for j in range(columns):
61         ellipse=((j-x0)**2)/rx**2 + ((i-y0)**2)/ry**2
62         if ellipse>1:
63             obj[i][j]=0
64
65 # Object depth image post-processed
66 plt.figure(4)
67 plt.title('Estimated Object')
68 plt.imshow(obj,cmap='plasma')
69 # Object RGB image
70 plt.figure(5)
71 plt.title('RGB')
72 plt.imshow(rgb,cmap='plasma')
```

```
plt.show()

# Raspberry Pi camera sensor parameters
f=3.6                       #3.6mm
SensorLength_x=3.76         #3.76mm
SensorLength_y=2.74         #2.74mm

#mmPerPix
mm_pix_x = [ [ 0 for i in range(columns) ] for j in range(rows) ]
mm_pix_x=np.array(mm_pix_x)
mm_pix_x=mm_pix_x.astype('float64')

mm_pix_y = [ [ 0 for i in range(columns) ] for j in range(rows) ]
mm_pix_y=np.array(mm_pix_y)
mm_pix_y=mm_pix_y.astype('float64')

for i in range(rows):
  for j in range(columns):
    if obj[i][j]==0:
      mm_pix_x[i][j]=0  #to avoid useless computations
    else:
      mm_pix_x[i][j]=(back_plus_obj[i][j]*SensorLength_x)/(f*columns)

for i in range(rows):
  for j in range(columns):
    if obj[i][j]==0:
      mm_pix_y[i][j]=0  #to avoid useless computations
    else:
      mm_pix_y[i][j]=(back_plus_obj[i][j]*SensorLength_y)/(f*rows)

#Transformation in centimeters
obj=obj/10
cm_pix_x=mm_pix_x/10
cm_pix_y=mm_pix_y/10

#Volume computation
volume=0
for i in range(rows):
  for j in range(columns):
    volume += cm_pix_x[i][j]*cm_pix_y[i][j]*obj[i][j]

