UNIVERSITAT JAUME I

DEPARTAMENT D'ENGINYERIA I CIENCIA DELS COMPUTADORS

UNIVERSITAT JAUME·I

European Master In Advanced Robotics (EMARO)

From 3D Point Cloud to Grasping by Using Deep Learning Techniques in Underwater Domain

Candidate:

Aleks Attanasio

Supervisor:

Prof. Pedro J. Sanz

Co-Supervisors:

Prof. Enrico Simetti

Dott. Toni Peñalver

15th September 2017

Abstract

This work is based on the detection of a couple of points for optimal and robust grasping in an underwater domain. The objective is to provide a base for the development of a routine able to autonomously gather information from the surrounding environment in order to provide a robust grasp for underwater intervention robot such as the G500. For this purpose a neural network and a point cloud processing are considered: the former is meant to be able to classify shapes out of a segmented point cloud, the latter have the purpose to reconstruct those shapes out of the classified pat of the point cloud scene. The neural network has been described in its details and the routine explained step by step. Eventually results obtained on real scenes are provided.

Acknowledgements

I would like to thanks Professor Sanz that gave me the opportunity to work for this project.

A special thanks goes to my collegues from EmaroLab in Genoa Riccardo, Andrea, Alessio and Massimiliano that helped me along the project development.

I would like to thank also all those people that supported me in the last year: Aidan, Daniel, Silvio, Davide, Ilaria, Laura and Maddalena.

A particular acknowledgement goes to the lab technician Jésus without whom my work would not have been so interesting and challenging.

Finally the greatest thanks goes to my parents that gave me the possibility to undertake this journey always being present and available to listent to my concerns and my happiest moments.

Contents

Chapter 1

Introduction

In a scenario where robots are demanded to work for and with humans, one of the most challenging required task for these machines is the ability of grasping objects. Although performing such a task looks simple and immediate for a human being, many are the problems that arise in reproducing this behavior on a machine. In Robotics, a grasp is defined as a configuration of an arm, and its relative gripper, with respect to an object that guarantees a solid and robust grab. The aforementioned configuration may depend on the kind of gripper and arm that we consider. Grippers may differs in shape, dimension and number of fingers yet usually a grasping pose is defined by at least two points called *grasping points*. Any additional point used to define a pose is considered redundant and contributes to increase the grasp robustness. The learning process of the grasping function starts early in a human life: since the first day of its life a human being acquires more and more information regarding the possible grasp of objects allowing him to figure out how to grasp even those items that are new to his personal experience. We can consider the action of grasping as divided in three steps:

- Visual perception: thanks to our visual system we can detect the presence of an object in our view and determine the position in space with respect to us.
- Approaching: once the object is detected and localized our body has to move in a proper way to get closer to it
- Grasping: once we are in the proximity of the object, our arm has to reach it assuming a proper configuration such that the object would be perfectly grasped.

Once that the main task has been divided in this way, we can approach the problem of reproducing grasping capabilities on robots.

1.1 Robotic grasp

Nowadays, one of the most challenging task for a robot is to detect a possible grasp of an object in a way robust enough to allow the robot to move without dropping the item. Many are the solutions to this problem that have been proposed till now and almost all of them find their inspiration in a great functional system which is nature. Bio-mechanic world behind nature has always been inspiring for Robotics and, also in this case, it is possible to transpose the example of a human grasping to a robotic environment. In other terms we can see the grasping task as the following steps:

- Visual perception: through a vision system it is possible to detect the presence of the object of interest in the scene. This vision system is supposed to be able also to localize the object retrieving information about the distance of it from the robot. For this purpose, usually, stereo vision systems, RGB-D cameras and laser scanner are the most commonly used devices.
- Approaching: even in this case, as well as in the human one, the robot must be able to reach the proximity of the object. This operation could be performed thanks to modern control techniques that allows manipulator to reach a given position in space aligning the gripper frame to the one attached to the object.
- Grasping: differently from the human case, robots have to know a priori which is the grasping pose to perform a grasp. This is one of the trickiest part of the grasping action since even for us is difficult to explain what leads us to choose a certain pose rather than another one. In this point of the grasping process the characterization of the grasp is crucial. It is in fact necessary for the robot to know the geometry of the gripper and the position of the object. In [1] are reported different way to detect a grasping pose and as many methods to have, for example, a two-finger gripper performing a robust grasp.

Being able to reproduce these actions on a robot would open new possibilities to the application of Robotics in all kind of contexts.

1.2 Applications

More and more frequently, robots are used in critical situation such as natural disasters or dangerous accidents. In case of a nuclear incident, for example, getting close to the plant core would result lethal for a human being due to radiation. For this reason, in this kind

of critical situations, teleoperated robots are used to substitute human operators. Even in this case the whole success of the operation depends on the ability of human operators while controlling robots in the accident place. Being able to perform a task such as grasping would help operators to complete actions by automatizing part of the required tasks making the whole task more affordable. Without considering critical situation as complicated as this one, possible application are in many fields such as surgery. The recent Da Vinci Surgical System [2] is an example. This robot allows the surgeons to operate patients through small incisions and reducing invasiveness of operations thanks to small teleoperated arms. Even in this case the whole success of the operation is in the hand of the surgeon and its ability with the instruments but, introducing automated grasping behaviors would make operations easier for the human.

1.3 Problem Statement

In the last years more and more efforts have been focused on how to automatize the grasping process. As mentioned above, many are the applications of such process and, in our specific case, we will investigate the possibility of automatizing the task for sea operations. Recently, Robotics found place for its applications also in a arduous environment such as the sea, where abilities like retrieval of objects from the sea floor may be needed. For this purpose, University of Girona has designed a robot, the G500 (Figure 1.1), for sea rescue operations [3]. The robot moves in the underwater domain thanks to five motors and it is equipped with a robotic arm on which is mounted a laser scanner able to gather information about the surrounding environment in the form of a point cloud. Both the manipulation and perception systems have been developed in collaboration with the Universitat aume I.

G500 is able to autonomously navigate to reach a given position, however, by now, the manipulation maneuvers are performed manually and could require several time to be accomplished depending also on the condition of the sea. Most of the time the aim of these operations is to retrieve an object of interest that is lying on the sea floor. Simple examples of objects that could be considered of interest for these operation are archaeological evidences or, in case of water landing, the black box of a plane. The operation starts with the scanning of the sea floor looking for possible items of interest. This operation may require days to be accomplished. Once that the object has been located, the robot is controlled to reach the object and grasp it through maneuvers that could be more or less difficult depending on sea conditions. What it is required now is a method that could provide the detection of an optimal grasp of an unknown object starting from its point cloud representation.

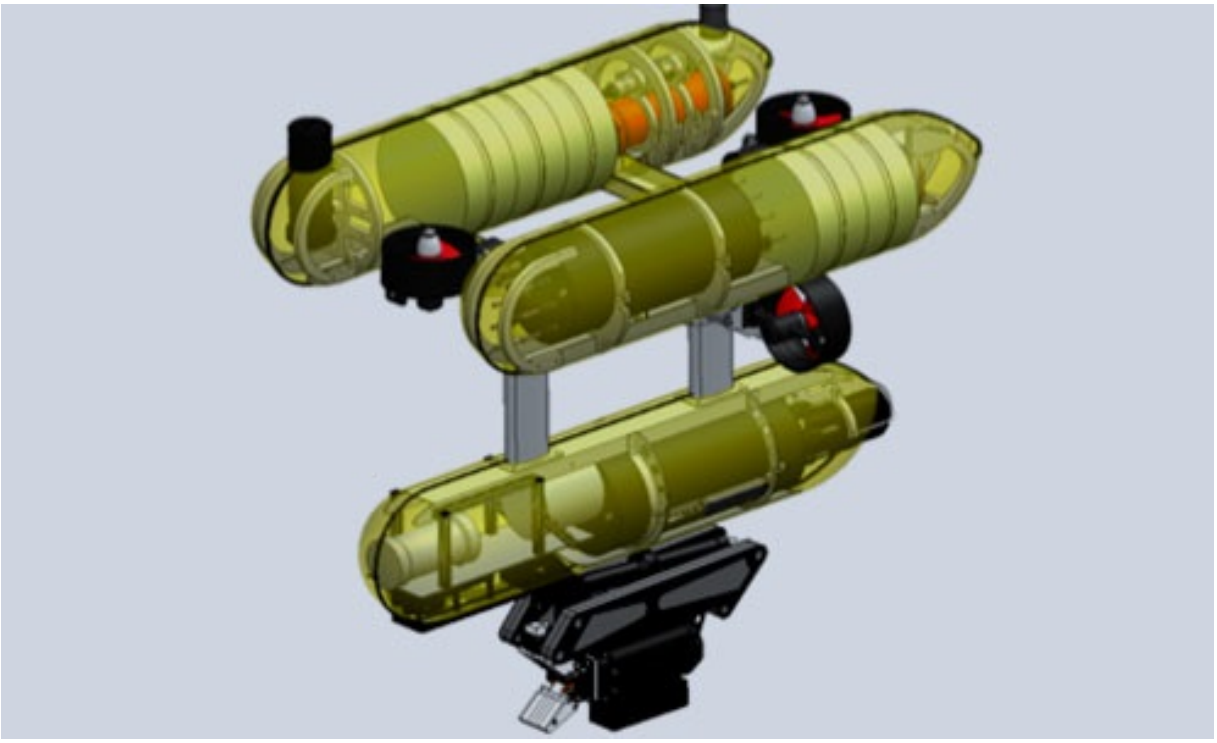

FIGURE 1.1: G500 underwater robot (photo from Universitat de Girona).

1.4 State of the Art

Detecting an optimal grasp for an object could prove to be hard depending on the situation and the condition that we consider. This leads to several approaches will be analyzed and discussed to better understand the project choices that are going to be presented later. First some basic concepts of Machine Learning and Deep Learning are provided and then some insight will be given about point cloud processing.

1.4.1 Machine Learning and Deep Learning

One of the latest, most powerful and promising branch in Robotics of recent years is Machine Learning and it provides a new approach to implement complex functions on robots and computational systems. In particular neural networks and Deep Learning [4] are being more and more used in robotics to solve every kind of task, from feature detection in image [5] to optimal grasp detection in point clouds [6]. Deep Learning, as explained in [7], is a representation learning method with multiple levels of representation. It works thanks to simple non linear modules that retrieve each information regarding a different level of abstraction guaranteeing a great success in learning every kind of function, even the most complex ones.

In the past years many were the fields in which deep learning found useful applications, especially in computer vision and image processing. Only recently has been applied to new technologies like point clouds that can be seen, in practice, as a 3D image. In [8] is proposed a method that exploits both RGB information and depth map of unknown objects to determine an optimal grasping pose for them. Here, information regarding the depth of the object is enriched by an RGB image that, having higher resolution then the depth map, allows the computer to detect more features on the target. In favor of increasing precision of the neural network, in this case the objects are presented as in a neutral scene with a white background. In [6, 9] is presented a method that exploits knowledge of complete known objects and analyzing the normals of it can detect a couple of grasping points. Even in this case, for sake of simplicity, the model are presented to the neural network as complete and not as partial as in a normal acquisition of a point cloud. The aim of this project is to use a partial point cloud, representing only one or more views of the object without having knowledge about the complete model of it. In fact, in a real scenario it would be impossible to obtain the complete representation of an unknown object to proceed as in [6, 9]. In addition to this there is to consider that on the sea floor the visibility may be compromised; therefore information such as RGB images are no longer helpful.

As far as this project is concerned, in order to implement machine learning, a Python open source library called TensorFlow [10] has been exploited. In particular this project will take advantage of the recently added functions to create three-dimensional convolutional layers to manage data in the form of a 3D matrix. Later, in 2, this aspect of the project will be analyzed more deeply, explaining better which is the structure of the neural network and which are the function providing a correct functioning of it.

1.4.2 Point Cloud Processing

As introduced before another powerful instrument recently more and more used are point clouds. Being distributions of points in a three dimensional space, point clouds can easily represent an object. The way they are acquired may vary; the most common used devices usually are RGB-D cameras (such as Microsoft Kinect or Asus Xtion) that can gather both information as RGB images and depth map. Usually, it is very typical to see both the depth map and the RGB images overlapped in the same frame. This is possible since depth maps can be seen as a gray scale image where the brightness of a single point is instead describing the distance of it from the camera. On the other hand, the resolution of point clouds is usually lower than the one from images and for this reason sometimes it could prove to be very tricky to detect certain kind of features.

Thanks to new open source instruments such as PCL (Point Cloud Library) [11] nowadays it is possible to process point clouds with ease. Many are the application but one of the main usage is in the field of grasping detection. In [12] is presented a method to create the minimum volume bounding boxes of a point cloud in order to create a model composed of boxes that would results easier to grasp. Even in this case the requirement needed to perform such task is the complete knowledge of the whole point cloud. Another interesting result has been obtained in [13] where geometry has been exploited to detect grasping pose starting from the point cloud of the object and merging it with mechanical and geometrical characteristics of the manipulator. In our case, PCL will be used mainly to process point clouds for segmenting and reconstructing models easier to handle from the user's side.

1.5 Underwater Intervention Robotics

In the past years, starting from 2009, as already mentioned in the previous sections, many were the works regarding manipulation of objects in underwater environment. In particular in Universitat Jaume I, at IRS Lab, a lot of efforts have been put in research regarding underwater robotics.

1.5.1 Reconfigurable Autonomous Underwater Vehicle for Intervention mission (RAUVI) (2009 - 2011)

The main goal of the RAUVI [14] project is to provide and improve the necessary technologies for autonomously performing an intervention mission in underwater environments. The approach can be summarized in two different steps:

1. survey
2. intervention

To start, the I-AUV explores the region of interest, taking visual and acoustic data, synchronized with robot navigation. Then, the robot surfaces, and the information is downloaded to the base station, where a computer provides a reconstruction of the explored region. By means of a specific human-robot interface, an operator identifies the items of interest and describes the task to be performed. Subsequently, the I-AUV robot navigates again to the region of interest, identifies the target object and performs the intervention task.

Therefore, the RAUVI project aims to design and develop an Underwater Autonomous Robot, able to perceive the environment by means of acoustic and optic sensors, and equipped with a robotic arm in order to autonomously perform simple intervention tasks.

Briefly, the main goals to achieve are the design and development of:

- A reconfigurable I-AUV for exploration and intervention tasks.
- A new hydraulic manipulator and its gripper.
- The control architecture and planning algorithms for manipulation.
- Visual methods for target description and recognition.

Finally, several milestones are proposed, as experimental validation, in order to gradually take the I-AUV to realistic scenarios. Initially, a real prototype will be evaluated in water tank conditions, before the final evaluation in open sea conditions.

1.5.2 Triton and Grasper (2012 - 2015)

TRITON [15] project is an on going research project which has as main purpose the development of an Autonomous Underwater Vehicle (AUV) capable of autonomous underwater interventions. The scenarios chosen for the development and experimental validation of the project are: search and recovery, permanent observatories. In Figure 1.2 is shown how these two scenarios affects the actions required to the robot to act properly in a underwater environment: on the left it is represented how the robot scans the sea floor and, once detected the object of interest approaches it in order to grasp and recover it; on the right are shown the steps necessary to move closer to an accident location in order to intervene on the problem. In the former scenario the aim of the robot is to scan the sea floor looking for a typical black box of a plane, while, in the latter, the aim is to perform all those maintenance operation characteristics of permanent marine stations that, most of the times, require pipe and valve maintenance.

Thanks to the aforementioned RAUVI it was possible to have significant improvements in underwater operation allowing research to work on three different sub-projects:

1. Cooperative Robotics, led by subproject “COMAROB” [16] at the UdG.
2. Multisensorial Perception, led by subproject “VISUAL2” at the UIB.

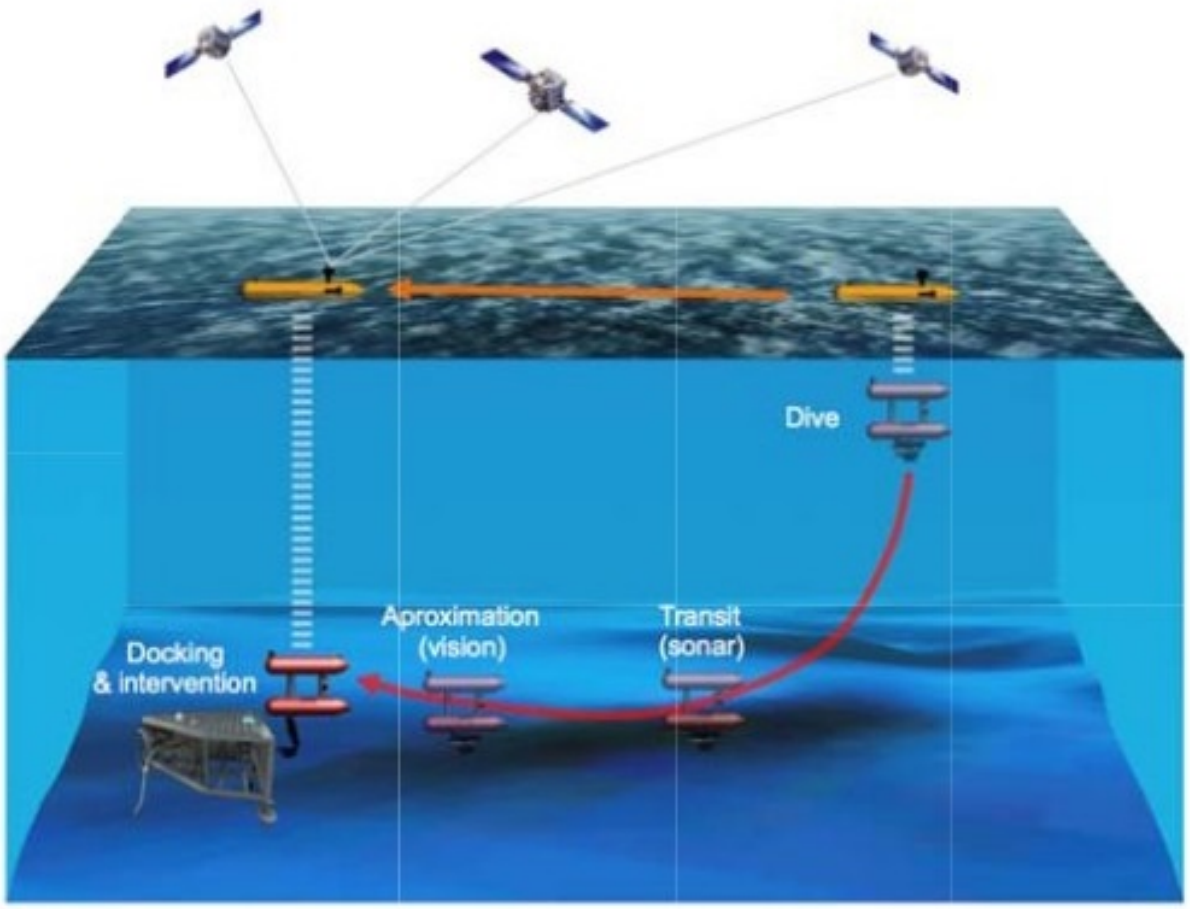

FIGURE 1.2: Permanent Observatories scenarios.

3. Autonomous Manipulation, led by subproject "GRASPER" [17] at the UJI.

For our particular interest, the GRASPER project proved to be interesting and helpful. This sub-project focuses on developing a necessary manipulation skill in order to guarantee a robust grasp for objects retrieving information about the surrounding environment through a laser scanner. Once that it is possible, via point cloud analysis to define a grasping pose that could guarantee a robust grasp of the object in the scene.

1.5.3 UWSim (2012)

In order to achieve the best results from a real situation, a simulator for underwater environments has been provided. UWSim (UnderWater Simulator) [18, 19] is a software working on ROS able to simulate the behavior of a underwater robot in order to optimize its action in a real scenario. Thanks to this simulation software it is possible to simulate natural effect such as currents, viscous friction of water and physical effects such as the

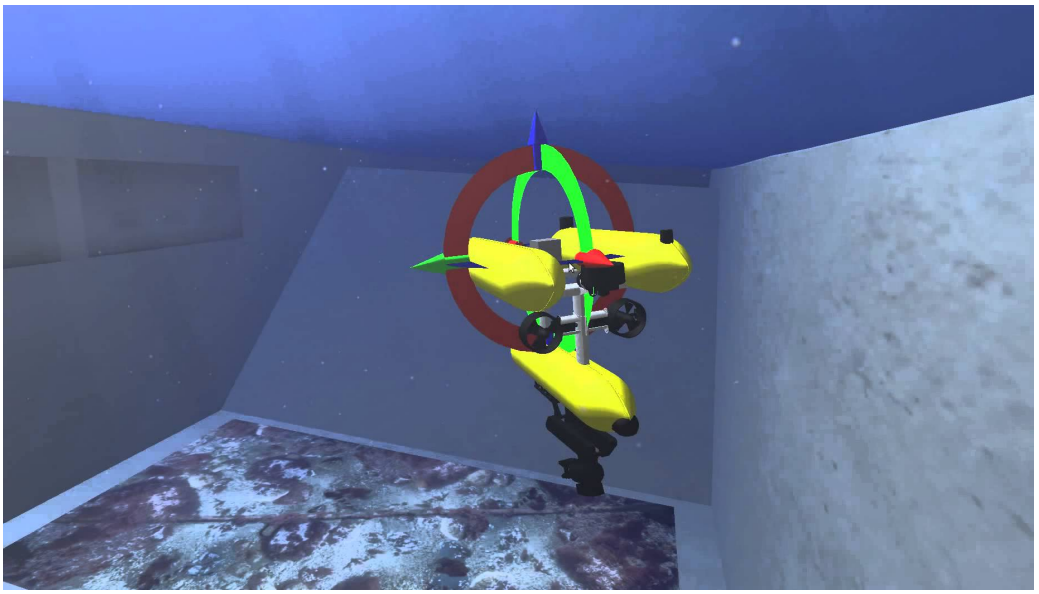

FIGURE 1.3: Screen-shot of UWSim

inertia of the robot while moving. In Figure 1.3 it is possible to see a capture of the software while the Girona500 robot is floating in a reproduction of the University of Girona pool where lots of tests took place.

1.6 Objectives

The aim of this project is to implement neural networks and Deep Learning to detect the optimal grasping points of an unknown object knowing only its point cloud representation. The main idea is to use PCL and neural networks together in order to obtain a reconstruction of the object as composed of basic geometrical shapes like cubes, cones, spheres and cylinders. Defining an optimal grasp may require the evaluation of few parameter such as the curvature of the object or the distance within the line connecting the grasping points and the center of mass [20, 21]. All these parameters are useful to define a grasping characterization of an object in order to also state what is to be considered to guarantee a grasp to be the optimal one.