print('\n')
print('Estimated volume:',volume,'cm^3')
```

Acknowledgements

Per prima cosa desidero ringraziare tutto il team di ReLearn, che mi ha coinvolto in questa bellissima realtà e mi ha accolto a braccia aperte all'interno della loro famiglia. In particolare Simone e Federico che mi hanno assistito nello svolgimento della tesi e sono sempre stati super disponibili e cordiali. Ringrazio inoltre il professor Chiaberge e il PIC4SeR per avermi affidato questo lavoro.

Eccoci qui, anche questo capitolo della mia vita si sta concludendo e per quanto mi riguarda è doveroso ringraziare tutte le persone che mi hanno supportato e accompagnato in questo percorso.
Dopo la scrittura di cento pagine in inglese non ricordo quasi più come si scrive in italiano. Non è vero dai, la verità è che non sono mai stato bravo con le parole e soprattutto non sono mai stato capace ad esprimere al meglio e far fuoriuscire tutte quelle sensazioni ed emozioni che provo dentro di me. Però forse questo è il momento giusto per farlo, o quantomeno provarci.
Inizio con le persone più importanti, i miei genitori, Anna ed Enzo. Vi ringrazio con tutto il cuore, perchè mi avete sempre sostenuto e avete cercato in ogni modo di dare il massimo per aiutarmi e farmi felice. Nonostante io sia un figlio difficile da comprendere non avete mai gettato la spugna. Penso che siate dei genitori perfetti. Sento un certo peso, perchè in un futuro, qualora diventassi anche io genitore, non penso che riuscirei mai ad essere al vostro livello. Giuro, ne ho conosciuti tanti di genitori, e non ho mai pensato che ne esistessero meglio di voi. Vi voglio bene!!!
Grazie a tutta la mia famiglia. Mio fratello Leo per aver sopportato quotidianamente i miei scleri e momenti no. I miei nonni, che mi hanno cresciuto e mi hanno dimostrato ogni giorno il loro amore, questo tragurdo lo dedico soprattutto a voi! I miei zii e tutti i miei parenti, spero un giorno di poter ricambiare tutto l'affetto che mi avete dato.
Dopodichè vorrei ringraziare la mia ragazza Miriam. Mi sei sempre stata vicino addolcendo il sapore di molti giorni che senza di te sarebbero stati veramente amari e grigi. Mi trasmetti tranquillità e serenità. Sei come una casa, un luogo di pace in

cui non hai paure, in cui riesci a staccare il cervello e a goderti la vita.
Infine, ultimi ma non per importanza, i miei amici. Grazie ai miei broski del Poli, senza di voi probabilmente non sarei qui in questo momento. Nei momenti condivisi con voi, lo studio è diventato piacevole e le giornate meno pesanti. Grazie a quei pazzi dei Nvrcos e dei Loschi, con voi ho trascorso molti momenti di felicità e di spensieratezza. Nonostante i miei numerosi no dovuti ad impegni vari, voi non mi avete mai abbandonato e mi avete sempre coinvolto nelle vostre pazzie.
Grazie a tutti, ma anche grazie a me, se c'è un pregio che mi riconosco è la determinazione e la voglia di dare il massimo in quello che faccio. Al me futuro dico: "non perdere mai questa forza e combatti per ottenere tutto ciò che desideri!".

Bibliography

[1] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019 (cit. on p. 5).

[2] Tom Mitchell. «Machine learning». In: (1997) (cit. on p. 5).

[3] *Machine Learning Deep Learning Fundamentals*. URL: `https://deeplizard.com/learn/playlist/PLZbbT5o_s2xq7LwI2y8_QtvuXZedL6tQU` (cit. on pp. 6, 7, 10).

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Deep residual learning for image recognition». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778 (cit. on p. 15).

[5] Joachim Dehais, Marios Anthimopoulos, Sergey Shevchik, and Stavroula Mougiakakou. «Two-view 3D reconstruction for food volume estimation». In: *IEEE transactions on multimedia* 19.5 (2016), pp. 1090–1099 (cit. on p. 17).

[6] *Informatik IX chair of computer Vision Artificial Intelligence*. Mar. 2016. URL: `https://vision.in.tum.de/research/image-based_3d_reconstruction/multiviewreconstruction` (cit. on p. 17).

[7] Xian-Feng Han, Hamid Laga, and Mohammed Bennamoun. «Image-based 3D object reconstruction: State-of-the-art and trends in the deep learning era». In: *IEEE transactions on pattern analysis and machine intelligence* 43.5 (2019), pp. 1578–1604 (cit. on p. 18).

[8] Steffen Herbort and Christian Wöhler. «An introduction to image-based 3D surface reconstruction and a survey of photometric stereo methods». In: *3D Research* 2.3 (2011), pp. 1–17 (cit. on pp. 20, 22).

[9] Ruo Zhang, Ping-Sing Tsai, James Edwin Cryer, and Mubarak Shah. «Shape-from-shading: a survey». In: *IEEE transactions on pattern analysis and machine intelligence* 21.8 (1999), pp. 690–706 (cit. on p. 21).

[10] German KM Cheung, Simon Baker, and Takeo Kanade. «Visual hull alignment and refinement across time: A 3d reconstruction algorithm combining shape-from-silhouette with stereo». In: *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.* Vol. 2. IEEE. 2003, pp. II–375 (cit. on pp. 24, 25).

[11] Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J Gortler, and Leonard McMillan. «Image-based visual hulls». In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 2000, pp. 369–374 (cit. on p. 25).

[12] Stefano Mattoccia. «Stereo vision: Algorithms and applications». In: *University of Bologna* 22 (2011) (cit. on pp. 28, 29, 31–36).

[13] Luca Iocchi and Kurt Konolige. «A multiresolution stereo vision system for mobile robots». In: *AIIA (Italian AI Association) Workshop, Padova, Italy*. Citeseer. 1998 (cit. on pp. 27, 28).

[14] Nalpantidis Lazaros, Georgios Christou Sirakoulis, and Antonios Gasteratos. «Review of stereo vision algorithms: from software to hardware». In: *International Journal of Optomechatronics* 2.4 (2008), pp. 435–462 (cit. on pp. 31, 32).

[15] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. «Unsupervised monocular depth estimation with left-right consistency». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 270–279 (cit. on pp. 38, 42).

[16] Chaoqiang Zhao, Qiyu Sun, Chongzhen Zhang, Yang Tang, and Feng Qian. «Monocular depth estimation based on deep learning: An overview». In: *Science China Technological Sciences* (2020), pp. 1–16 (cit. on pp. 38–43).

[17] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. «Unsupervised learning of depth and ego-motion from video». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1851–1858 (cit. on pp. 39, 41).

[18] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J Brostow. «Digging into self-supervised monocular depth estimation». In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 3828–3838 (cit. on pp. 39, 41).

[19] David Eigen, Christian Puhrsch, and Rob Fergus. «Depth map prediction from a single image using a multi-scale deep network». In: *arXiv preprint arXiv:1406.2283* (2014) (cit. on pp. 44, 45, 47, 50).

[20] Amlaan Bhoi. «Monocular depth estimation: A survey». In: *arXiv preprint arXiv:1901.09402* (2019) (cit. on p. 44).

[21] Yue Ming, Xuyang Meng, Chunxiao Fan, and Hui Yu. «Deep Learning for Monocular Depth Estimation: A Review.» In: *Neurocomputing* (2021) (cit. on p. 45).

[22] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. «Deeper depth prediction with fully convolutional residual networks». In: *2016 Fourth international conference on 3D vision (3DV)*. IEEE. 2016, pp. 239–248 (cit. on pp. 45, 47, 50).

[23] Yuanzhouhan Cao, Zifeng Wu, and Chunhua Shen. «Estimating depth from monocular images as classification using deep fully convolutional residual networks». In: *IEEE Transactions on Circuits and Systems for Video Technology* 28.11 (2017), pp. 3174–3182 (cit. on pp. 46, 47).

[24] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. «Deep ordinal regression network for monocular depth estimation». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 2002–2011 (cit. on pp. 46, 48, 49).

[25] Jin Han Lee, Myung-Kyu Han, Dong Wook Ko, and Il Hong Suh. «From big to small: Multi-scale local planar guidance for monocular depth estimation». In: *arXiv preprint arXiv:1907.10326* (2019) (cit. on pp. 48, 49, 72, 76).

[26] *Camera calibrator*. URL: `https://it.mathworks.com/help/vision/ug/camera-calibration.html` (cit. on pp. 56, 59).

[27] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008 (cit. on pp. 57, 60).

[28] *Intel RealSense Camera D400 Series Product Family Datasheet*. URL: `https://www.intelrealsense.com/wp-content/uploads/2020/06/Intel-RealSense-D400-Series-Datasheet-June-2020.pdf` (cit. on p. 71).

[29] Jin Han Lee, Myung-Kyu Han, Dong Wook Ko, and Il Hong Suh. *bts*. URL: `https://github.com/cogaplex-bts/bts` (cit. on p. 76).

[30] *Installing previous versions of PyTorch*. URL: `https://pytorch.org/get-started/previous-versions/` (cit. on p. 78).

[31] *D400 Series Visual Presets*. URL: `https://dev.intelrealsense.com/docs/d400-series-visual-presets` (cit. on p. 79).

[32] *Post-processing filters*. URL: `https://dev.intelrealsense.com/docs/post-processing-filters` (cit. on p. 79).