As stated before, neural networks able to classify data basing on their features already exist, but few are the systems able to detect the grasping point starting only from a point cloud representation of an object that is not known. The project merges techniques from both point cloud processing and machine learning with the aim of creating a robust system able to perform the following logical steps:

- Train a neural network to classify basic shapes (cubes, cylinders, spheres and cones)

- Obtain and segment the point cloud to isolate the object from the background and to divide it into parts
- Use the neural network to classify every single part of the object to reconstruct a geometrical model
- Define the grasping points on the geometrical model

In Figure 1.4 is reported a sample of what the desired result should look like: the point cloud has been divided in parts and for each part has been detected a corresponding shape in order to obtain a complete three dimensional model of the object. Once obtained the

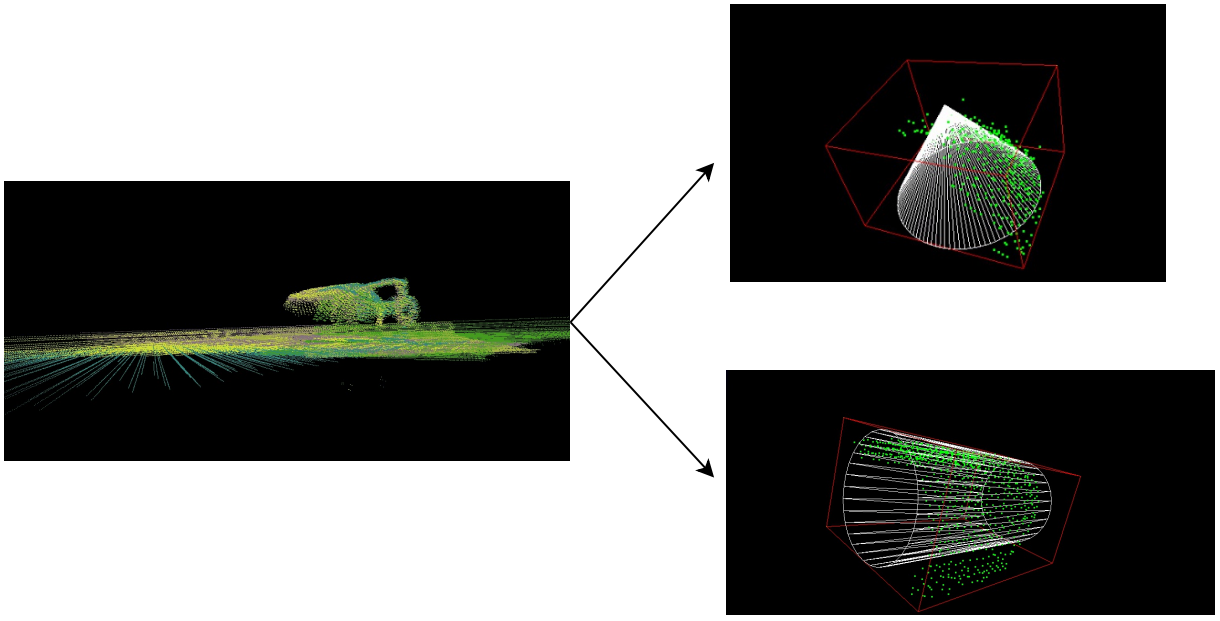

FIGURE 1.4: Example of desired result on an amphora point cloud.

model, defining the grasping points for each shape, it is possible, taking into account also the center of mass of the object, to define a couple of graping points to guarantee a robust grip.

In order to simplify the classification and the processing, objects are analyzed in an environment without cluttering. This means that in every scene there will be one or more objects properly separated among them. This decision has been taken in order not to mix different objects within them having then separated entities in a scene. In this way once the objects are separated from the background it is possible to analyze them without needing a routine able to detect overlapping of shapes.

In the following chapters will be analyzed the algorithms and concepts exploited during the developing of the project: in Chapter 2 is presented a structure for a neural network able

to classify basic shapes into four classes (cubes, cylinders, cones and spheres), in Chapter 3 is reported the routine used to process the point clouds to segment and separate them from the background and to divide them into their subparts, in Chapter 4 the results obtained in the tests are shown and, finally, in Chapter 5 are discussed the conclusions regarding the project.

Chapter 2

Deep Learning for Classification

As aforementioned in Section 1.6 to face the problem of detecting optimal grasping points the first step is to train a neural network in order to have a classifier for basic shapes like cylinders, cubes, spheres and cones. In our particular case we will exploit the process called *Deep Learning* that finds its roots in neural networks, an approach to computational problems that could lead a computer to learn from examples in a supervised or in an unsupervised way. Neural networks are inspired, as lots of concepts in robotics, to a natural and biologic system: human brain. This latter is composed of a thick network of neurons that communicate between them thanks to synaptic links. What happens during learning is that lots of these links are activated and, each time they receive a stimulus, a proper response is provided and improved step by step. Artificial neural networks work similarly, with the difference that the fundamental parts of which it is composed are not biological neurons but sigmoid neurons: a non-linear module as said before. In the next sections will be presented some basic concepts about neural networks and Deep Learning in order to have later a better confidence with the proposed solution.

2.1 Basic Concepts

2.1.1 The Sigmoid Neuron

Neural networks, as stated before, work thanks to non-linear modules that takes the name of *sigmoid neurons* which could have one or more inputs and outputs. Each incoming value will be multiplied by a weight w_j and the result will be compared to a threshold value called bias b that defines which will be the outputs of the neuron. Values such as weights and biases are defined for each neuron and may lead the neuron to "fire" or not, in other

words if the neuron will have an output or not. Considering $z \equiv \sum_j w_j x_j - b$, the sigmoid function is defined as:

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}} \tag{2.1}$$

It is possible to deduce the shape of this function simply thinking about its extremes: when z is large the output σ will be 1 , on the contrary, for small values of z we obtain a 0 output. The representation of the sigmoid transfer function is provided in Figure 2.1.

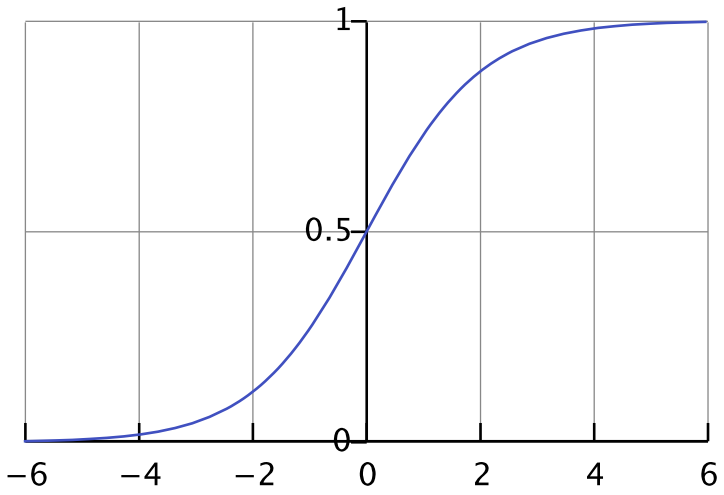

FIGURE 2.1: Sigmoid Neuron Transfer Function

This function's smoothness suggest that small changes in biases and weights causes small variations in the output, making linear the output with respect to variation of weights Δw_j and biases Δb. Exploiting calculus what comes out is that the output variation can be represented as:

$$\Delta output = \sum_j \frac{\partial output}{\partial w_j} \Delta w_j + \frac{\partial output}{\partial b} \Delta b \tag{2.2}$$

This property of the sigmoid function will be very useful during the learning phase since it can help to get closer to the desired result basing on the values of the cost function that will be explained in the next section.

2.1.2 Structure of a Neural Network

Now that the working principle of the units that compose the neural networks is defined, it is possible to talk about how the structure of a network is designed. As aforementioned the structure is composed of a high number of neurons that can provide a more complex behavior. In Figure 2.2 it is shown how generally neurons are divided into layers in a feedforward network:

- Input layer: composed of neurons receiving raw data
- Hidden layers: composed of neurons receiving data from other neurons
- Output layer: the last layer outputting the final result

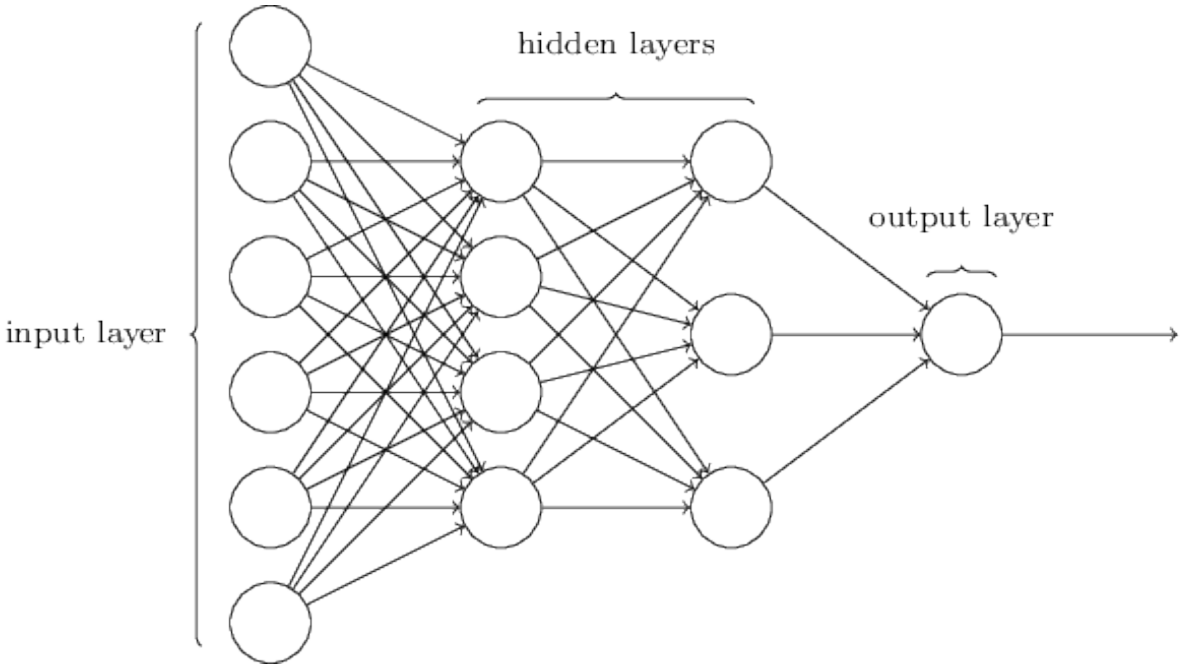

FIGURE 2.2: Classic neural network structure.

The number of hidden layers represents the complexity of the network: if it has one or two hidden layers, usually, it is said to be *shallow*; otherwise if it has more than three layers it is said to be *deep*. Concepts about depth and complexity of networks lead to the aforementioned definition of Deep Learning, the process directly related to deep networks. As said before the neurons provide a non linear function but how this behavior is linked to a possible learning has not been defined yet. Usually neural networks are used to gather high level information out of data such as images that may be of interest for a specific task. This characteristic data are obtained by a proper tuning of the inner parameters of neurons: weights and biases. As human learning, neural networks provide a better solution at each step improving the response at every iteration. To quantify how much the neural network is learning and how much it is getting closer to the desired solution a cost function is provided and it proves to be useful to determine if the network is achieving its goal or not: the higher the value of the function, the further we are from an optimal solution. Commonly, the cost function is defined as the *Mean Square Error*:

$$C(w,b) \equiv \frac{1}{2n} \sum_{x} \|y(x) - a\|^2 \tag{2.3}$$

where w and b are all the weights and biases of the network, n is the number of learning inputs, a is the output of the network when x is the input and y is the reference output. Values of x and a are given in a dataset used during the learning phase containing the learning input. Cost function $C(w, B)$, as it is possible to notice, is always positive and the closer we get to $C(w, b) \approx 0$ the better is our solution. Since reaching the exact zero of this function is almost impossible, it would be enough satisfying at least reaching the proximity of a local minimum in the cost function. In order to get closer to this stationary point some algorithms are provided: one of the most commonly used is the *stochastic descent*. Classic gradient descent works thanks to evaluation of gradients of cost function at each learning step and then calculating the next step where the gradient results negative. The number of steps is called *epochs* and the dimension of the single step *learning rate*. Classic gradient descent have the drawback of computational cost since it takes as learning inputs all the elements of the dataset. From this point of view, its stochastic form results lighter because it takes a smaller random part of the dataset and then it generalize the result. Once the local minimum or in its nearness is reached, a satisfactory neural network model could be described by the weights and biases in that point.

Tuning parameters like number of epochs and learning rate could result crucial for a satisfactory behavior: a high number of epochs with a low learning rate could lead to the possibility of never reaching the proximity of a minimum, on the other hand, a high learning rate could cause overfitting. This last is a anomalous behavior that could appear in training a network and it may be caused by several factors such as a too high number of parameters describing the model of the network, an inappropriate dimension of the dataset or a too long training phase. Since overfitting is a collateral effect, some easy operation are available to avoid it such as *cross-validation* in which a part of the dataset is used to make the network fit correctly the data. This concept will be better explained in Section 2.1.4

2.1.3 Different Kind of Layers

Designing a neural network find its main difficulty in finding a fitting solution to the particular problem. As said before the structure of neural networks may vary from problem to problem: different structures provide different solutions to the same issue. The nature of the layers, as already stated in Section 2.1.2, may be different, especially regarding the hidden layers. In this region of the network usually the most of the calculations happen and different layers provide different function to process data.

One of the most used and versatile layer is the convolutional one. This kind of layer result to be perfect for processing data in the form of a matrix such as images or, most recently, point clouds. The main working principle behind the convolutional layer find it's roots in image processing and in particular in local processing. The main feature that makes so powerful convolutional layers is the analysis by window: the whole image is scanned point by point and for every point its neighborhood is considered. The number of neighbors to be analyzed depends on the size of the window that is chosen by the network designer. By analyzing the image, or more generally the matrix, every single layer is able to retrieve information on a different level of abstraction depending also on the number of *filters* that the user wants to detect inside that patch. Thanks to these filters the convolutional layer is able to detect features and store them into a feature map which is going to be processed eventually by another kind of layer: the pooling layer that simply reduces the sizes of the map. This operation leads to have a smaller and more concentrate map where local groups of feature, usually highly correlated between them, are going to be collected and interpreted as motifs for further analysis in subsequent layers.

Since the only limit in this kind of structure is the memory required to store all the data, potentially speaking there is no constraints on the size of the networks. This leads to have even more deep networks that can provide, reporting what said before, even more complex function. A great example is reported in [22] where a deep network (almost 25 layers) has been designed to recognize different objects from images acquired by a camera.

2.1.4 Dataset Structure

In order to provide a satisfying response neural networks need many different examples that are usually called *dataset* and its structure may vary depending on the kind of learning that we want to provide: supervised learning requires labeled data, unsupervised learning don't need labels to work. Data then can come in two forms:

- labeled: to each data is related its own target (for example in images classification problem every example has it's own class label)
- non-labeled: data have no target but a pre-training network is provided to extract feature and define classes by itself.

As aforementioned datasets are supposed to be as big as possible to avoid overfitting. For this reason, usually, the dataset is divided in three parts:

1. **Training Set**: used to find values of weights and biases to fit the data. Usually it is composed of the 70% of the dataset
2. **Validation Set**: used to validate parameters from the first stage and to avoid overfitting. Usually the validation set is the 20% of the dataset
3. **Test Set**: used once the model has been trained to asses the effectiveness of the training. Usually it takes the 10% of the whole dataset.

Also in this case, as it was for the parameters defining the network, the dataset can be split arbitrarily by the user in the percentages he feels more confident with.

One of the hardest part in working with neural network is finding a proper dataset, big enough and containing enough information to allow the network to generalize. Usually it is possible to find online these datasets that have been already used by others but if no one of the present datasets contain the information needed, it may be necessary to build up a personal dataset for further analysis. Depending on the kind of information, creating a dataset could be tricky and this aspect of neural networks will be presented later in Section 2.2.1 where is going to be shown how a dataset for shape recognition could be built.

2.2 Classifying Shapes with a Deep Network

As mentioned before in Section 1.6, the aim of this project is to define geometrical models of an object starting from its point cloud. In this Section the approach to classification problem with Deep Learning will be presented showing the structure of both the dataset and the network used. At the end, results of training will be presented.

2.2.1 Dataset

Since the objective of this network is to classify object, what we will need is basically a labeled dataset of point clouds representing shapes in space. Seeing that objects in point clouds are partially represented because some parts of them could be concealed to the camera, the dataset needs to be as much similar as possible to the real case. For this purpose, a C++ script generates shapes seen by a random point of view resulting, therefore, partially complete. Since the neural network will be trained via supervised learning, the dataset need to be labeled. A very classic way to classify objects is using a one-hot vector having as size the number of classes where a label is defined by a one

value along the vector. In our case the number of classes is four since the aim is to classify objects within the four types of shapes (cube, cone, cylinder, sphere). Thus every single shape in the dataset is labeled with a vector representing its classification. In Figure 2.3 is shown how the shapes are generated and labeled.

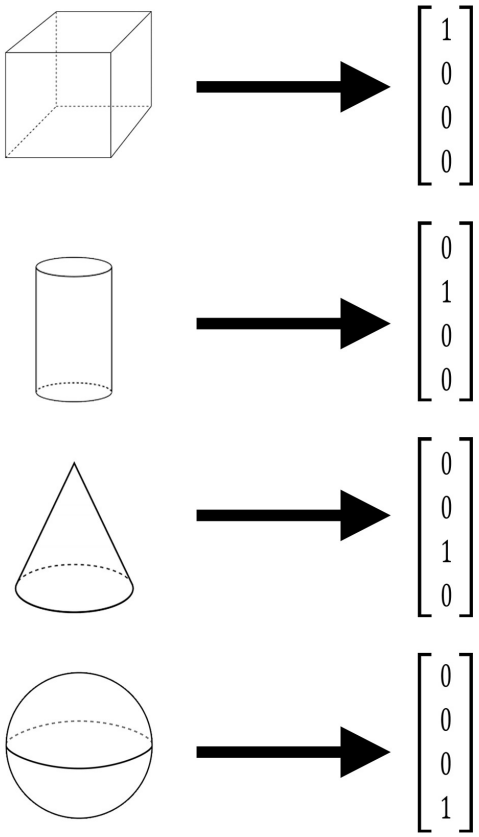

FIGURE 2.3: Generation and labeling of dataset.

Once generated the shapes, due to memory issues that could lead to excessive computational effort, it is required to reduce the size of point clouds representing them using voxels, a volumetric unit of measure similar to pixel for images. This transform divides the space of point clouds into a three dimensional matrix and, for every element of it, a check is performed: if inside the element subspace is present at least a point or a group of points, than that space will be labeled with a one, if not its label will be a zero. This method provides a good shrinking in size of clouds without distorting them and, depending on the size of the matrix that we use to divide the space, we can have voxels representation with more or less details. Obviously applying this kind of compression causes a loss in resolution of the cloud but, since the shapes used are very basic this creates no losses of generality in the results. In Figure 2.4 is reported an image from [23] of a scene with the overlapped matrix used to create the voxel representation of it. In this case the scene has

lots of detail and the matrix dimension used will only make possible to basically recognize the human shapes without any further analysis on the context.

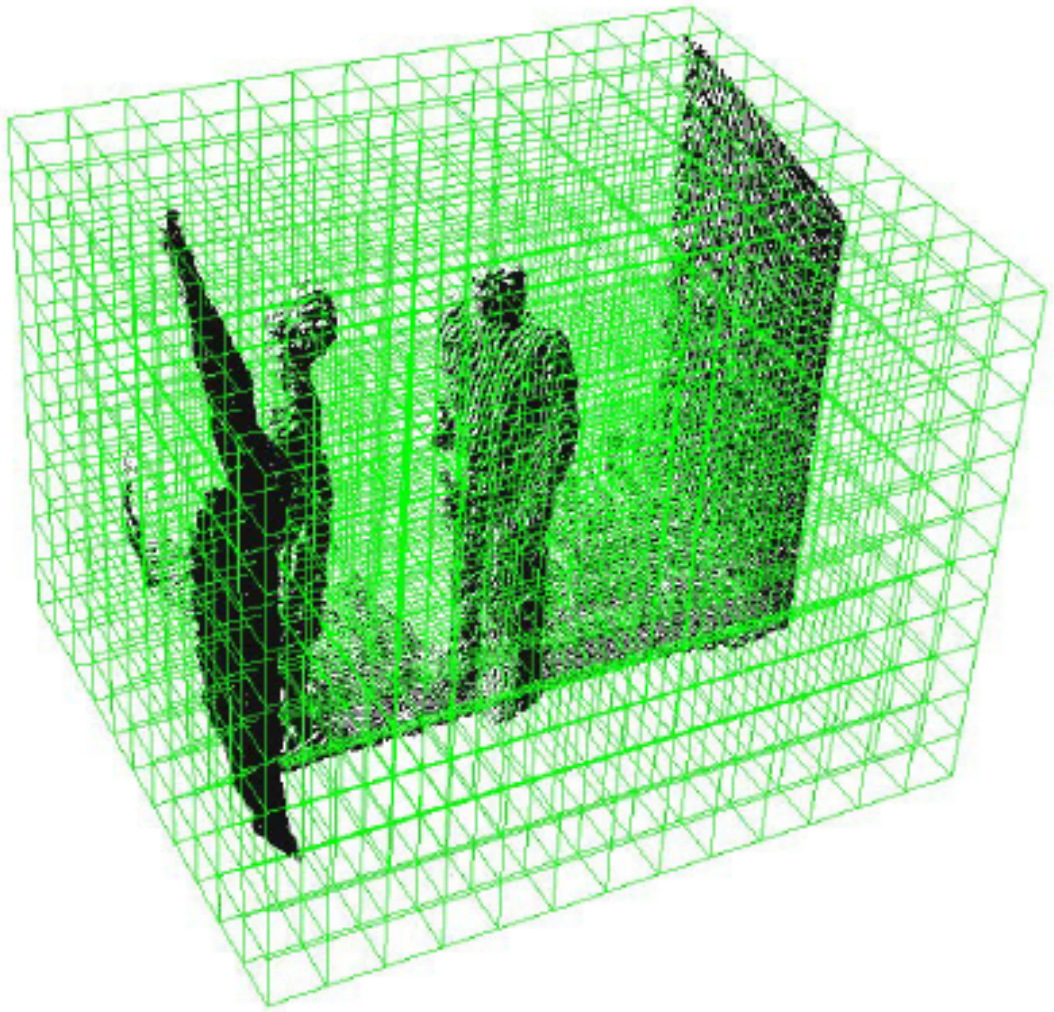

FIGURE 2.4: Voxel representation of a scene.

The voxelization process has to be applied also to clouds that we're going to input into the neural network. This operation is slightly different from the generation of the dataset and it will be analyzed more precisely later in Section 3.2.4.

2.2.2 Neural Network Structure

In order to create the classifier neural network we exploited Tensorflow [10], a Python open source library developed by Google including all the required function to create and customize a neural network. In this section will be presented the structure of the network

describing every single layer composing it. Creating a good neural network could be tricky since there is no clue on how to create it and which is the optimal solution. This is due to the fact that, by now, it's still not clear what theoretically happens within a neuron. For this reason, a good structure to solve a problem could prove not to be fit for other issues, even if the problem is similar. For this reason, before obtaining the structure that we will call from now on *optimal*, few trials are required. What is going to be shown in this Section is the structure of the optimal network that makes it able to classify objects with high reliability.

In Figure 2.5 is reported the structure of the network as a graph obtained through Tensorboard, an interactive platform with GUI to display results of trainings.

Starting from the bottom it is possible to explore, layer by layer, the topology of the network. At the beginning placeholders containing the weights and biases are created and initialized with random values that will be updated during the first iteration. Once created placeholders and imported data from the dataset, the structure of the network is composed of:

1. *conv1_class*: Convolutional layer with Pooling layer
2. *conv2_class*: Convolutional layer with Pooling layer
3. *conv3_class*: Convolutional layer
4. *dense1_class*: Densely Connected layer
5. *lineal1_class*: Densely Connected layer

Tensorflow works thanks to tensors used as representation of layers output and offers many different functions to customize the structure of the network by changing parameters of every single layer.

Regarding convolutional layers the first choice is to chose which kind of convolution is needed. In our case, since data comes in form of a 3-D matrix, the 3-D convolutional layer proves to be the most appropriate for our structure. Actually with this kind of layer it is possible to scroll the matrix in every dimension granting the possibility to detect the most of features. For sake of clarity, in Figure 2.6 is shown how a kernel moves within the space of the matrix, in this case the kernel is represented in red and the total matrix in black.

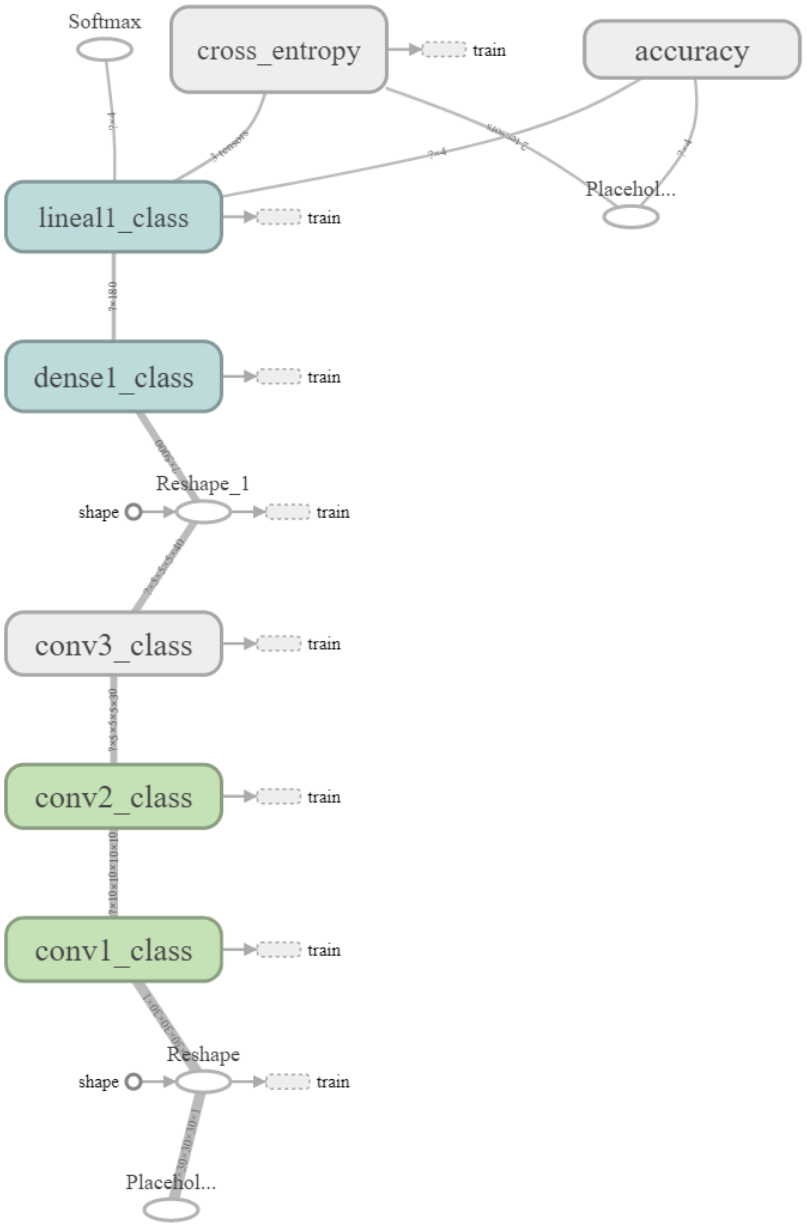

FIGURE 2.5: Graph of the network used.

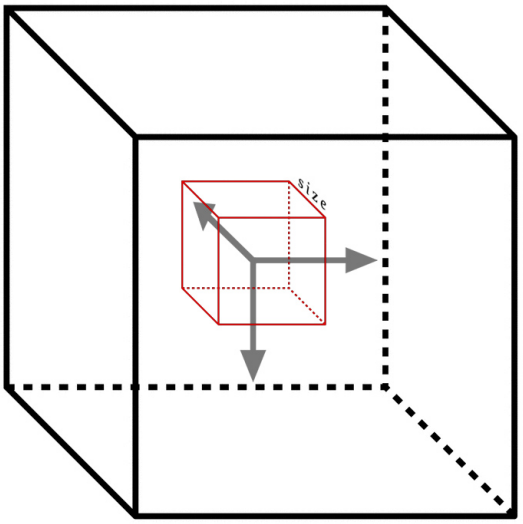

FIGURE 2.6: Representation of a kernel moving.

Once a convolutional layer is defined there are few parameters to tune:

- Activation function
- Size of kernel
- Number of filters
- Padding
- Stride

The activation function determines if the neuron will fire or not as a result of the incoming input. Usually the rectifier, that looks like a ramp, is the most common function used as activation and is called ReLU (Rectifier Linear Unit) in is application. In order to be more precise, the definition of the ReLU function is:

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{2.4}$$

where x has to be considered as the input of the neuron. In Figure 2.7 is shown the rectifier function.

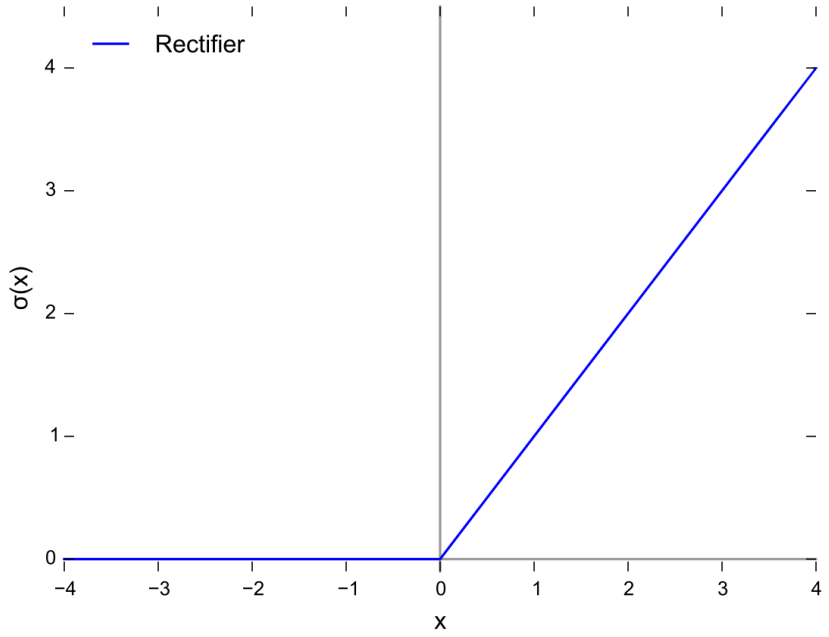

FIGURE 2.7: Rectifier function (ReLU).

This is not the only function available as activation function but there are many such as the Idenity that will be used later.

The size of kernel and the number of filters define how big is he scrolling square and how many filters to use for each moving patch. These subregion of the space will contain features that, once detected, are used to create a map containing all of them. Feature maps are shrunk later by the pooling layer in order to make similar and close feature more related within each other.

On the other hand, the value of Padding can be *"same"* or *"valid"*: the former will add no values to the edges of the output tensor, the latter will add a proper number to make fit the kernel size to data.

Finally, the Stride value means the step to use to apply the matrix of the kernel. To make it clearer in Figures 2.8, 2.9, 2.10 are reported three examples of striding in a two-dimensional space to make it simpler to visualize. In Figure 2.8 is shown a case in which there is no overlapping of the patches with a Stride dimension of $[2, 2]$ (red crosses, one every 2 steps in each dimension) and a kernel size of $[2, 2]$ (blue windows). Decreasing the number of stride at $[1, 1]$ (one cross every step in each dimension), keeping the same size of kernel, the result obtained is shown in Figure 2.9 where overlapping of kernels is clear. Another case is in Figure 2.10 where the stride dimension is $[4, 4]$ (one cross every 4 steps in each dimension) and the kernel size is the same: here we have no overlapping of region

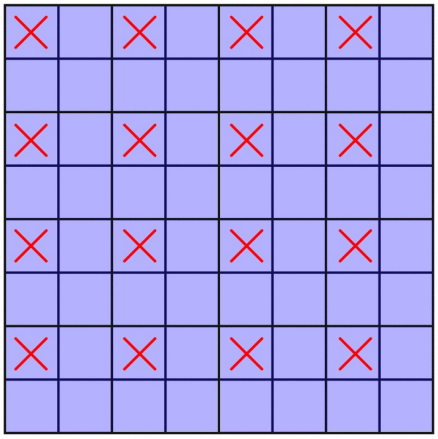

FIGURE 2.8: Stride = $[2, 2]$.

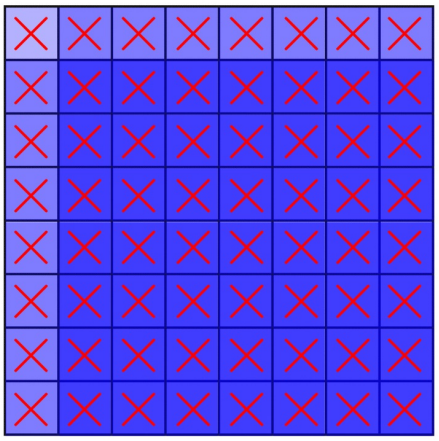

FIGURE 2.9: Stride = $[1, 1]$.

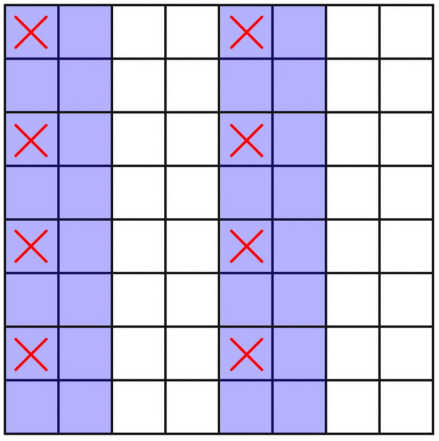

FIGURE 2.10: Stride = $[4, 4]$

but there are few regions that are not covered by the patches. Usually the most common case is the first one in Figure 2.8 where all the matrix is covered without overlapping that results to be a waste of time and computational cost. The reason why it is possible to overlap patches is because, in few cases, could be required to analyze more deeply the features contained in the dataset. In that case, a small Stride dimension will increase the training time but would have more precision in those cases where features are a lot and all close within each other. On the contrary the case shown in Figure 2.10 performs better training in those situation where features are sparse. In fact, being not needed to find lots of features, to improve performance it is possible to choose not to analyze all the matrix. In addition, in this case, in order to make it visually more clear,the Stride and patch dimensions have been chosen in order to have them in the shape of squares; however it is possible to arbitrarily select the dimension to fit different data with proper dimensions.

Once defined these parameters it is possible to discuss about the structure of layers in all their details. In Table 2.1 is reported, layer by layer, the characteristics of the network. Since some values have no sense applied to a certain kind of layer, a "—" is written in these case. In order not to encumber the table, the input layer and the output layer has been left apart. In Section 2.2.3 will be described better the output layer and its function.

TABLE 2.1: Structure of the neural network used

Layer	Activation	Kernel Size	Number of Neurons	Padding	Stride
Convolutional	ReLU	[5, 5, 5]	10	Same	[1, 1, 1]
Pooling	—	[3, 3, 3]	—	Same	[3, 3, 3]
Convolutional	ReLU	[5, 5, 5]	30	Same	[1, 1, 1]
Pooling	—	[2, 2, 2]	—	Same	[2, 2, 2]
Convolutional	ReLU	[3, 3, 3]	40	Same	[1, 1, 1]
Densely Connected	ReLU	—	180	—	—
Densely Connected	ReLU	—	4	—	—

As it is possible to observe from Table 2.1 the structure of the network presents three convolutional layers, two pooling layers and two densely connected layers. Pooling layers have a different function from convolutional ones: mainly pooling layers scroll the feature maps extracted by the convolutional layer but, instead of looking for features, they evaluate the maximum of the elements in the kernel and shrink the whole patch to a single element having as value the maximum found before. This process is called *maximum pooling* and allows the features in the map to be closer within each other whether they prove to be related. The structure of the network will be now analyzed layer by layer considering every single parameter.

As aforementioned, most of these layers has as activation function the rectifier introduced before (see Equation 2.4). However, the output layer, not reported in the table, has a different function that gives also the name to the kind of layer: Softmax Layer that will be shown in details in Section 2.2.3.

Regarding the size of the kernels there is nothing much to say but that the second pooling layer changes dimension from $[3, 3, 3]$, of the first pooling layer, to $[2, 2, 2]$ in the second one, in order to have an even pooling. In fact as shown in Figure 2.11 the size of input changes after the first pooling from $30 \times 30 \times 30$ to $10 \times 10 \times 10$ since the pooling size is $[3, 3, 3]$. For this reason, in order not to have odd divisions the second pooling layer provides a $[2, 2, 2]$ kernel thus obtaining a $5 \times 5 \times 5$ output.

Further analysis of the structure lead to the number of neurons for each layer. Convolutional layers, as mentioned in Section 2.1.3, are able to retrieve information by analyzing maps of feature in the matrix. In our specific case an ascending number of neurons has

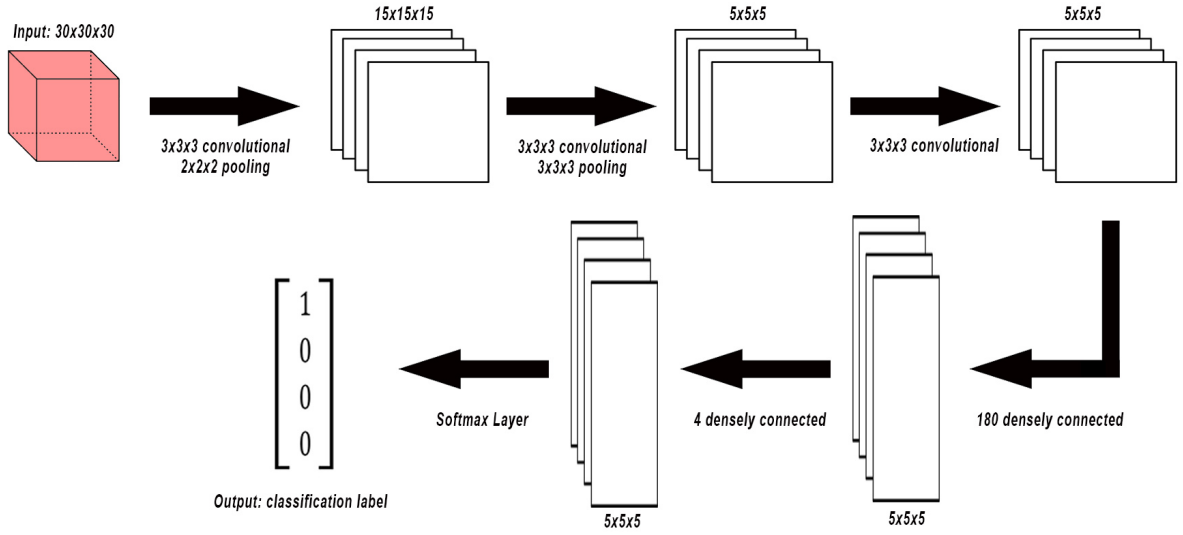

FIGURE 2.11: Network structure with highlight on dimension of data.

been set for the convolutional networks in order to achieve a sufficient level of abstraction to generalize characteristics and features of the four classes of shapes that are represented with a confidence value by four neurons in the last densely connected layer.

Continuing the analysis of the table it's possible to observe that all the padding variables has been set to *same*, therefore no values are added to the boundaries of the matrix. This choice has been taken because the shapes that are going to be considered will never be that big to occupy the whole matrix. For this reason, it's rare to find a case in which some features are on matrix's boundaries thus the border effect will not affect the results.

Finally, strides values simply state that there is overlapping of the kernels in the convolutional layers but not in the pooling ones. This is a common decision in designing neural networks regarding pooling layers, in fact, while it is required convolutional layers to scan the whole image (or matrix) looking for every possible feature, this requirement is not desired on pooling layers that are expected to concentrate the feature map without neither distorting nor confusing it.

Now that the structure of the network is well defined the next step is to define the details of the learning phase, in other words the settings with which the network will perform learning. In Section 2.2.3 is reported how the network was set to learn and which where the last parameters to be tuned to complete the learning of the four classes to implement a classifier.

2.2.3 Learning Phase

In the previous section has been largely described the topology of the network. In this section will be presented the functions used to make actually the train happen.

As previously said, machine learning exploits the power of gradients and other mathematical operators to explore a cost function looking for its minimum. This leads to the decision about which kind of cost function to use and which kind of optimizer to use in order to find it's minimum. In our case, since the objective is to classify shapes, the Cross Entropy combined with a Softmax layer proved to be effective for classification. For this reason we are going to be explain in detail these concepts in the next section.

2.2.3.1 Softmax over Cross Entropy

In Section 2.1.2 has been reported the importance of the cost function related to the actual learning of a neural network. Reporting what stated before, usually the cost function is defined as in Equation 2.3. However, it is possible to define different cost functions depending on the kind of network that we are using. Choosing the most suitable cost function may be decisive in defining the performance of the network, in fact it can determine the speed with which the network will reach the desired results.

In our case of study, in order to improve the speed in learning, has been introduced *Cross Entropy* as cost function defined as:

$$C = -\frac{1}{n}\sum_{x}[yln(a) - (1-y)ln(1-a)] \tag{2.5}$$

where n is the number of training items, the sum is over x training inputs, y is the desired output and a is the actual neural network output. In order to being able to define this function as a cost one it is necessary that it satisfies at least two requirements:

1. It has to be always defined positive ($C > 0\ \forall x$)
2. It has to be $C \approx 0$ for $y \approx a$

In other words: the cost function can't be negative and its value gets closer to 0 whenever the output of the network is getting similar to the desired one. In this case both the requirements are satisfied, in fact, since both the logarithms terms are all in the range between 0 and 1 and the other terms in the sum are negative, having a minus at the beginning of the sum, C can't have negative values. In addition to this, considering that

in our case, for classification, the values of output stays in the interval within 0 and 1, it is possible to see how considering values such as $y = 0$ and $a \approx 0$, or $y = 1$ and $a \approx 1$, the cost function tends to the zero. Having satisfied the requirements to define Equation 2.5 as a cost function, it will be shown now which are the advantages that Cross Entropy provides in place of the simple Quadratic Error cost function. Actually the previous seen Quadratic Cost may result slow during learning, especially in the unfortunate case in which the initial error is big. In that case, being the weights and biases updating basing on the partial derivative of the cost C with respect to weights and biases, $\partial C/\partial w$ and $\partial C/\partial b$, it may occur the case in which, for the first epochs, the cost function slightly decreases remaining more or less around the same value. Only once that the cost function reaches a cliff, the partial derivatives of C start to have significant values to reduce as much as possible the cost function.

On the other hand, Cross Entropy provides a solution to this problem that could be noticed only once analyzed the partial derivatives of it with respect to the weights and the biases. In fact considering the partial derivative $\partial C/\partial w$ with $a = \sigma(z)$:

$$\begin{aligned} \frac{\partial C}{\partial w} &= -\frac{1}{n}\sum_x \left(\frac{y}{\sigma(z)} - \frac{(1-y)}{1-\sigma(z)}\right)\sigma'(z)x_j \\ &= -\frac{1}{n}\sum_x \frac{\sigma'(z)x_j}{\sigma(z)(1-\sigma(z)}(\sigma(z)-y) \end{aligned} \tag{2.6}$$

Now given that $\sigma'(z) = \sigma(z)(1-\sigma(z))$ the previous equation becomes:

$$\frac{\partial C}{\partial w} = \frac{1}{n}\sum_x x_j(\sigma(z)-y) \tag{2.7}$$

where is clear that the greater the error the faster the function will tend to zero. This advantage is also given to the fact that $\partial C/\partial w$ does not depend on $\sigma'(z)$ that being small would slow down the function in reaching the zero. A similar operation could be performed on $\partial C/\partial b$ obtaining:

$$\frac{\partial C}{\partial b} = \frac{1}{n}\sum_x (\sigma(z)-y) \tag{2.8}$$

where again it is possible to see the $\sigma'(z)$ disappearing and allowing the function cost to be fast in tending to zero. Thanks to these properties, properly sought-after, the Cross Entropy prove to be a versatile and optimal solution for learning especially for classification cases.

Another fundamental concept of which the project takes advantage is the *Softmax function*, usually implemented in the so called Softmax layer. In general this kind of layer is placed as output layer of a neural network and used to implement classification. It is in fact able to to shrink a K-dimensional vector z of real values to a K-dim ensional vector $\sigma(z)$ of real values in the range $[0, 1]$ which added returns the value 1. The Softmax function has been defined as

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad for j = 1, \ldots, K. \tag{2.9}$$

Thanks to this it is possible to interpret the values of $\sigma(z)$ as confidence values related to the considered class. In our case, since the classes are basically four the $\sigma(z)$ vector will have four output values indicating the confidence with each class. In other words the results of the Softmax layer is a distribution of probability, the probability that the current instance is belonging to a class rather than another one.

Combining Cross Entropy and Softmax is one of the most common and effective way to build a classifier having as input an instance of an object to classify and as output its values related to the probability of the object to belong to each class. For example, in our case of study, it is expected a cube to have a classification output vector having a 1 on the confidence value related to the cube class. However, since usually reaching a 0 in the cost function is quite impossible, we will never obtain a confidence value of 1, yet we will have hopefully a peak value on the corresponding class.

2.2.3.2 Adam Optimizer

As previously introduced in Section 2.1.2 during the training of a network is necessary to provide a function able to understand which is the direction of the next step along the loss function in order to find its minimum. Antecedently has been introduced the most used optimization algorithm based on the evaluation of loss function gradient intuitively called *Gradient Descent*. Even in its stochastic form, this algorithm does not guarantee the best performance on training and sometimes it may require more time. For this reason more optimizers have been provided to guarantee the best performance in optimization. The algorithm used in this project is called *Adam Optimizer* [24] that stands for adaptive moment estimation. It's advantage consists in being able to evaluate an adaptive learning rate for each weight instead of Gradient Descent Algorithm that keeps that value fixed. In addition to this Adam algorithm could be considered a mixture of two different optimization algorithm:

- **Adaptive Gradient Algorithm (AdaGrad)** [25]: used mainly in computer vision provides per-parameter learning rates updating them making this algorithm versatile with problem presenting sparse gradients.
- **Root Mean Square Propagation (RMSProp)**: provides learning rates adapted on the basis of averages of recent magnitudes of the gradients for the weights.

Basically Adam takes the advantages of both not only considering the mean as in RMSProp, but also it takes into account the variance. Recently Adam proved to be very effective applied to Deep Learning problems such as classification. In [24] it has been tested on the MNIST [26] classification problem and on CIFAR-10. Due to its versatility and effectiveness, Adam optimizer has been chosen as optimizer for this project.

2.2.4 Results

In this section will be presented the results obtained taking advantage of the previous neural network structure and parameters set. The reported results are the best that have been obtained during the test and are relative to the previous described neural network.

The dataset used has been generated thanks to a C++ script introduced in Section 2.2.1. This latter is able to generate a dataset given the proportion of the amount of shapes so that it is possible to create dataset with more, or less, number of a specific shape. In addition it is possible to chose the amount of shapes contained in the training and in the validation set. The shapes are created as point cloud and then converted into their respective voxel representation. Finally every single shape, now converted into a three dimensional matrix, is saved as a text file reporting every single element of the matrix. In our case the dataset is composed of:

- 3000 shapes for the training set
- 800 shapes for the validation set
- 300 shapes for the test set

with the following proportion: every 8 cubes are generated

- 8 cylinders
- 4 cones

- 6 spheres

since, empirically, it's possible to observe that cylinders and cubes occur more frequently than cones and spheres.

After few tunings of parameters, the neural network has run for 16 hours and around 600 epochs improving the precision from 20% to around 85%. This means that for each batch of the dataset composed of 150 elements the 85% were correctly classified. In Figure 2.12 and 2.13 are represented respectively the graph of the loss function and of the output precision.

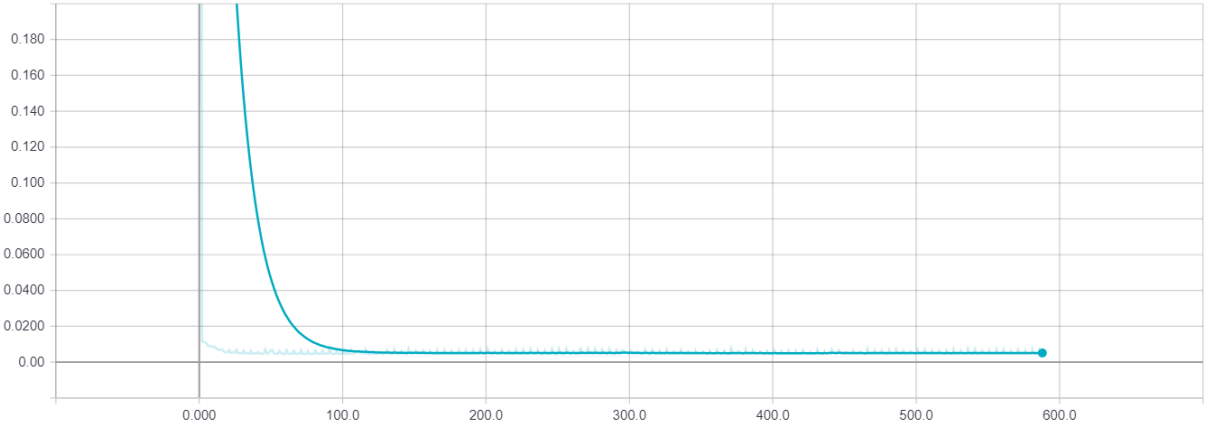

FIGURE 2.12: Loss function.

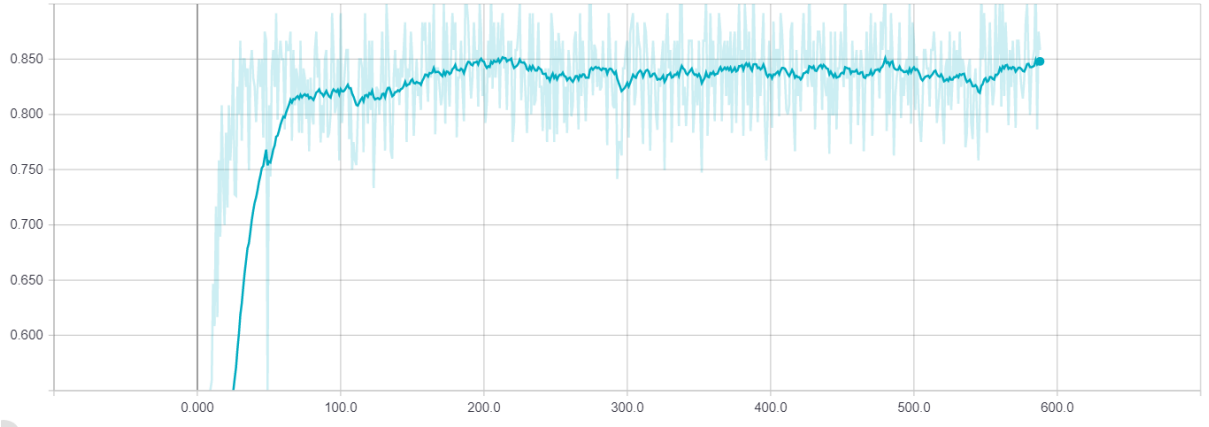

FIGURE 2.13: Accuracy of the network.

The curves have been obtained thanks to TensorBoard, a tool implemented in TensorFlow that provides a graphic interface to better show what actually is happening inside the network. The graphs have been smoothed to better appreciate the dynamics of them. In Figure 2.12 is shown how the cost function decreases in time and tends to zero. On the other hand Figure 2.13 shows how the accuracy of the network has an ascending trend

that reveals how the network is learning effectively how to classify objects. In fact in this case, to evaluate accuracy, has been chosen the percentage of correct prediction out of the total ones.

Later, in Chapter 4 will be reported the results of the network fed with the output of the point cloud processing that will be now presented in Chapter 3.

Chapter 3

Point Cloud Processing

In previous sections it has been presented a neural network able to recognize basic shapes out of their point cloud and voxels representation. As introduced in Chapter 1 the next step after shape classification is a point cloud processing routine able to reconstruct the object using the classified basic shapes. The reconstruction problem has been assessed taking advantage of few features retrievable from a partial point cloud such as curvature values, center of mass and orientation. In the following section will be presented an approach to perform such reconstruction in order to have a simpler object representation on which easily detect the optimal grasping points.

3.1 Definition and Characteristics of a Point Cloud

Before starting to describe the routine outlines is fundamental few concepts to be clear in order to fully understand the steps of reconstruction. First of all is necessary to define what a point cloud is: a point cloud is a 3D representation of a scene composed of points defined with their coordinates along the x, y and z axis in space. The result of this is a scene such as the one in Figure 3.1 where it's possible too see what could be the sea floor with an amphora and a small stone.

The point cloud representation of a single view of a scene is also assumable as a gray scale image where, instead of considering the intensity of light for a single point (pixel), we consider the depth value related to that point in the scene. Having clear the basic structure of a point cloud it is now allowed to introduce the first concepts related to the processing of this kind of data structure. For the purposes of this project the operators that will be used are mainly local operators. In other words, the point cloud processing

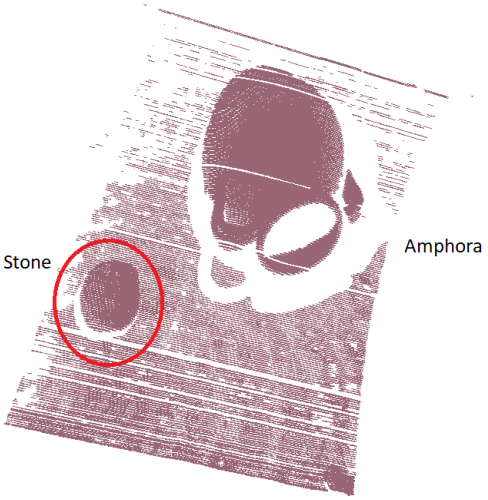

FIGURE 3.1: Point cloud of a scene with an amphora and a stone.

will be always performed along the whole scene and not considering point by point. These kind of local operators allow us to evaluate features such as the curvature of the cloud and difference of normals (DoN) that will be explained in details in Section 3.2.2. In Figure 3.2 is shown a point cloud where the values of curvature are represented by color: the more the color gets closer to the blue, the higher is the curvature value relative to that point. In this case the point cloud represents the head of the typical bunny sample used for PCL testing.

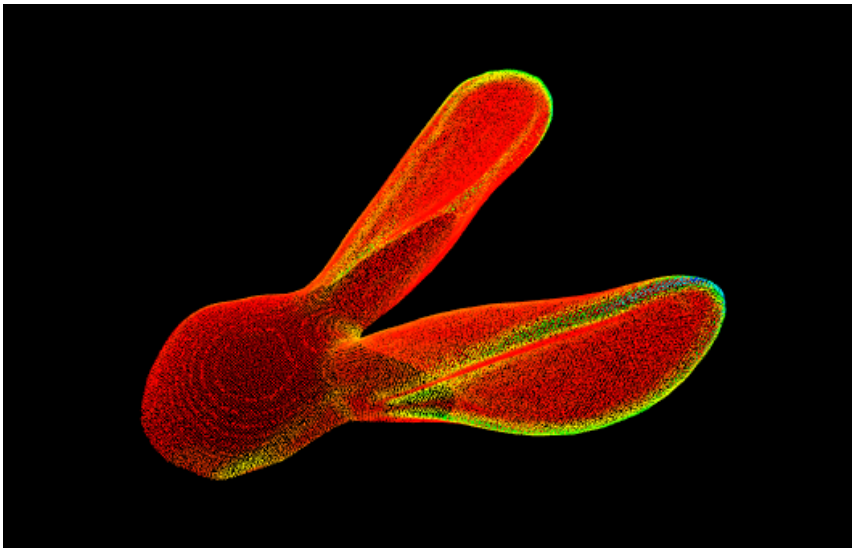

FIGURE 3.2: Curvature values along a point cloud.

In the following sections will be explained more precisely every step that involves point cloud processing in order to obtain a couple of points on it for an optimal grasp.

3.2 PCL Processing Routine

In this section will be shown step by step the routine represented in Figure 3.3 as a flowchart that allows the program to detect the grasping points starting from the point cloud obtained, as mentioned in Chapter 1, via laser scanner. In blue are reported the data input and output of the routine functions, in orange are the functions related to point cloud processing and, finally, in green are reported all the processes performed by the neural networks.

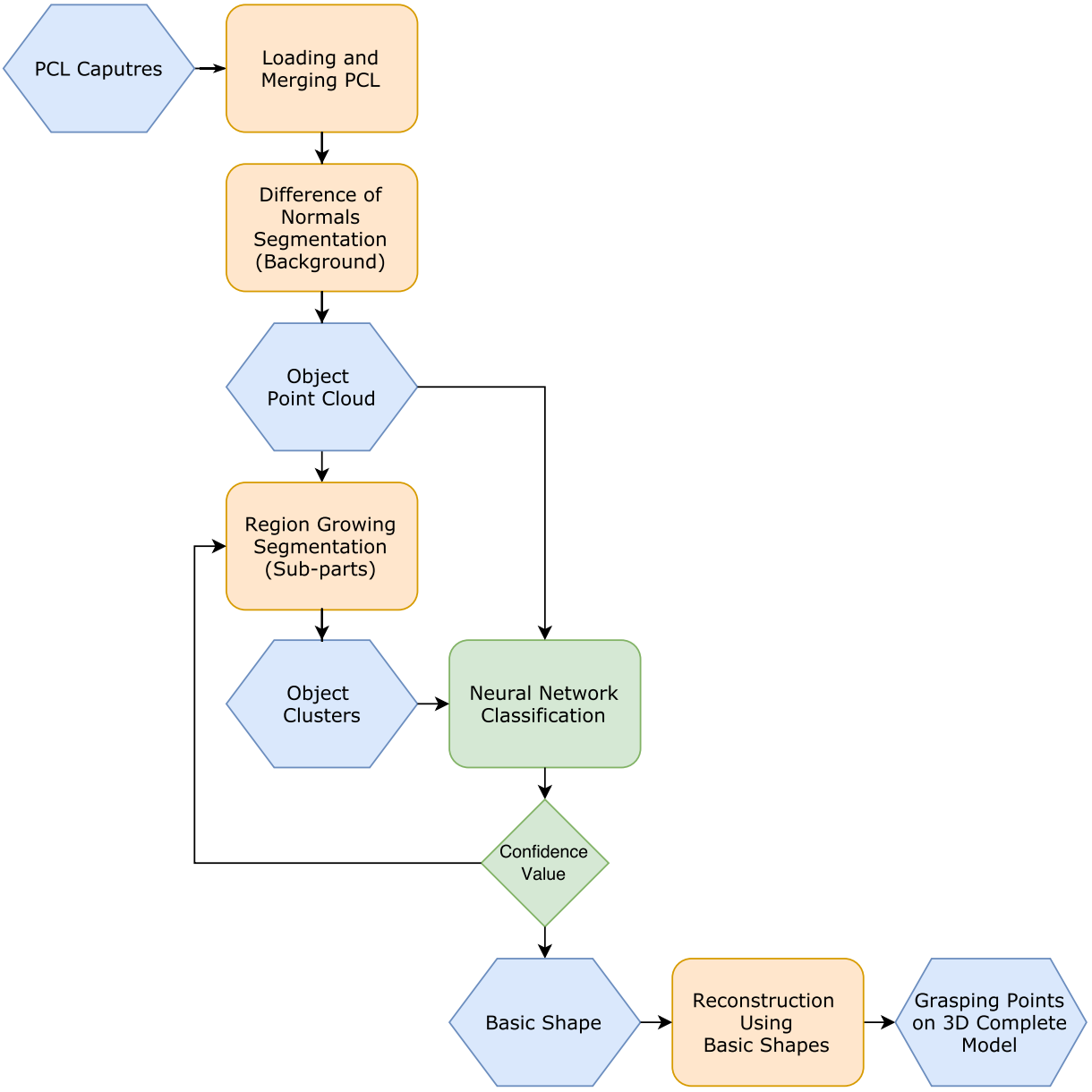

FIGURE 3.3: Routine flowchart for optimal grasping point detection.

3.2.1 Acquisition and Merging of Point Clouds

As previously stated in Section 3.1, the routine starts with the captures of point clouds that are basically gathered by the laser scanner mounted on the G500 arm shown in Figure 3.4 while grasping a box from the pool floor.

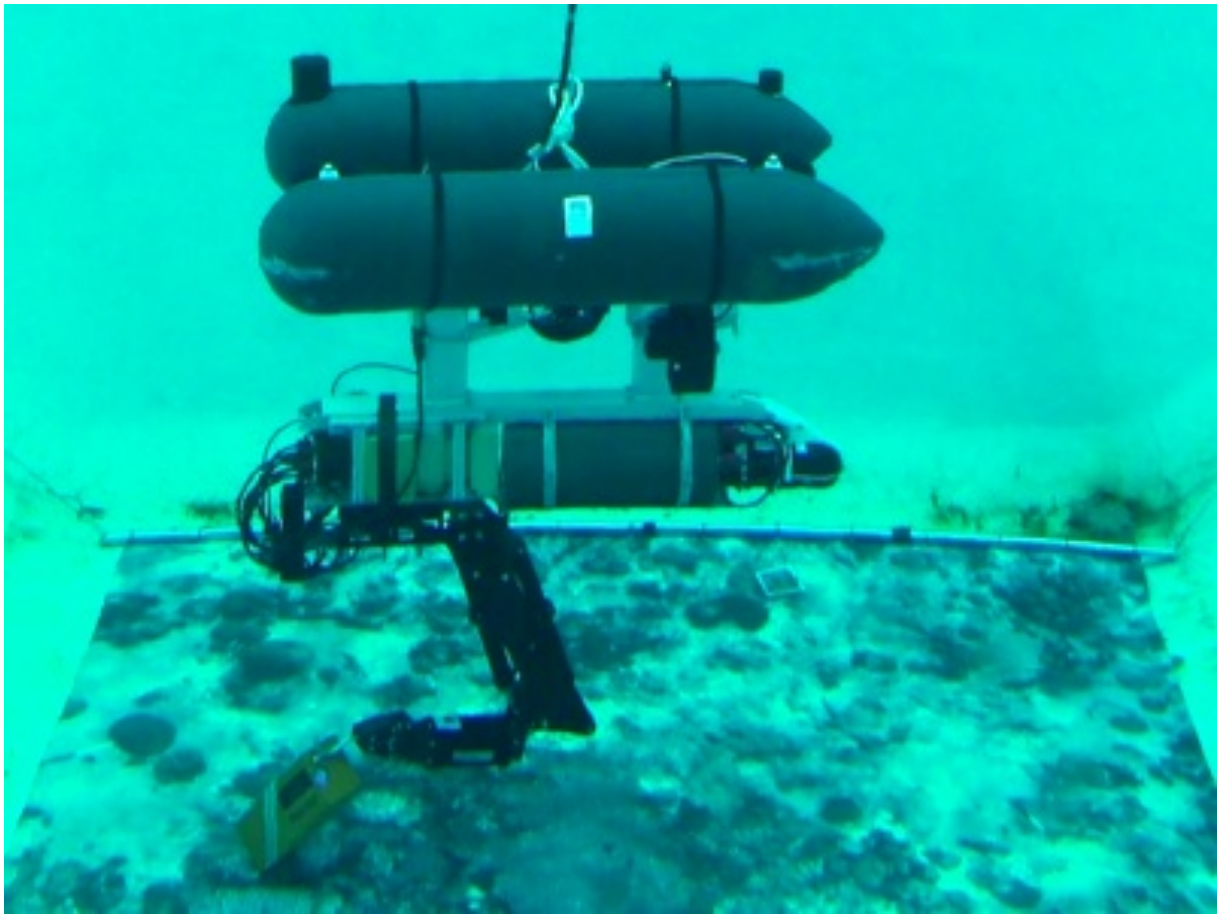

FIGURE 3.4: G500 with the UJI arm grasping a box from the pool floor.

All the used point clouds have been acquired in an environment simulating the sea floor: a pool where the floor is covered with sand, or with a picture of a generic sea floor, and where the robot is floating. Before, the point clouds were said to be similar to a gray scale image where the intensity of light is, in a way, related to the points depth in the scene.

In our specific case the situation is slightly different: the point clouds that will be analyzed are not retrieved by a single capture of a scene but, instead, they are created by overlapping more views (from 4 to 7) one over the other in order to have a more complete knowledge of the scene and in particular to have more information related to objects of interest. Generally, overlapping point clouds requires the above point clouds to have the same camera frame in order to have the same orientation for the points in the cloud. However, the G500 laser scanner is equipped with a routine able to set the origin frame at the base

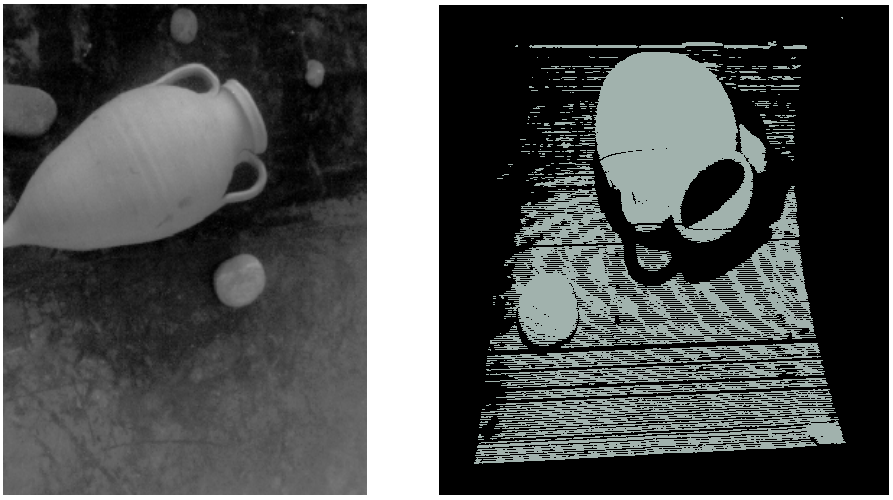

FIGURE 3.5: Real scene of an amphora and its point cloud.

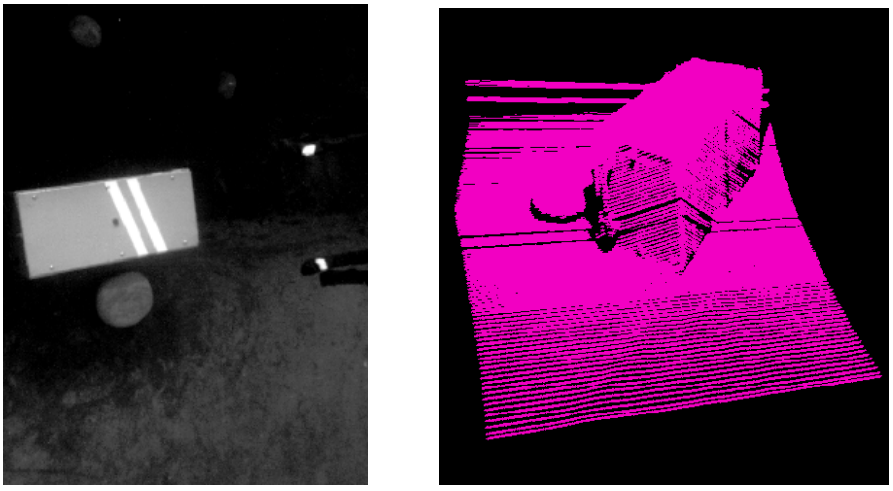

FIGURE 3.6: Real scene of a airplane black box and its point cloud.

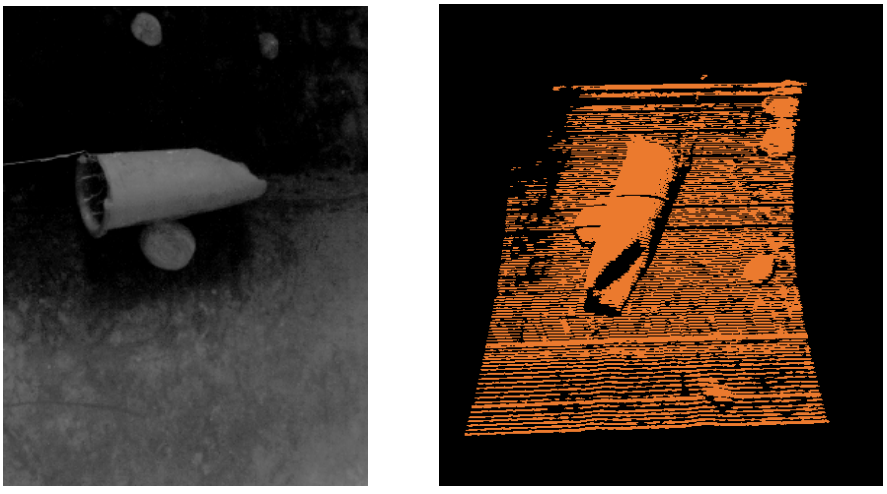

FIGURE 3.7: Real scene of a broken amphora and its point cloud.

of the arm, in this way the views acquired by the scanner have all the same origin frame so that overlapping them requires only to overlap all the points. In Figure 3.8 is shown a single capture of a scene representing a skull laying on the sea floor while, in Figure 3.9, is shown an example of reading and merging performed on the same scene.

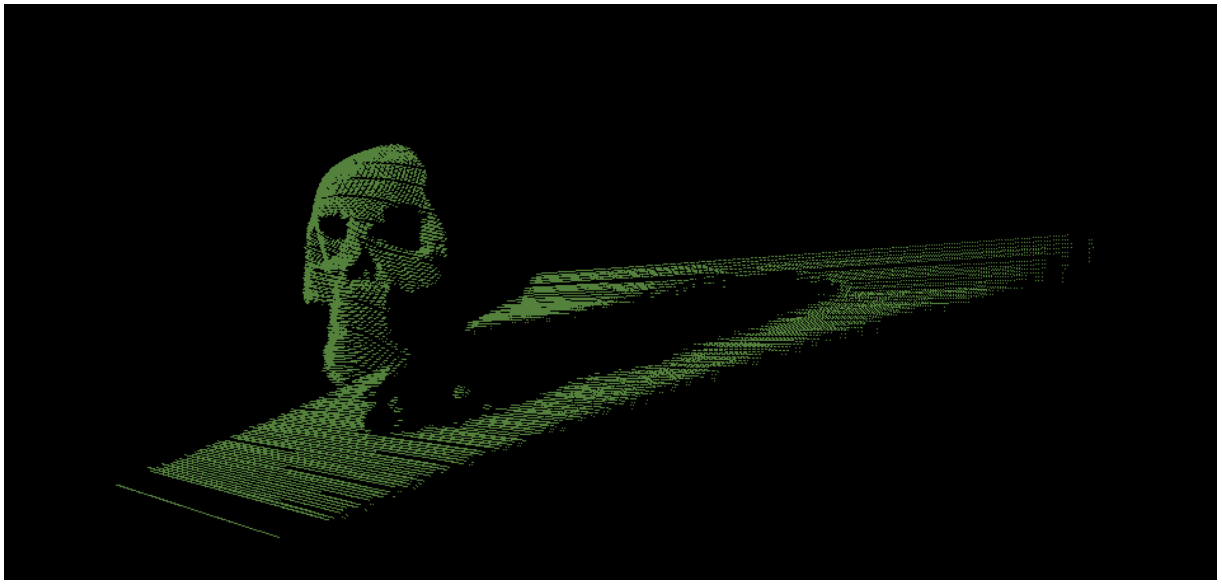

FIGURE 3.8: Single capture of a skull on the sea floor.

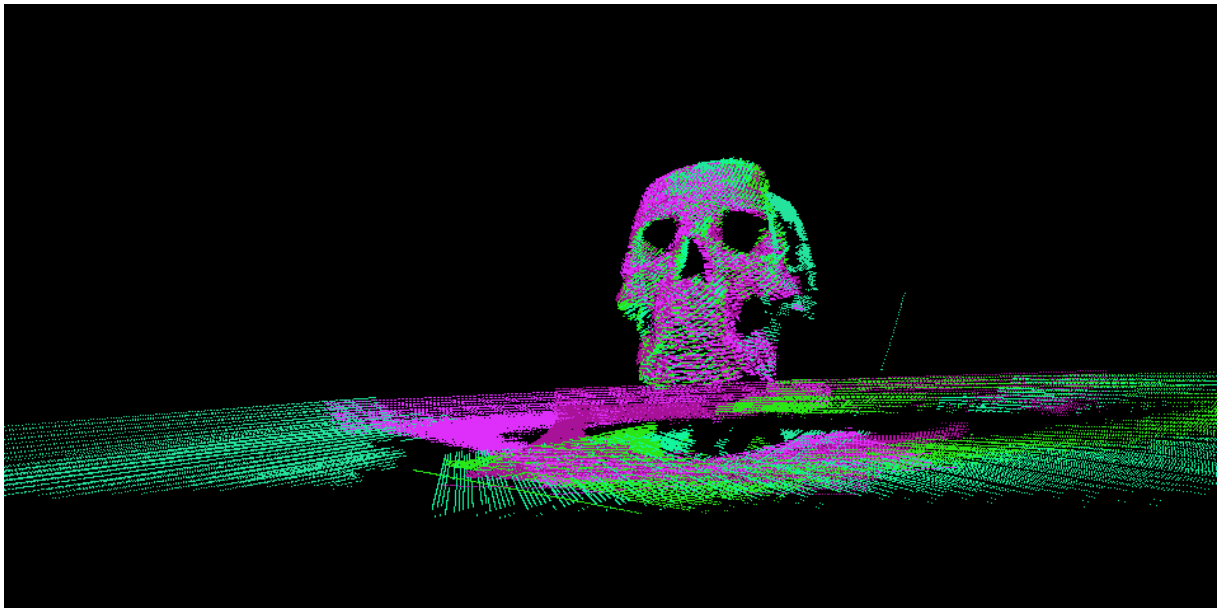

FIGURE 3.9: Merged capture of more point of views of the same scene.

Since the scene that we are using are simply overlapping, the resulting point cloud may present many points making very tedious the processing. For this reason a sub-sampling filter is provided in order to make more efficient the calculation in the next steps of the routine. In Figure 3.10 is represented the original merged scene of an amphora on the

bottom of the sea floor, while, in Figure 3.11 is shown the same scene passed through a sub-sampling filter.

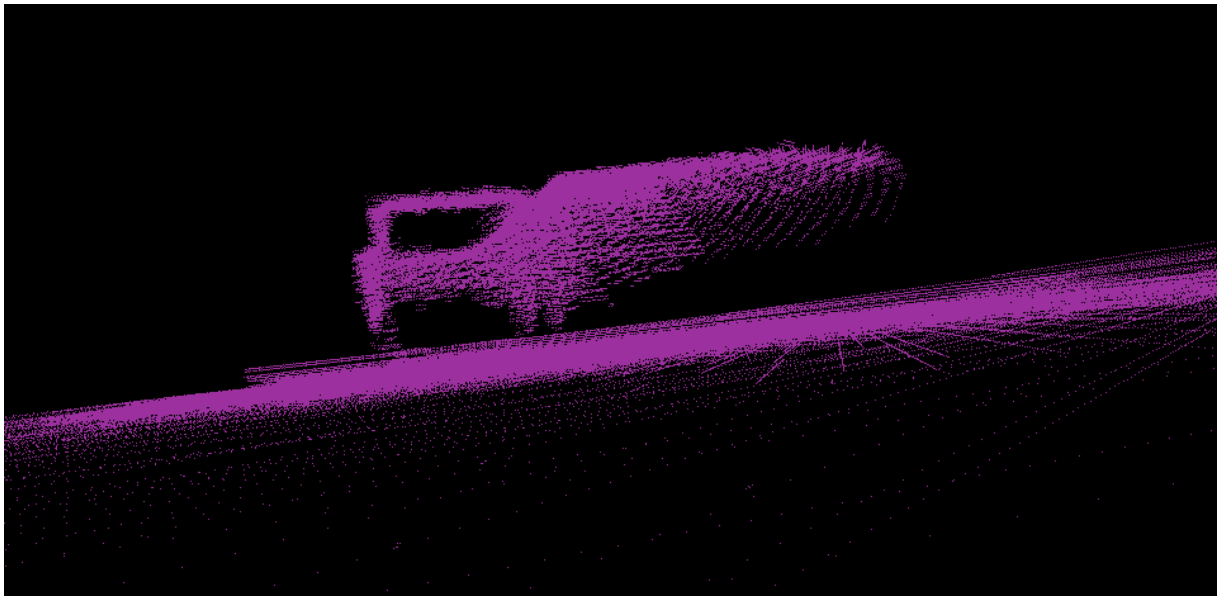

FIGURE 3.10: Merged scene of an amphora on the sea floor without subsampling.

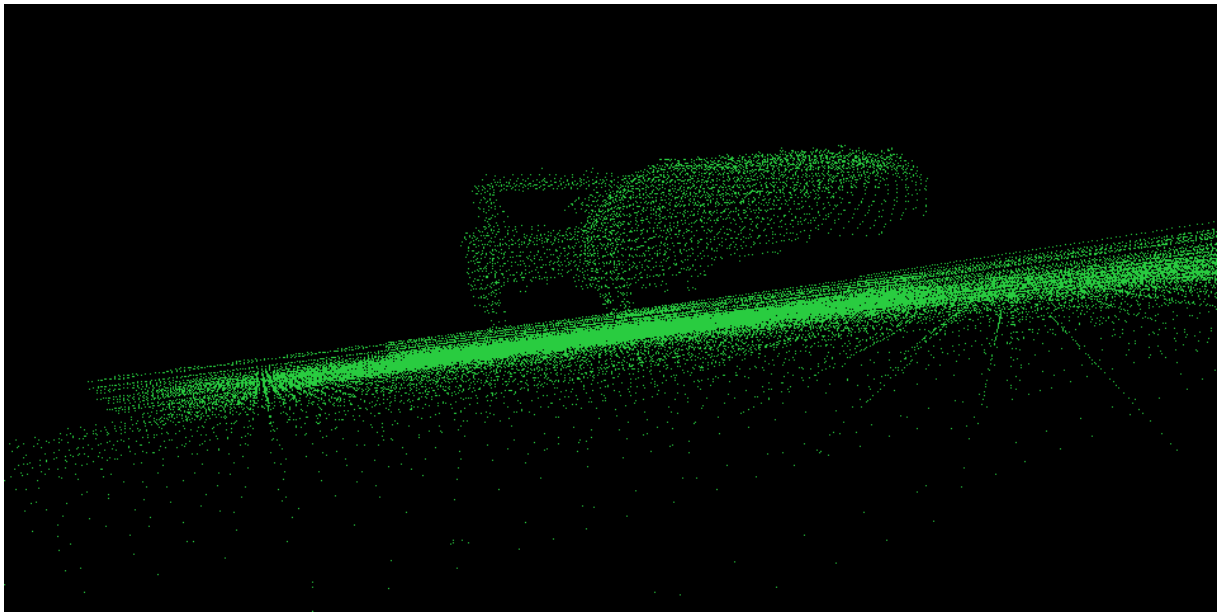

FIGURE 3.11: Subsampling of the previous scene.

It is possible to observe, in Figure 3.11 how the density of points lowers reducing the overall thickness of the cloud. Once obtained a complete and sub-sampled point cloud merging all the different views obtained via scanner, the next phase plans to separate the objects in the scene from the background.

The following section will explain which algorithm has been chosen to face this particular problem and the results obtained from a point cloud.

3.2.2 Difference of Normal Segmentation

In this section will be presented the algorithm used to separate the object from the background that in our case is the sea floor. In order to perform this task, the Difference of Normals segmentation has been chosen for his robustness, also in those point clouds where a little bit of noise is present on the surfaces. The idea behind the algorithm is simple yet very effective and can provide good results in a relatively small amount of time. The main concept, as the name suggests, find its basis in the usage of difference of normals of points in the cloud to segment it in a reliable way.

In Algorithm 1 is shown the basic functioning of the segmentation routine using Difference of Normals (DoN) [27].

Algorithm 1 Difference of Normals Segmentation

```
1: procedure EVALUATE DON FOR SEGMENTATION
2:     define r_l, r_s with r_l > r_s
3:     for each point in pointcloud do
4:         evaluate n̂(p, r_l) and n̂(p, r_s)
5:         evaluate Δn̂(p, r_l, r_s)
6:     for each Δn̂ in vector field do
7:         filter the field to obtain clusters
8:     use Euclidean Cluster Extraction to segment clouds
9: procedure SEPARATE OBJECTS FROM BACKGROUND
10:    compare all the normals of clusters
11:    if normals having the same orientation > 80% then
12:        label the cluster as background
```

The above steps have been implemented thanks to PCL (Point Cloud Libraries) and its functions that will be named, when used, along the explanation of the single steps. The first step is to define two radius values in order to evaluate the normal to a point taking advantage of those points contained in the defined range. The values of radius have to be such that one is major that the other, in our case we name r_l the bigger one and r_s the smaller one. Once defined the two values the next step is to consider every single point in the cloud and estimate the normal value using first r_l and then r_s. The normal estimation has been performed thanks to the class *NormalEstimationOMP* from PCL. This class has different methods that allows to set different parameters in order to compute the normal to a point having as input:

- the complete point cloud
- a radius size

- a search method
- a view point

The complete point cloud is needed to have all the local information regarding the single points and their relation with their neighbors within the radius set. The search method is need to define a way to look for points inside the point cloud in an ordered way. In our case the search method chosen is the *Kd-Tree* that consists basically in dividing the space into subspaces easier to access thanks to a binary tree. Finally, setting the view point is useful to evaluate the normals having all the same orientation that is, usually, external to the hollow point cloud. Once these parameter are set it is possible to evaluate the normals to the point cloud. In the following figures are shown the original point cloud with the normals estimated using different scenes, one with the amphora (Figure 3.12) and one with the skull (Figure 3.13).

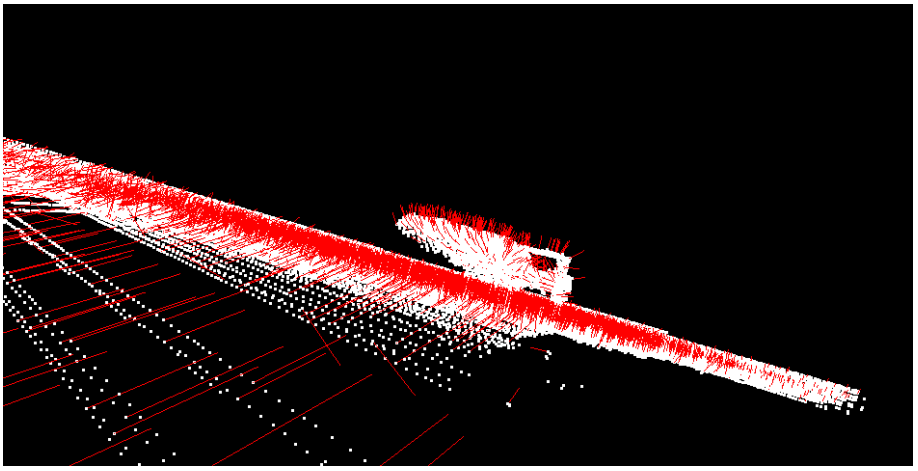

FIGURE 3.12: Normals evaluated on an amphora.

FIGURE 3.13: Normals evaluated on a skull.

Now that the normals to a point p are estimated considering two different radii r_l and r_s, is necessary to evaluate the Difference of Normals operator defined as:

$$\hat{n}(p, r_l, r_s) = \frac{\hat{n}(p, r_l) - \hat{n}(p, r_s)}{2} \tag{3.1}$$

The output value of this function is normalized vector having components in the range $(0, 1)$ and their norm will always be 1. In order to make clearer what is the purpose of this operator in Figure 3.14 is represented the DoN of a point obtained from the normal evaluation and difference of the normals estimated using r_l and r_s.

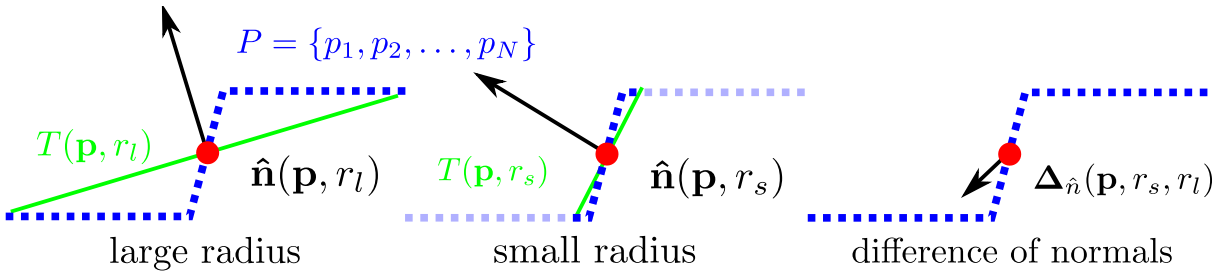

FIGURE 3.14: DoN of a point considering different radius values.

The next step is to filter all the differences of normals to separate them in clusters using a filter to discriminate the values of DoN along the whole vector field. In our case, a magnitude filter, implemented thanks to *ConditionalRemoval* and *ConditionOr* classes, has been applied to obtain a point cloud composed of those DoN vectors having high response with the previously given parameters. Once the point cloud is filtered, the final step is to extract the clusters out of the cloud with the *EuclideanClusterExtraction* class. As the name suggests this class exploits Euclidean Cluster Extraction algorithm [28]. The idea behind this algorithm is to analyze the point cloud as an organized set of point ordered thanks to the octree structure [29] that basically provides a search tree where every node has eight children. This particular structure make possible to easily search for object clusters, for example, on a plane surface like the one that is on the background of the point clouds and that is supposed to be the sea floor. After reordering the cloud with this criteria the algorithm takes place as shown in Algorithm 2.

The first step is to generate an organized Kd-Tree representation of the cloud $\mathbf{P}$ in order to make easier the search of clusters in it. Then two point sets are defined: $\mathbf{C}$, the set of clusters, and $\mathbf{Q}$, composed of all the points to check in the cloud. Subsequently all the points of the clouds are checked and added to the currently analyzed queue $\mathbf{Q}$ where all the points contained are checked to be within a sphere of radius r that must be less than a distance threshold d_{th}. Once found a point p_i^k that has not been already processed, it will be added to the point set $\mathbf{Q}$. Once checked all the points in $\mathbf{P}$ the

Algorithm 2 Euclidean Cluster Extraction

1: *create a Kd-Tree representation of point cloud* $\boldsymbol{P}$
2: $\boldsymbol{C} \leftarrow$ *empty list of clusters*
3: $\boldsymbol{Q} \leftarrow$ *set of point to check*
4: **for each** *point* p_i **in** P **do**
5: *add* p_i *to the current queue* **Q**
6: **for** *every* $p_i \in$ **Q do**
7: *search for the set* P_k^i *of points neighbors of* p_i *in a sphere with radius* $r < d_{th}$
8: **for** *every* $p_i^k \in P_i^k$ **do**
9: *check if the point has been already processed*
10: **if** p_i^k has not been processed **then**
11: *add* p_i^k *to* **Q**
12: *add* **Q** *to the cluster list* **C**

queue is added to the cluster list **C**. From the implementation point of view, again, as for *NormalEstimationOMP*, there are few parameters to be set also for the Euclidean Cluster Extraction that are:

- input DoN cloud
- search method
- minimum and maximum cluster size
- cluster tolerance radius

As before the input cloud is necessary to retrieve all the possible local information regarding points in the scene. However, differently from the previous case, the point cloud is representing the DoN vector field and not the simple scene. This means that we have to imagine the cloud no more as a scene composed of points where the depth is defined but, rather, as a cloud where every points has a value representing the DoN of that point.

The next parameter to be set is the search method and, again, the Kd-Tree was elected as the most fitting one. In addition to this is required to set a minimum and maximum size in order to discard those clusters that are not relevant for being too small or too big. Finally it is necessary to set a radius tolerance for the cluster to be segmented from the scene. Since objects may vary in shape and dimension the above radius is not optimized for every single item. However since we can rely on more than a view of a single scene, the information that may be lost during segmentation can be recovered by overlapping different views.

Once that the aforementioned parameters are set the segmentation can be performed. In the following figures are shown some of the results obtained using the segmentation routine just explained. In Figure 3.15 and 3.16 are shown two objects, an amphora and a skull, segmented and separated from the background. On the other hand, in Figure 3.17.

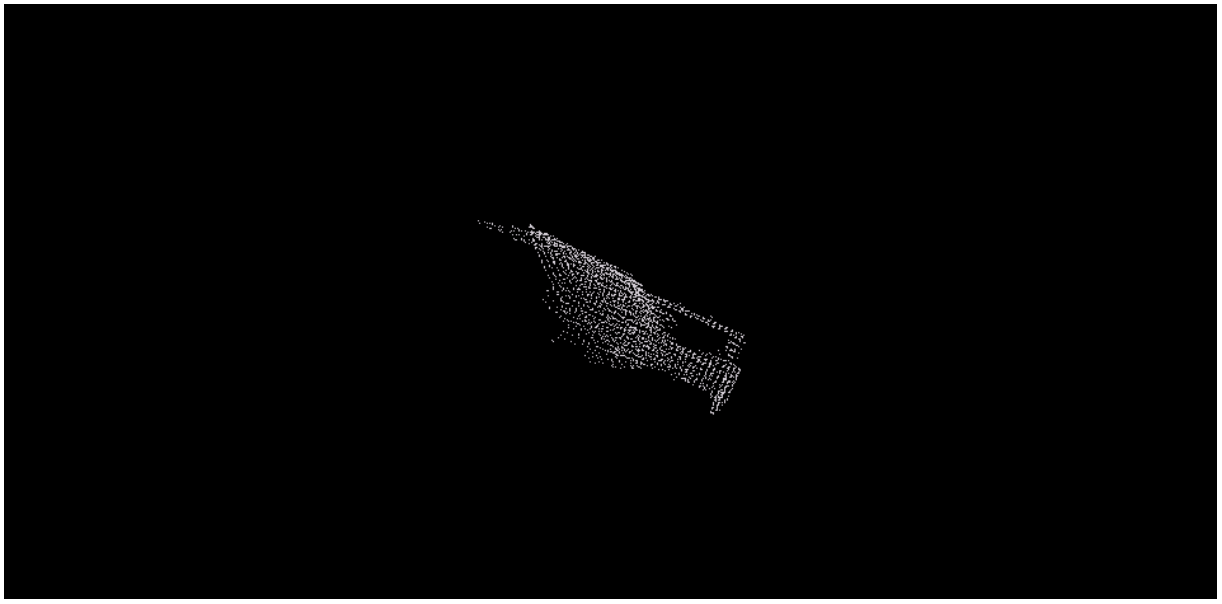

FIGURE 3.15: Amphora segmented from the background.

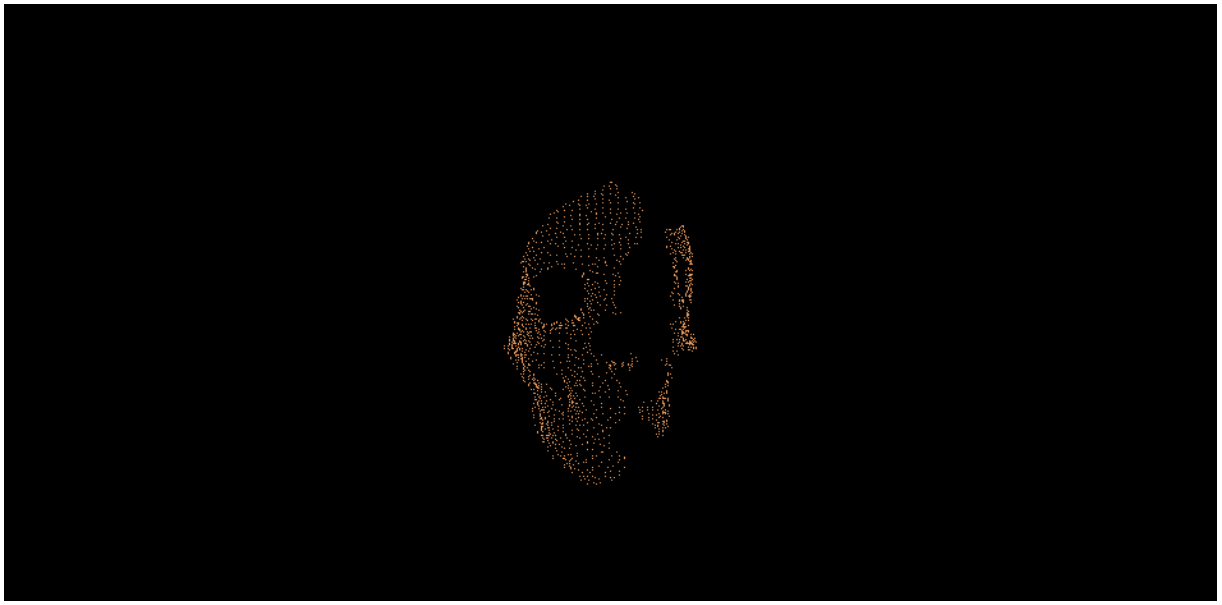

FIGURE 3.16: Skull segmented from the background.

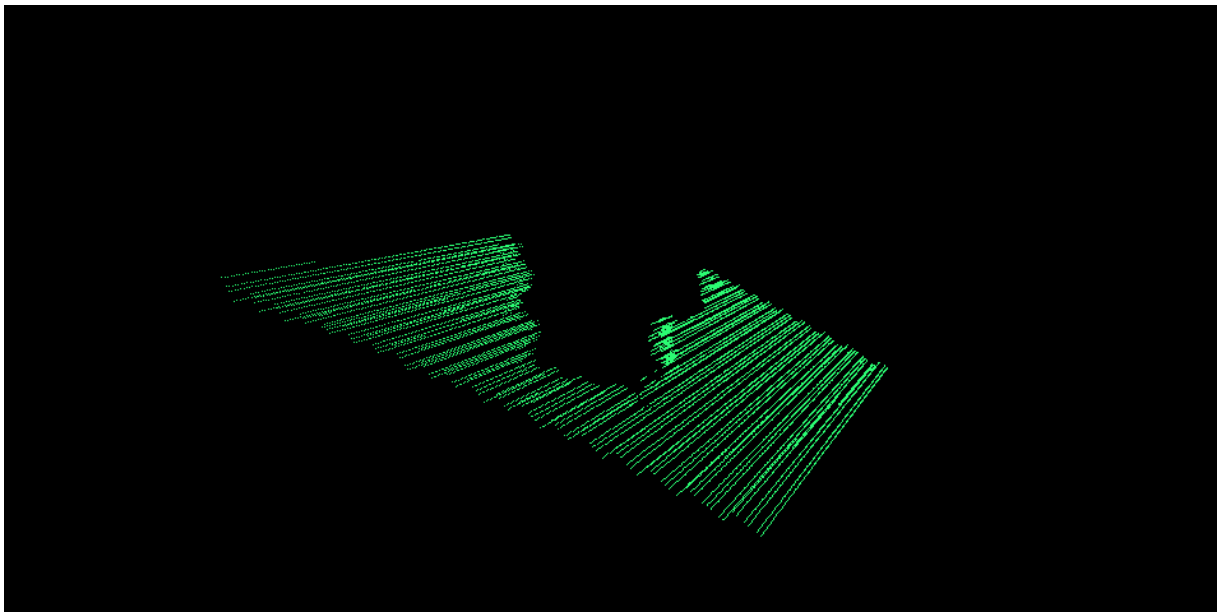

FIGURE 3.17: Background of the scene.

The last step to segment the scene consists in discriminating the background from the objects of interest in an autonomous way. In order to achieve this result the normals of every single cluster are compared within them. Since it is expected the background to be extracted from the scene as a cluster as a plane surface the normals of it should be all oriented in the same direction. For this reason a check on the normals to every cluster is performed and, if the number of normals having the same direction within a given tolerance range are more than a threshold, it will be classified as a the background, otherwise it will be labeled as object. In our specific case, the percentage of aligned normals for a cluster to be identified as a background has been set to 80% to avoid object with plane surfaces (such as boxes) to be classified as background. In addition, in case the sea floor would result to be not perfectly plane due to small rocks or sand dunes, this algorithm would ensure more robustness. In Figure 3.18 are shown the normals to the background plane: it is possible to notice how they are all aligned accordingly to what stated before.

Once obtained the objects discriminated from the background, since the the items of interest may be too difficult to be classified, a second segmentation, based on the curvature values, is performed along the cloud. In the following section will be explained how, starting from the point cloud of the object separated from the background, it will be segmented to classify its single parts to reconstruct the 3D model.

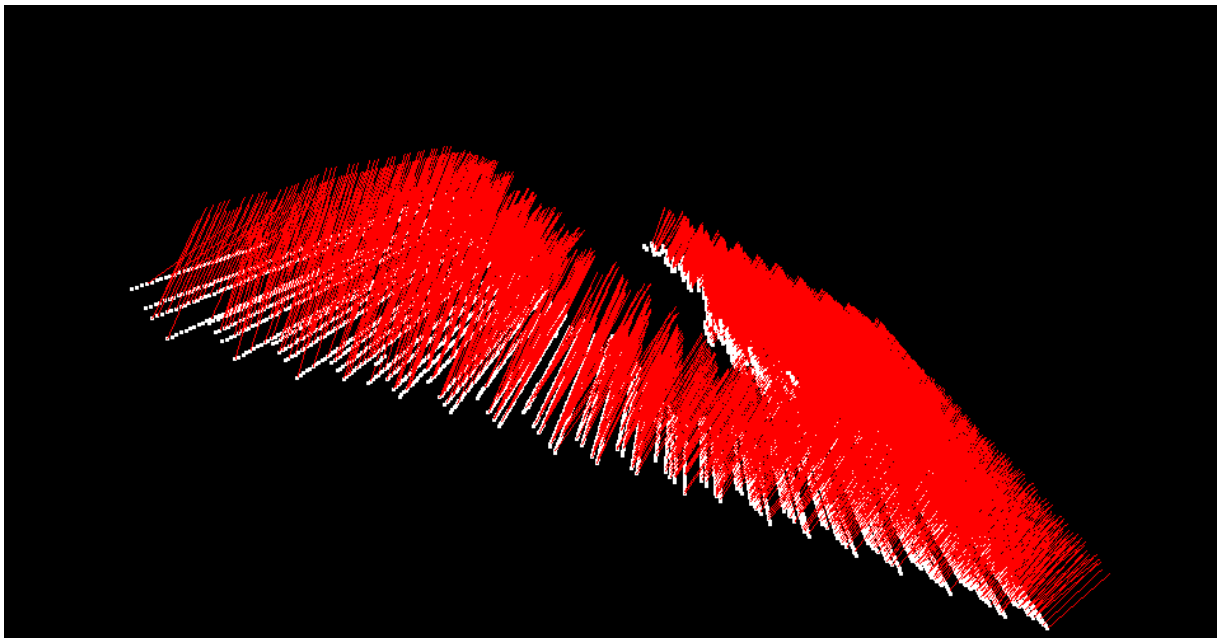

FIGURE 3.18: Normals relative to the background of the scene.

3.2.3 Region Growing Segmentation for Objects Subparts

In this section it will be presented the approach to segment the point clouds of objects into their cluster representing their subparts. The aim of this procedure is to divide the point clouds losing as less information as possible obtaining at the end every single part separated from the others. In Figure 3.19 is shown the point cloud of the previous

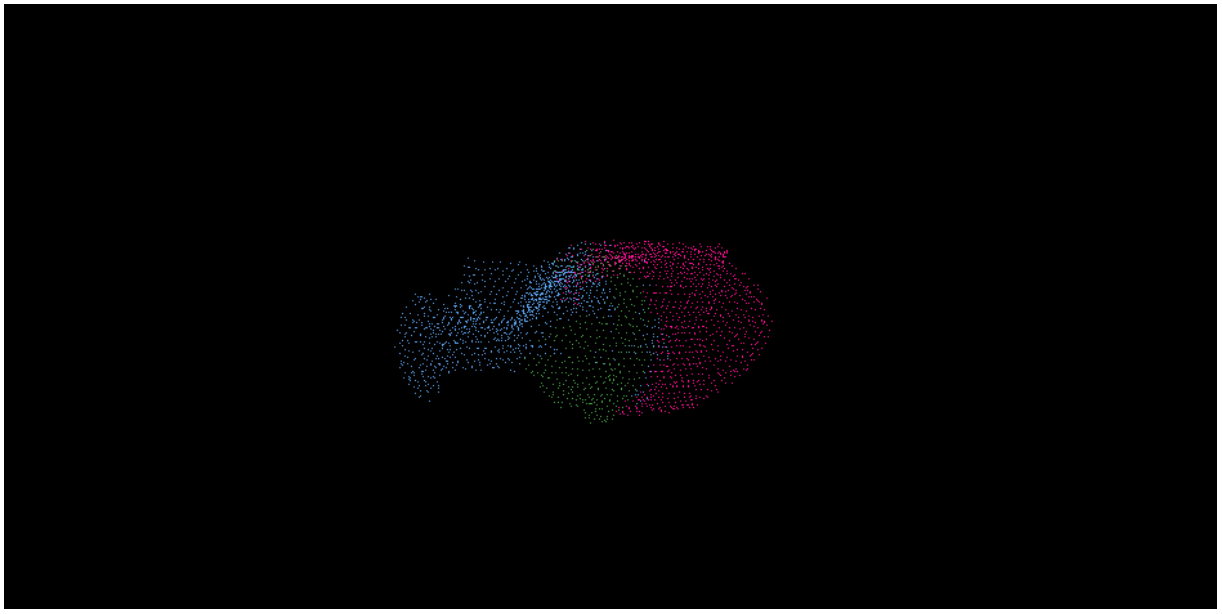

FIGURE 3.19: Normals relative to the background of the scene.

analyzed amphora and its segmentation in subparts with different colors. In this case

the parts composing the amphora are basically three: the body (pink), the neck and handle (light blue), and the neck base (green). To achieve this results a Region Growing Segmentation algorithm has been implemented. Its functioning is related to the values of curvature of the point clouds and for this reason it proves to be very effective for our purposes. Basing on this, since usually a variation of curvature in an object is related to a link of two or more different parts, it is possible to obtain the point cloud of the single parts of the entire object. Algorithm 3 starts evaluating every points curvature value and

Algorithm 3 Region Growing Segmentation

```
1: evaluate all points curvature value
2: point with minimum curvature is set as a seed
3: for every seed point p_i do
4:     find p_i neighbors
5:     if the neighbor has a curvature within a threshold then
6:         add it to the region
7:         if neighbor curvature is lower than a threshold then
8:             add them to the seeds
9:             remove seed used from the list
10:    if seeds list is empty then
11:        region has been completed
12:    restart from the beginning
```

sorting the cloud points basing on that. The next step consists in finding the lowest value of curvature and its related point that means also to find the point in the flattest region of the cloud. Starting from that point than the neighbors are analyzed and if the normal of the neighbor is within a threshold with the normal of the seed then that point will be added to the current region. Once that this step is performed the next one consists in checking again the value of the curvature and if it is below a curvature threshold those points will be add as seeds. Once that the neighbors are all analyzed the previous seed is deleted and the routine will start again with the update list of the seeds. The routine stops when the seeds list results to be empty.

Now that the object is segmented in its subparts the next procedure to apply to the clusters has to convert them into a format that the neural network presented in Chapter 2 is ready to interpret. The next section will discuss this particular part that is the conversion from point cloud to a voxel representation.

3.2.4 Voxel Representation

Obtained the clusters point clouds, the next step is to convert these clouds into voxel representations. Voxels are basically values representing volume in a three dimensional

space such as the pixels are values representing intensity of light in a two dimensional space like an image. Since point clouds usually have lots of points, it would be too much costly from the computational point of view to analyze every one of them. For this reason the best way to reduce its size is to represent the point cloud as a 3D matrix that will basically represent the presence, or absence, of points in a given region. In the following section it will be presented a routine able to convert a point cloud from the typical clouds format (*.pcd*) to an organized text file (*.txt*) containing the values of the voxelized cloud.

In our case to obtain a voxel representation of the point cloud are required few steps reported in Algorithm 4.

Algorithm 4 Point Cloud to Voxel Conversion

1: *import the cluster cloud* $\boldsymbol{P}$
2: *initialize a 3D matrix* $\boldsymbol{M}$ *having side dimension* $\boldsymbol{s}$
3: *magnify the cloud by a factor* $\boldsymbol{mag}$
4: *evaluate the center of mass of the cloud*
5: *move the cloud in the x, y, z positive space*
6: **for** every point in **P do**
7: *write a 1 in in the* $\boldsymbol{M}$*[P.x,P.y,P.z] element*
8: *print all the elements in order on a .txt file*

The first two steps basically prepare the environment for the conversion importing the cloud and initializing the 3D matrix that is going to be used to store the cloud values. The choice of dimension has no constraints, although it has to be big enough to contain the whole clouds. In our case the matrix has been chosen to be a square matrix with side $\boldsymbol{s} = 30$. In this way the clouds resulted to be defined enough to extract salient features from them without weighting too much on the efficiency of the evaluation for the network.

The next step magnifies the clouds in order to lose as less detail as possible during the voxelization process. This will affect also the dimension of the voxel yet without distorting the resulting representation. In this case the magnification factor may lead points to be out of boundaries with respect to the voxel matrix. In our routine the magnification factor has been set in order to magnify eight times the dimension of the clouds since clouds and clusters appeared very small in space.

Once obtained a big enough cloud, it is required the clouds to be in the positive (x, y, z) space in order to simplify the indexing of the cloud within the 3D matrix. For this reason the center of mass is evaluated and, estimating the distance between it and the origin of the axis it is possible to move it in the interested region of space. In Figure 3.20 is shown the original point cloud (black) and the moved one (red). In this case the cluster considered is the body of the amphora previously seen segmented in Figure 3.19. It is

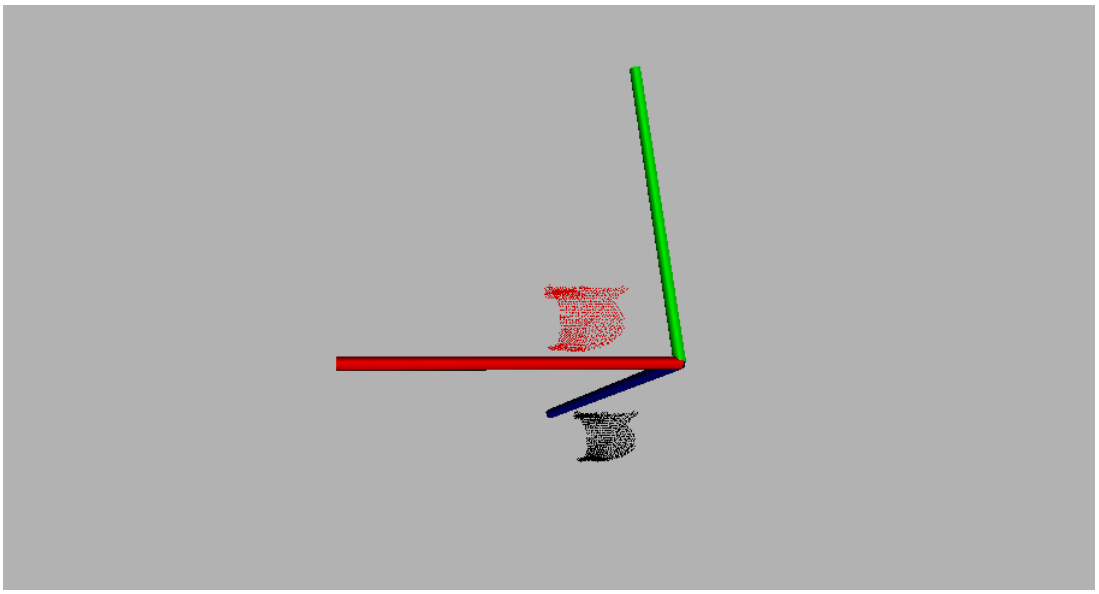

FIGURE 3.20: Original cluster (black) and moved cluster (red).

possible to observe in this case the position of the red cluster with respect to the origin frame. Thanks to this translation it is possible now to convert the point cloud into its voxel simply indexing the points of the cloud into the 3D matrix exploiting the coordinates value of them. In this way the elements relatives to subspaces in which are contained points will be filled with ones, otherwise, if no points are found, the corresponding element of the matrix will contain zero values. In Figure 2.4 is shown the final result of the voxel conversion taking as input the same cluster seen in Figure 3.20.

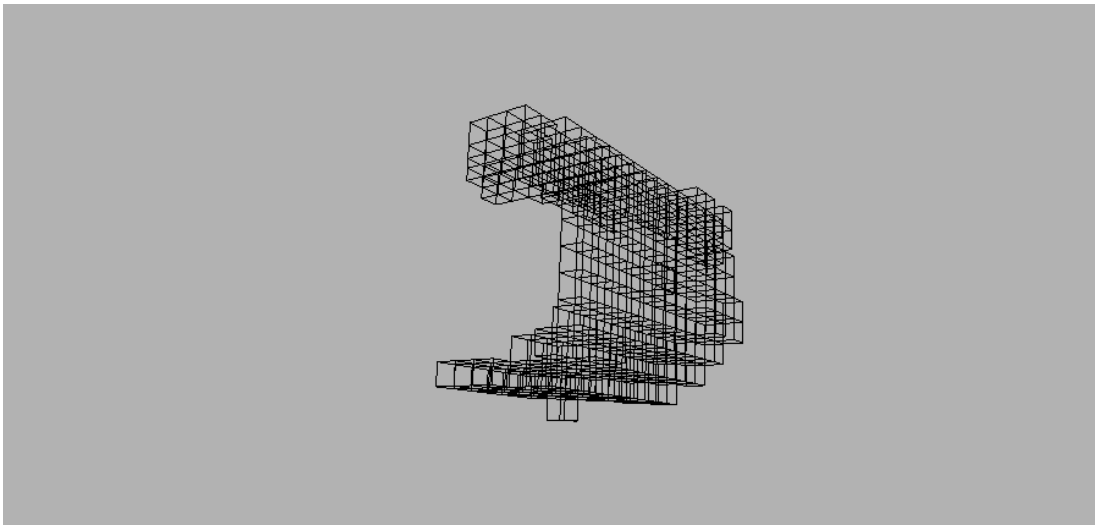

FIGURE 3.21: Voxel representing the body of an amphora.

Once obtained the voxel of the cloud it will be saved in a text file in order to be read, in a second time, by the input layer of the network to classify the shape. Classification will provide a confidence value that will be checked and, whether the value will result less

than a threshold that has been empirically set to 0.7, the segmentation will be performed again but with different parameters that will allow segmentation to be more sensitive to curvature variation. In this way it is possible to create smaller cluster easier to classify. In order to do this parameters relative to 3, such as curvature threshold and number of neighbors to check, are reduced at every iteration by 20% in order to retrieve information regarding smaller features that will correspond to smaller parts of the object. In this way it's possible to classify more precisely the object in all its parts. However, it may occur the case in which even segmenting in smaller parts the object it proves to be impossible to properly classify a shape. In Figure 3.22 is represented a case of these where, looking for smaller parts of the skull didn't help the classification process.

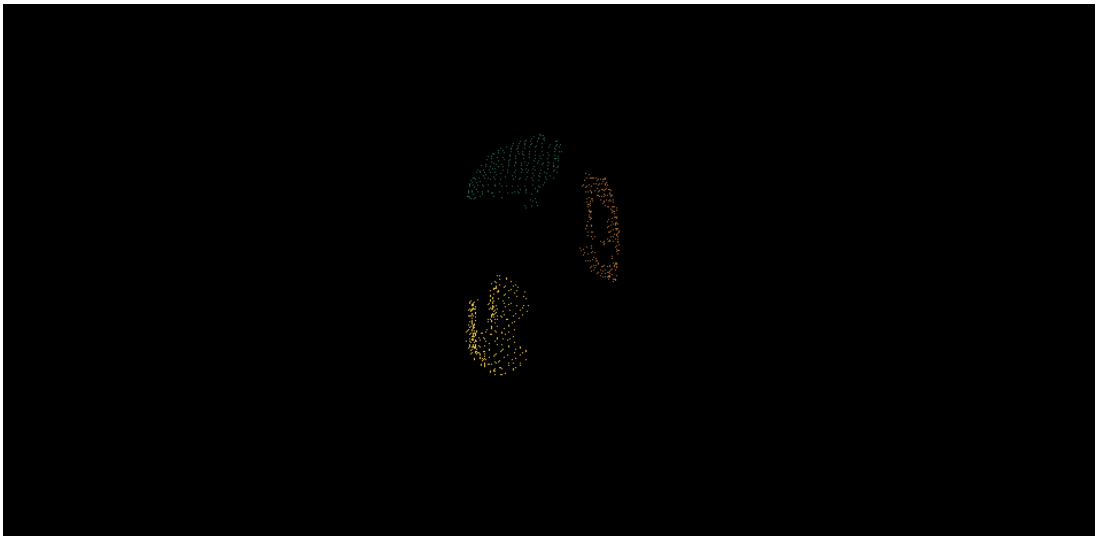

FIGURE 3.22: Region growing segmentation on skull point cloud.

As it is possible to see, it results to be impossible to the network to classify these clusters that are hard to define even to a human eye. In cases like these a failure recovery occurs that will be explained in detail in Section 3.2.5.5.

The next and final step performed by our point cloud processing is the 3D model reconstruction, once that the network . In the following section will be explained in detail how reconstruction works and how it will provide results in order to detect a pair of grasping point.

3.2.5 Reconstruction

In the previous chapter has been faced the problem regarding building an effective neural network to classify shapes retrieved as output out of a point cloud processing routine. The last step is to provide a process able to reconstruct the shapes that have been detected

by the network and overlap them to the point cloud. In this section will be reported in detail how this routine will work and how its output will result useful to our purposes of detecting a couple of grasping point able to ensure a robust enough grasp. Basically here will be presented how the routine is able to reconstruct the full shape out of the information retrieved by the partial point cloud. For every class of shape will be presented a different approach to the reconstruction. Although every single shape has its proper way to be reconstructed, all the shapes are retrieving information about dimension and orientation thanks to bounding boxes created in order to be the smallest boxes containing the clusters according with the direction of the moment of inertia estimated through the *MomentOfInertiaEstimation* class. This class offers different functions able to evaluate the topology of the cluster and, after finding the center of mass, compute the directions of the moment of inertia that will be used later to define the grasping point over the 3D model. In the following sections, routines used to reconstruct shapes out of their classification are going to be explained in detail.

3.2.5.1 Cube Reconstruction

The cube reconstruction proves to be the easiest to perform since it is possible to simply address the problem with the search of the smallest bounding box containing the point cloud. Thanks to the aforementioned *MomentOfInertiaEstimation* class and its functions it is possible to extract descriptors to determine which are the points that define the corners of the bounding box. To determine these points the covariance matrix of the point cloud is calculated and its eigen values and vectors are evaluated. It is possible to consider the resultant eigen vectors as normalized and always form the right-handed coordinate system (major eigen vector represents x axis and the minor vector represents z axis). The iteration process takes place on the next steps where every time the major eigen vector is rotated always with the same order and around the other eigen vectors. Thanks to this the invariance to rotation of the point cloud is provided. In this way it is possible to find out the corners of the bounding box and the corresponding aligned frame referred to the center of mass. In Figure 3.23 is it possible to see the result of this routine applied to an amphora. In this case the amphora has not been classified as a cube but it has been chosen to simply show the results of a possible reconstruction. It is possible to see how the bounding box surrounds effectively the point cloud of the object.

In red and green it is possible to see the aforementioned aligned frame. To determine grasping points in the case of a cube or a box it is needed to find the intersection between the axis of the frame and the plane of the box. Since the z axis would result perpendicular to the floor, its intersection points with the bounding box have been discarded. In fact,

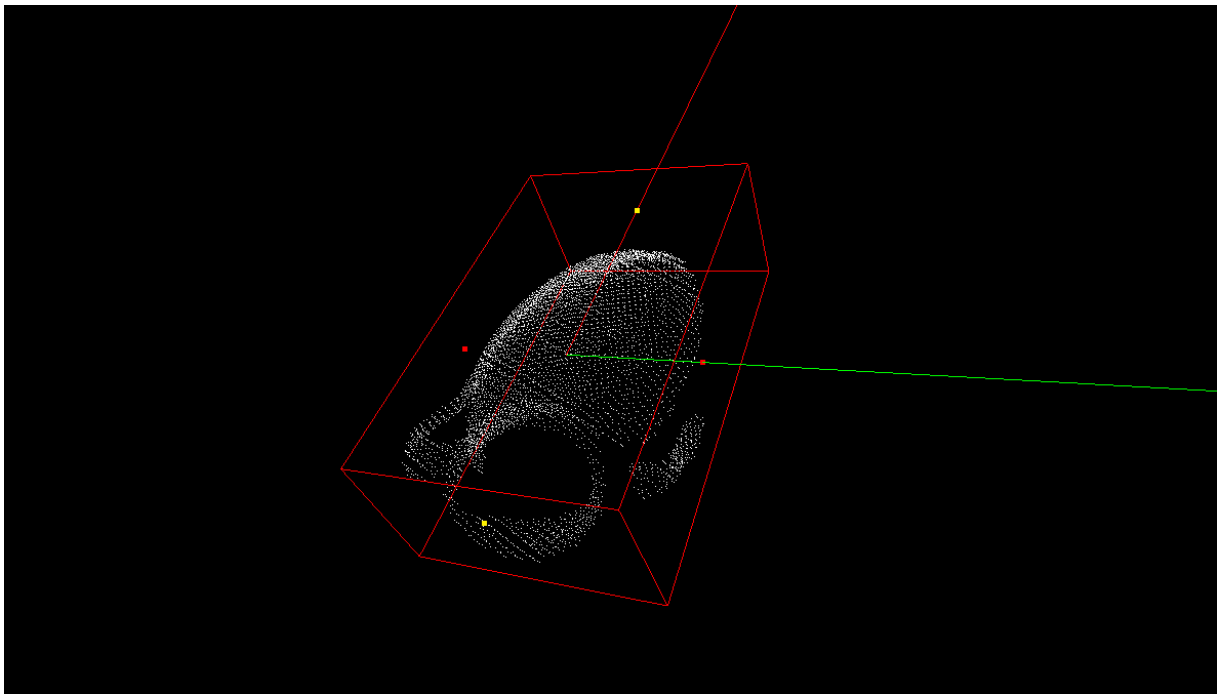

FIGURE 3.23: Bounding box surrounding an amphora.

since the object will lay on the floor, one of the gasping points on the z axis will be on the bottom of the object, one the contact point between it and the sea floor, where it is impossible to guarantee a strong grasp without scratching the floor with the risk of damaging the gripper. In this case the grasping points are found and shown in the figure in pairs: one in yellow and one in red. In addition the yellow points doesn't fit the requirements of a robust grasp because the point cloud analyzed is not representing neither a cube nor something similar to a box. In the case of a box correctly classified, thanks to the symmetry of the shape both the pairs would guarantee a tough grasp for the object.

3.2.5.2 Cylinder Reconstruction

In the previous section has been present the routine able to bound with a box a point cloud and basically to reconstruct the shape of a box or of a cube. In this section will be shown the basic geometrical concepts behind the the reconstruction of a cylinder.

In this case the reconstruction needs basically two values:

- radius of the cylinder base
- length of the cylinder

that are the inputs for the *ModelCoefficients*, necessary to build up the shape. It is possible to obtain these parameters thanks to the same bounding box shown in the previous routine and the coordinates center of mass. Using the length of the bounding box it is possible, in fact, to define the length of the cylinder and, evaluating the distance of teh center of mass from the line that defines the length it is possible to determine the radius of the base. In Figure 3.24 is shown in an easier way which are the dimensions to consider to reconstruct the cylinder. Finally the pair of grasping points are found in a similar way to

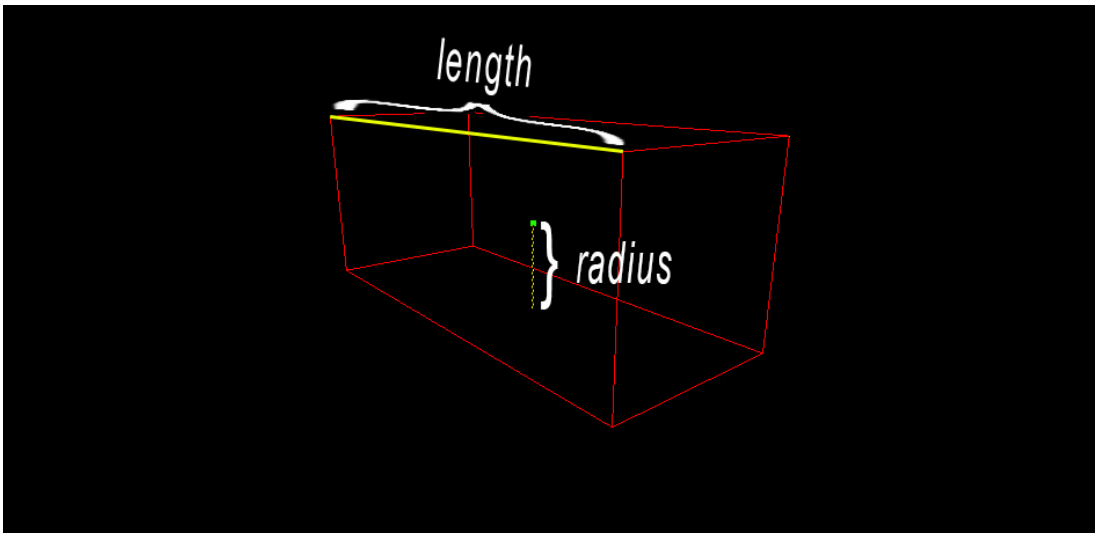

FIGURE 3.24: Measures relative to cylinder for reconstruction

the box previously seen. In this case the graping points are evaluated as the intersection between the oriented frame in the center of mass and the cylinder.

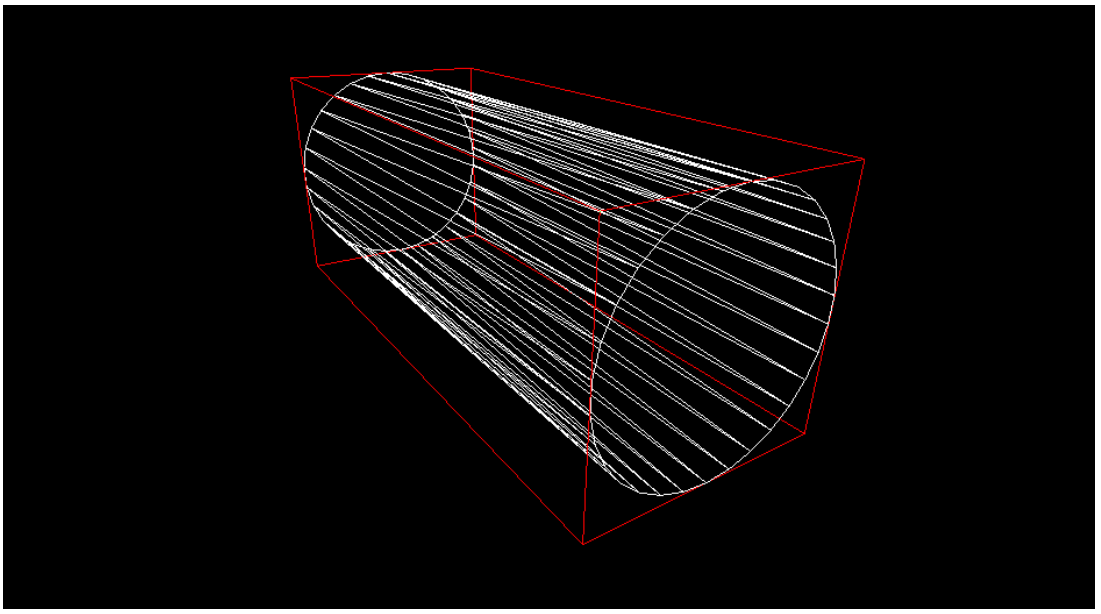

FIGURE 3.25: Cylinder inside the bounding box.

Cylinders are a particular case, in fact, it is possible to assume the grasping of a cylinder as the grasping of a box. In fact, the pair of points needed to perform the grip could be found as the intersection of the bounding box around the cylinder since the couple of grasping points will surely lay on the lines generated by the intersection between the box and the cylinder (Figure 3.26).

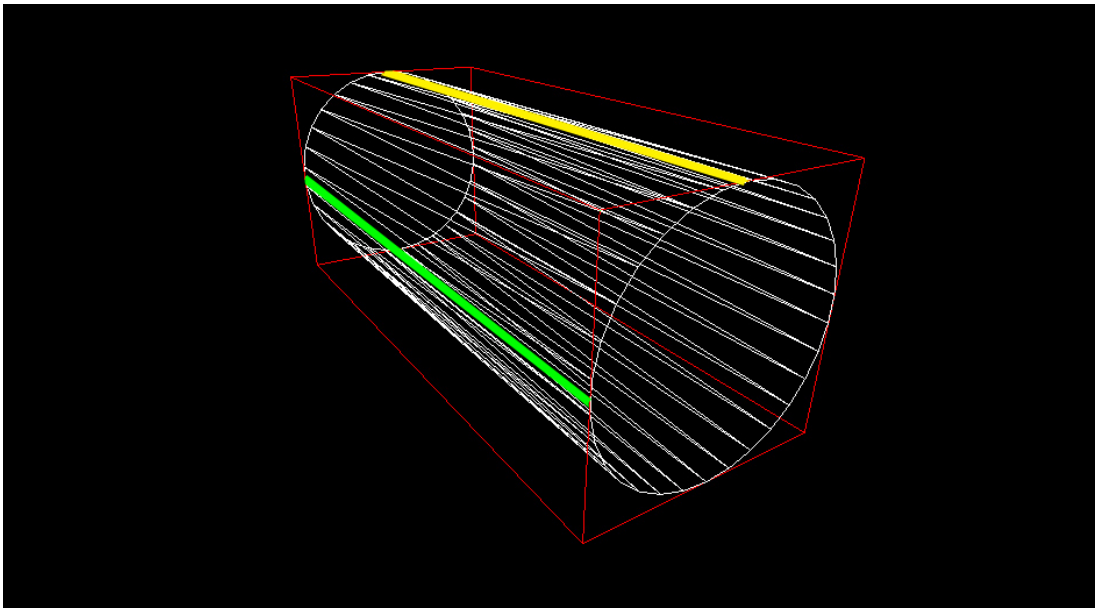

FIGURE 3.26: Intersection lines between a cylinder and a bounding box.

Depending on the center of mass of the complete object, the grasping points must be chosen as the pair of points laying on the intersection lines shown in Figure 3.26 that results to be collinear with the center of mass.

3.2.5.3 Cone Reconstruction

What has been shown in the previous sections are the processes needed to reconstruct both a cylinder and a box out of the descriptors given by the point cloud. In this section will be presented a more tricky reconstruction routine: the process to reconstruct a cone.

The cone reconstruction proves to be one of the trickiest one along the four shapes. In fact due to its symmetry properties the cone is the most difficult to orient with respect to the original point cloud. As in the previous case few measures are required to evaluate the size of the cone and to define the *ModelCoefficients*. These parameters are basically:

- angle width
- origin point

The cone is generated starting from the tip and, given the width of the angle, the rest of the shape follows. Regarding the origin point, again, it is evaluated, as in the cylinder, as the central point of the face of the bounding box. The width angle is evaluated thanks to few simple geometry calculations. In Figure 3.27 is shown the cone with it's parameters.

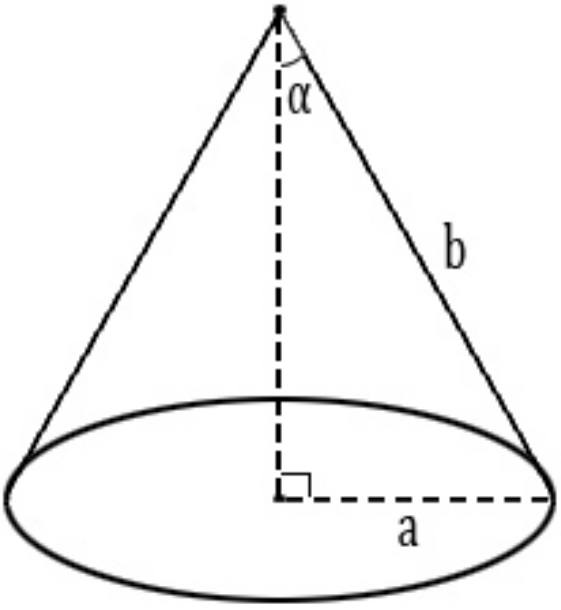

FIGURE 3.27: Parameters of the cone.

To find the value of the angle width the calculation is simple:

$$\alpha = arcsin\left(\frac{a}{b}\right) \tag{3.2}$$

where a is the simple distance between the center of mass and the bounding box and b is the distance between the origin and a point on the other side of the box. Having these two values it is possible to evaluate α and draw the cone respecting the size of the point cloud. A point that could lead to some ambiguities is how to orient the cone. In this case a check is performed and, if the center of mass is closer to the origin of the cone than to its base, the origin point is switched and so the orientation. In fact, the center of mass of a cone is supposed to be closed to the base than to the tip of the cone. Therefore, thanks to this property of the cone is it possible t determine the global orientation of the shape. In Figure 3.28 is shown the final results of the cone reconstruction with the cone bounded by the original box. In the cone case is not trivial to detect a set of points for a robust grasping without taking into account the friction of the gripper over the cone surface. In addition to this, considering that we are working in an underwater environment, object may result slippery making more unstable thus complicated the grasping of a cone-shaped

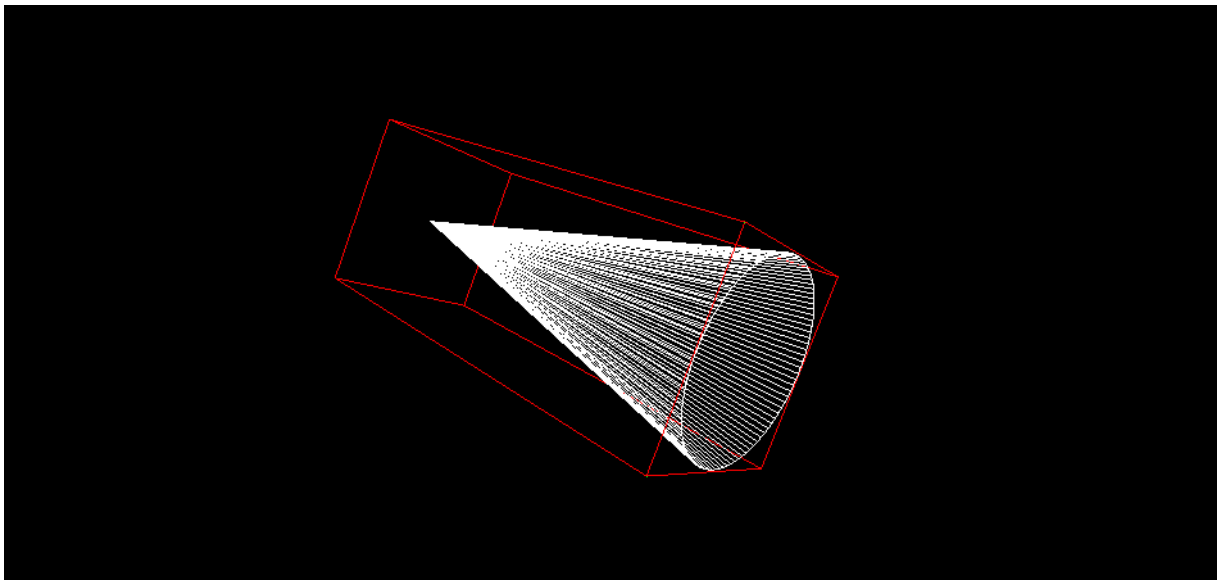

FIGURE 3.28: Reconstructed cone within the bounding box.

object. In this case the best solution would be to set the gasping point on the intersection points between the cone base and the bounding box. To make it clearer in Figure are highlighted these points. In green is highlighted a pair of grasping points and in yellow

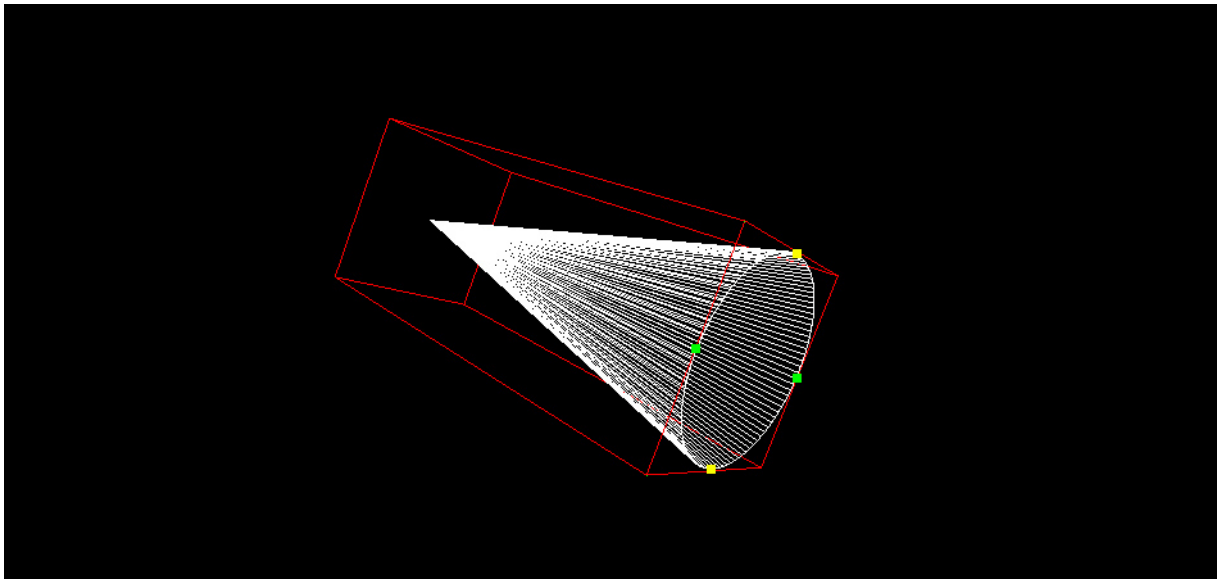

FIGURE 3.29: Grasping points detected on cone base.

another possible one.

3.2.5.4 Sphere Reconstruction

Previously three out of the four possible shapes have been shown and the routine to reconstruct them has been explained in detail. The reconstruction process to show is the one regarding the sphere.

The reconstruction of a sphere is simple since all the necessary for its parametrization is already given from the beginning. Actually what is needed to define a sphere is basically:

- sphere center
- sphere radius

In this case the center of mass of the partial cloud could be used as center of the sphere and, on the other hand, the distance between the center of mass and a face of the bounding box could be assumed to be the radius. Having these parameters it is possible to define an instance of *ModelCoefficients* to reconstruct the sphere. What we will obtain thanks to this routine is shown in Figure 3.30. In this case the pair of grasping points is easy to

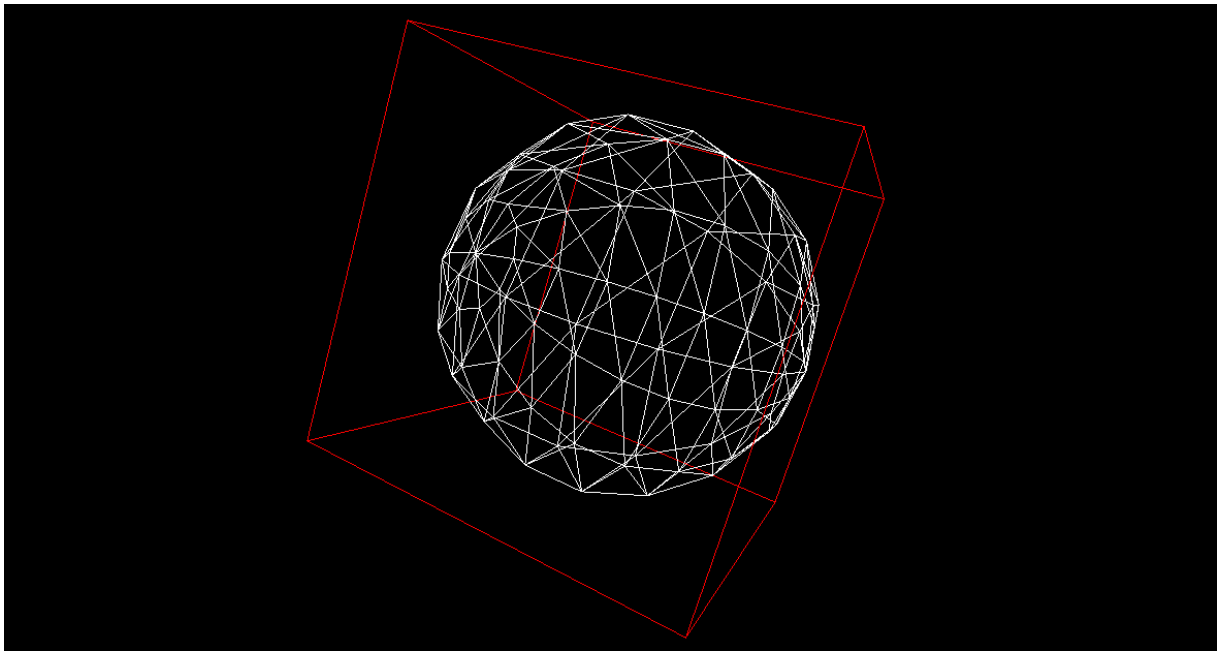

FIGURE 3.30: Sphere contained into the bounding box.

detect since every pair of points coming from the intersection between a diameter and the sphere is a valid one.

3.2.5.5 Recovery from Failures

Previously, in Section 3.2.4, it has been stated that, if the network is not able to classify in reliable way the shape taken as input, the segmentation must be performed with reduced values in order to extract smaller cluster to classify. This operation could lead to an infinite loop where the segmentation keeps on extracting smaller clusters out of the cloud until segmentation results contain so few points that it is impossible to classify them. For this reason this iteration is performed no more than three times. If, after three iteration, it results to be still impossible to classify with a confidence value high enough a recovery mode occurs.

Basically this mode consists in defining the easiest grasping points to detect along the point cloud. In other words, ignoring all the classification, the cloud is surrounded by a bounding box like the one shown in Section 3.2.5.1 and its grasping points are evaluated as the intersection between the vectors of the moment of inertia and the latter box. In this case the results obtained are similar to the one represented in Figure 3.23 where the two pairs of grasping points are shown. This recovery not always proves to be effective or safe not depending on the actual shape of the object but only on its dimension. However there are many case that will be presented later in Chapter 4 where this solution is not leading to a far result from the desired one.

Now that the algorithms behind point cloud processing have been described and their details have been explained the last step is to test them on real data coming from the laser scanner of the G500. In the next section the results of the neural networks and the ones coming from the point cloud processing are shown and discussed.

Chapter 4

Results

In previous section have been presented the fundamental instruments for this project to work: in Chapter 2 a neural network used to classify shapes out of a point cloud scene has been presented while, in Chapter 3 a point cloud processing routine have explained in detail. In this Chapter will be presented the results obtained using as input real scene gathered via laser scanner mounted on the G500 arm for grasping. Few of these scene have been partially seen in the previous sections as examples.

In order to increase performance the neural network have been implemented in Windows to exploit the power of the GPU at its maximum while the point cloud processing have been performed on Ubuntu due to it's ease in dealing with external libraries such as PCL. For this reason the automatic process have been simulated simply using the output of the network as input of the reconstruction routine manually. Making automatic the whole process requires the whole routines aforementioned to be all on the same operative systems like Ubuntu that supports better both the libraries from TensorFlow and PCL. On the other hand, Windows proves to be trickier in managing PCL libraries for point cloud processing. This makes Ubuntu the platform that better fits the needs of this project.

4.1 Scenes and Objects

In this section will be shown the point clouds with which we are going to work to test the neural network and the point cloud processing for both classification and reconstruction. Some of the objects have been already shown in previous sections as examples to explain point cloud processing. All the scene, as aforementioned are taken through laser scanning a real scene as shown in Section 3.2.1. The following images show the point cloud of the

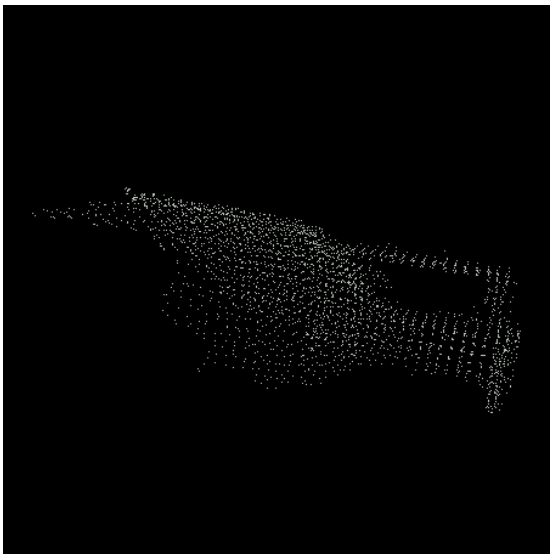

FIGURE 4.1: *amphora_0.*

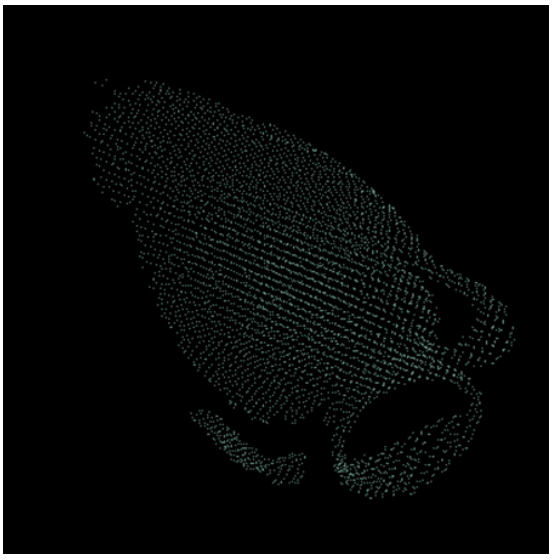

FIGURE 4.2: *amphora_1*

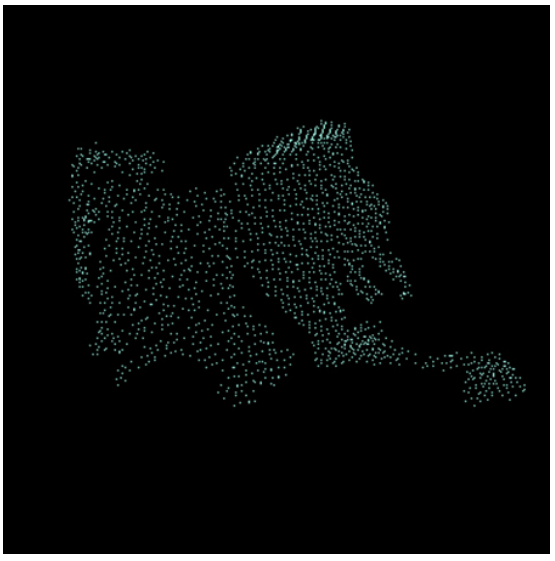

FIGURE 4.3: *black_box.*

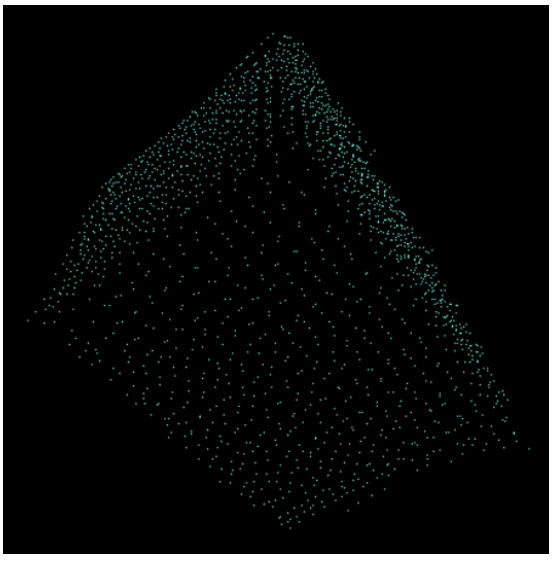

FIGURE 4.4: *box_0*

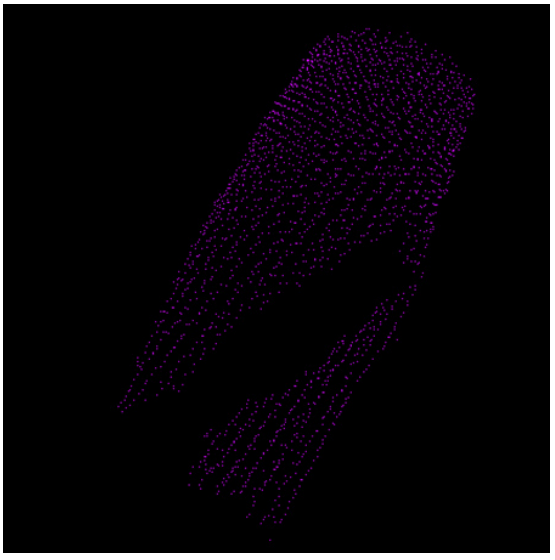

FIGURE 4.5: *broken_amphora.*

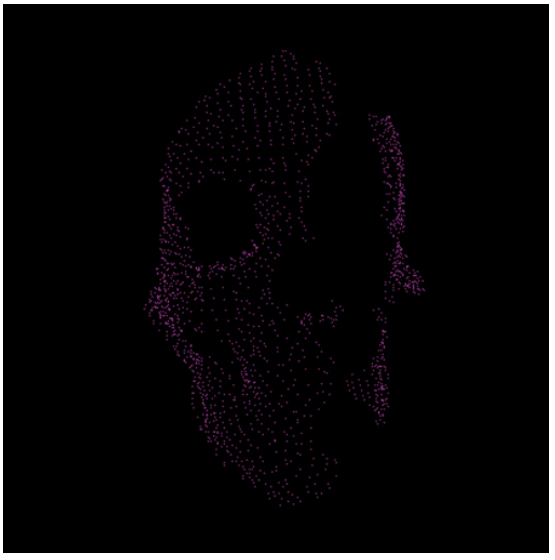

FIGURE 4.6: *skull.*

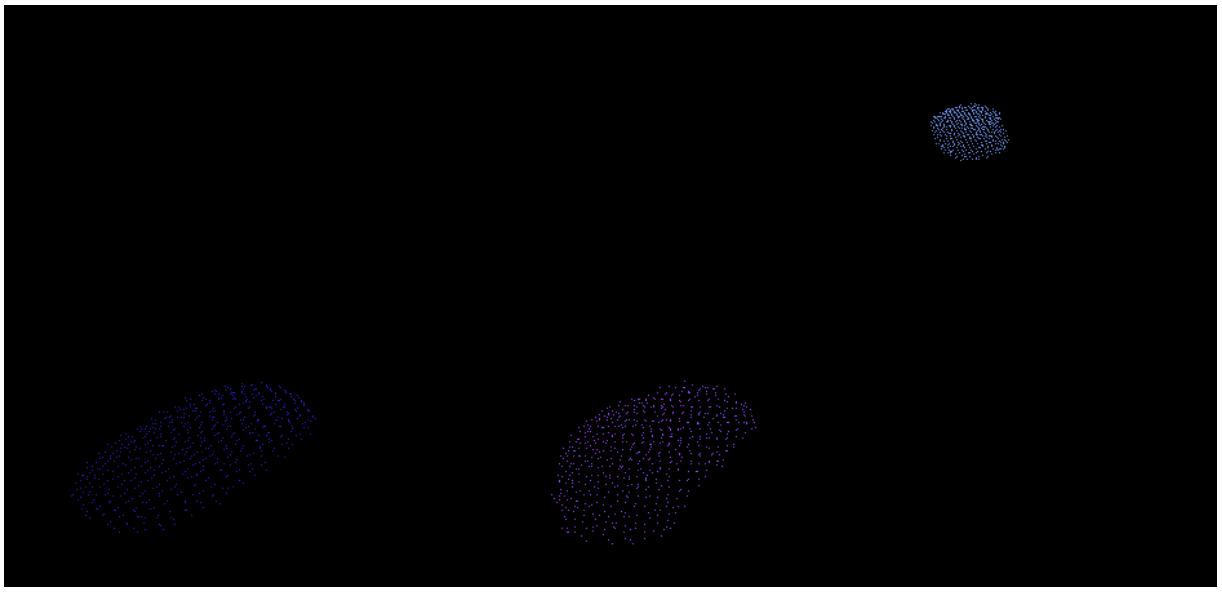

FIGURE 4.7: *stone_0, stone_1, stone_2.*

object analyzed already segmented from the background. Names reported in captions are the file names of the original clouds used for the tests.

Except for the stones point cloud, all of these point clouds are converted to voxels and fed into the network for classification. As far as the stones point cloud is concerned, in that case the scene has been shown with three stones all together just for sake of simplicity. In fact, from the practical point of view the neural network has been fed with the single stones in the scene. As it is possible to notice the object that we are considering are the typical ones that, as we stated before in Chapter 1, we may be interested to retrieve from the sea floor.

In the next section will be shown the results obtained through classification trying to classify the object just reported in the current section.

4.2 Classification and 3D Reconstruction

In this section will be presented the results of classification performed through the neural network described in Chapter 2. In Table 4.1 are shown the results of the performed classifications. The table is structured in an easy and intuitive way: in every line is reported the name of the cloud used for testing and the relative confidence values obtained.

TABLE 4.1: Results of confidence values

	Confidence Values			
Cloud	Cube	Cylinder	Cone	Sphere
amphora_0	0.9976	0.0008	0.0008	0.0008
amphora_0_body	0.0058	0.9825	0.0058	0.0058
amphora_0_neck	0.0005e-08	0.0005e-08	1	0.0005e-08
amphora_1	0.0001e-1	0.0001e-1	0.9997	0.0001e-1
amphora_1_body	0.9870	0.0081	0.0023	0.0023
broken__amphora	0.0005e-01	0.9998	0.0005e-01	0.0005e-01
skull	0.25	0.25	0.25	0.25
black_box	0.9569	0.0143	0.0143	0.0143
box_0	0.9993	0.0002	0.0002	0.0002
stone_0	0.0004e-02	0.0004e-02	0.9999	0.0004e-02
stone_1	0.0001	0.0002	0.9996	0.0001
stone_2	2.4854e-11	2.4854e-11	1	2.4854e-11
sphere_0	0.2801e-17	0.2801-17	0.2801e-17	1
sphere_1	0.3435e-13	0.3435e-13	0.3435e-13	1

4.2.0.1 Results: *amphora_0*

The first result obtained is a high value on the confidence value of *amphora_0* referred to the cube. In this case the confidence values results to be high enough to classify it as a cube. The results after reconstruction then will result to be like the one showed in Figure 4.8.

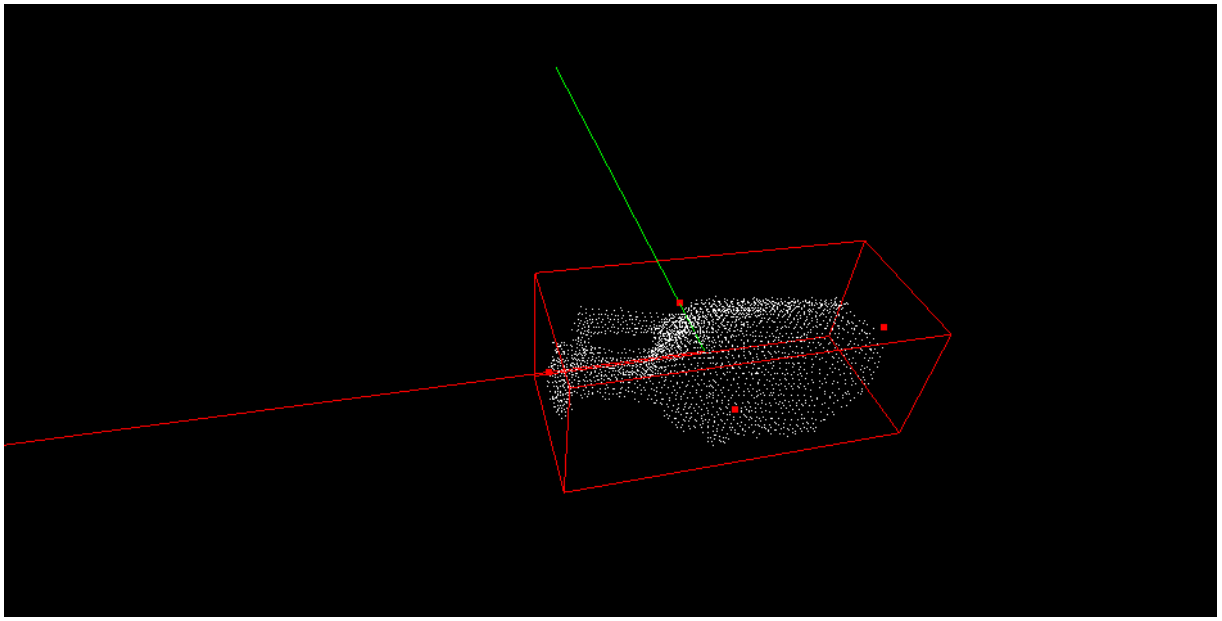

FIGURE 4.8: Reconstruction of a cube over *amphora_0*.

In this case the grasping points detected are four: two couple of them. Since the amphora is not perfectly classifiable as a cube, two of these points (the ones coming out from the intersection of the red line with the two bounding box faces) would not prove to be very robust to guarantee a robust grasp. Although the neural network classified with very few margin of error the cloud, we decided to segment manually the cloud to see if, in that way, it is possible to create a more precise 3D model. The results are shown in the line of *amphora_0_body* and *amphora_0_neck*. In fact these two point clouds are the ones related to the segmented cloud and in particular to the neck and the body of the amphora previously seen. In this case the network classified as a cylinder the body of the amphora and as a cone its neck. Following this classification, 3D reconstruction of the latter clouds produces results reported in Figure 4.9 and Figure 4.10.

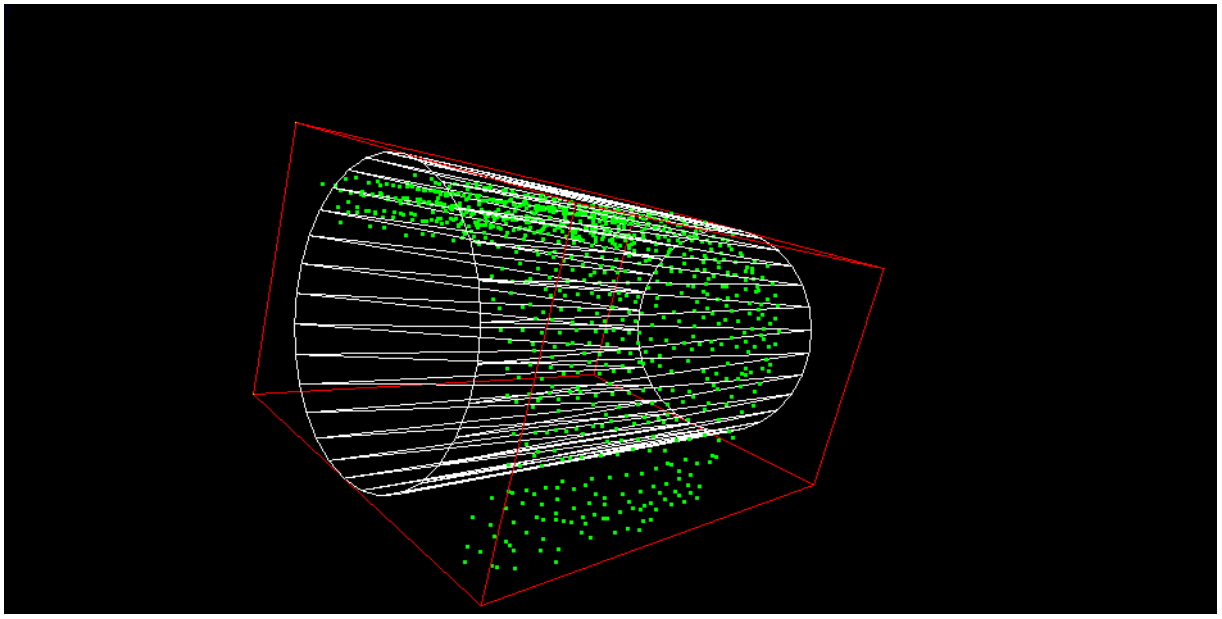

FIGURE 4.9: Reconstruction of a cylinder over *amphora_0_body*.

As it is possible to notice in this case the classification worked better probably because of the isolation of the two parts. Even reconstruction worked fine giving an idea of what the 3D model should look like after the correct classification. In this case, knowing that the center of mass of the amphora is located inside the cylinder the point of grasping will be found on it guaranteeing a robust grasp for the amphora.

4.2.0.2 Results: *broken_amphora*

Another similar result has been obtained with the cloud named *broken_amphora*. In this case the obtained result shows how it was possible from the beginning to classify the shape of the cloud as a cylinder. In this case the reconstruction proved to be correct but

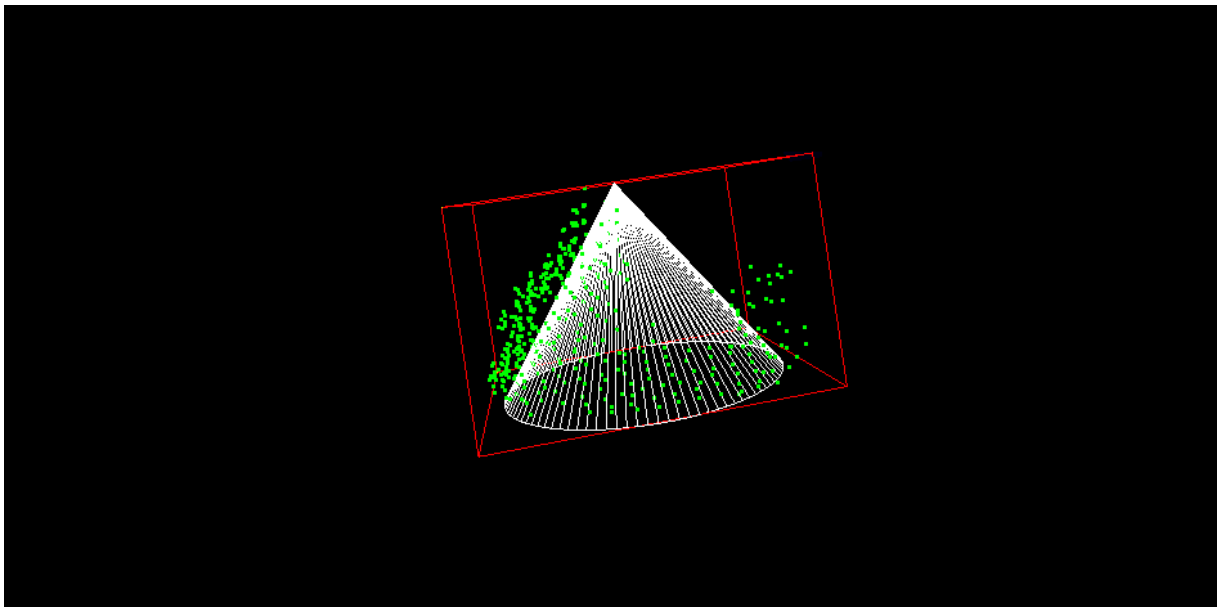

FIGURE 4.10: Reconstruction of a cone over *amphora_0_neck*.

showing a small error on the orientation of the cloud probably due to the hole present on a side of the amphora that slightly modified the orientation and the center of mass. In Figure 4.11 is shown the reconstruction performed on the cloud.

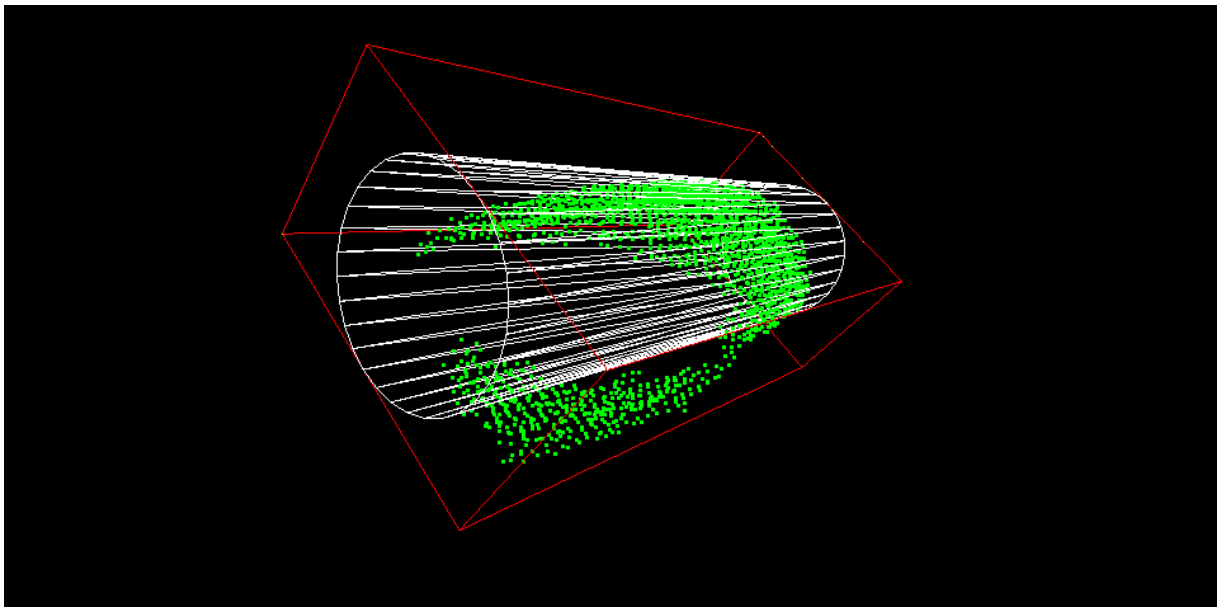

FIGURE 4.11: Reconstruction of a cone over *broken_amphora*.

Also in this case, like the previously analyzed amphora, although there is a small difference in the model and the point cloud orientation, the point of grasping will be found looking for two points aligned with the center of mass and lying on the cylinder surface.

4.2.0.3 Results: *box_0*

Regarding the point cloud of a box (*box_0*) it is possible to see from Table 4.1 that it has been correctly classified as a cube with a high value of confidence. In this case the reconstruction worked correctly showing the model of the box overlapped to its point cloud. In Figure 4.12 is shown the results of the reconstruction where it is possible to appreciate the precision of it in this simple case.

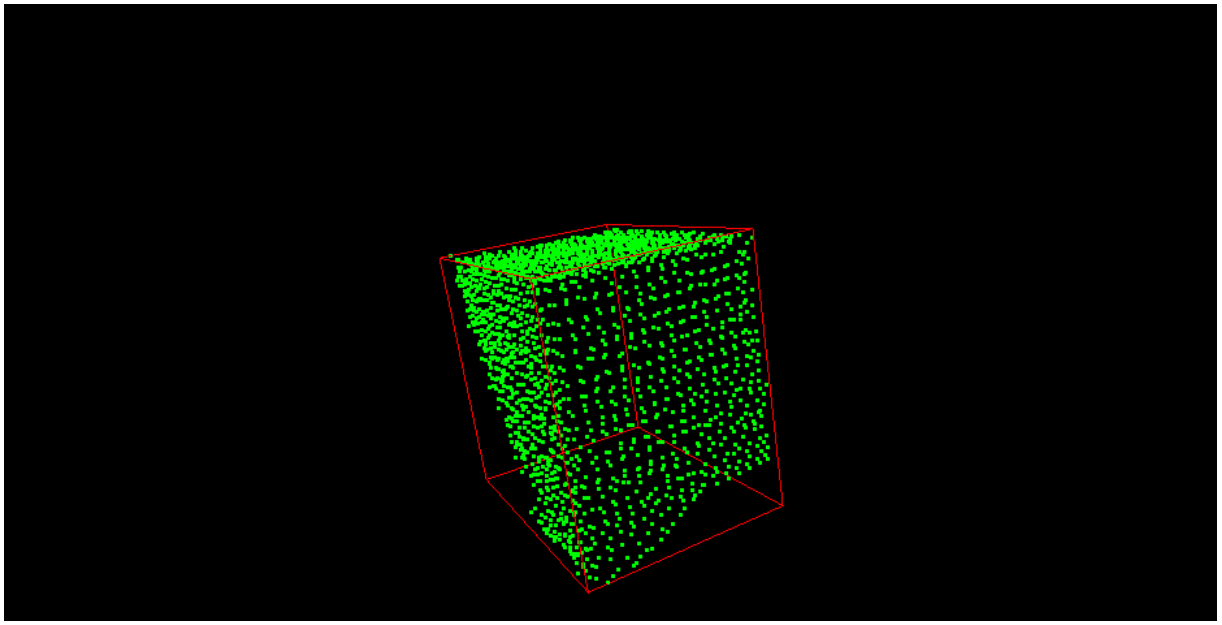

FIGURE 4.12: Reconstruction of a cube over *box_0*.

In this particular case the grasping point will be easily spotted looking for them as aforementioned in Section 3.2.5.

4.2.0.4 Results: *black_box*

Another cloud similar o the simple box has been tested. In this particular cloud it is present a black box of a plane. For some reasons related to the topology of the cloud the segmentation didn't remove the whole box but kept instead part of the floor. This luckily didn't affect the classification that proved to be effective enough to detect the shape of a cube. However the reconstruction of the cube didn't end in a successful way due to the aforementioned partially failed segmentation. In Figure 4.13 is shown the results of the reconstruction. 5As it was possible to imagine the bounding box for reconstruction was not able to be set in a correct way. This leaded to a wrong bounding of the point cloud that lead to a wrong reconstruction of the shape in the scene.

4.2.0.5 Results: Spheres

Since no spherical shapes have been found in the given laser scanned scenes, in order to test if the neural network is able to classify spheres, a couple of manually generated spheres has been tested. In this case since the shapes are generated directly voxelized there are no point cloud to show. The input spheres have been generated manually to avoid to use spheres that where already present in the training set. Using these new shapes we tried to see if the network was able to recognize them. The results on the Table shows a successful behavior for this case showing how the confidence values peak on the element of the vector related to the sphere.

4.3 Failure Modes

In the previous section have been presented results relative to the classification and reconstruction of the network. Few of those results proved to be wrong due to a bad classification or to a bad reconstruction of the cloud. In this section are going to be presented the wrong behaviors encountered along the tests and why these problems occur in any given situation.

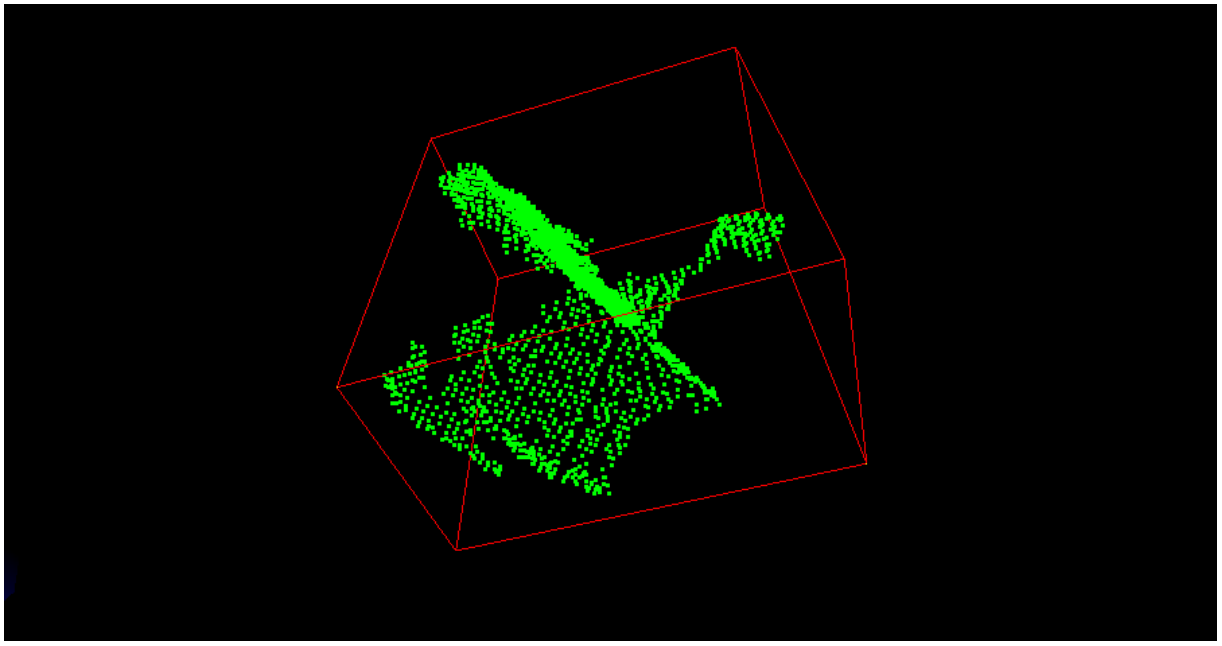

FIGURE 4.13: Reconstruction of a cube over *black_box*.

4.3.1 Bad classification

As it is possible to see from Table 4.1 there are few cases in which the classification is not having the expected result. A case of this was already reported before in Section 4.2.0.1 where at the first iteration the amphora was classified as a cube. Probably this result came out from a bad voxelization of the cloud where the upper part of the amphora, the one including the neck, was probably cut off by the low resolution of the conversion. For this reason the information about a feature that should help to discriminate the shapes has been misinterpreted leading to a bad classification.

Another case of bad classification and thus bad reconstruction occurred with the clouds containing few stones. The stones were classified singularly and not in group as shown in the scene in Figure 4.7. In this case the expected shape may be to a human eye a small cylinder with a large base and a very small length. However the features contained in the clouds analyzed led the neural network to classify them as a cone. This may be addressed to the fact that the stones have a small curvature on the borders and this probably was interpreted by the network as the typical feature of a cone. In the stones cases the reconstruction failed due to the bad classification and the result obtained is shown in Figure 4.14 where it is clear that the cone is not overlapped correctly to the stone.

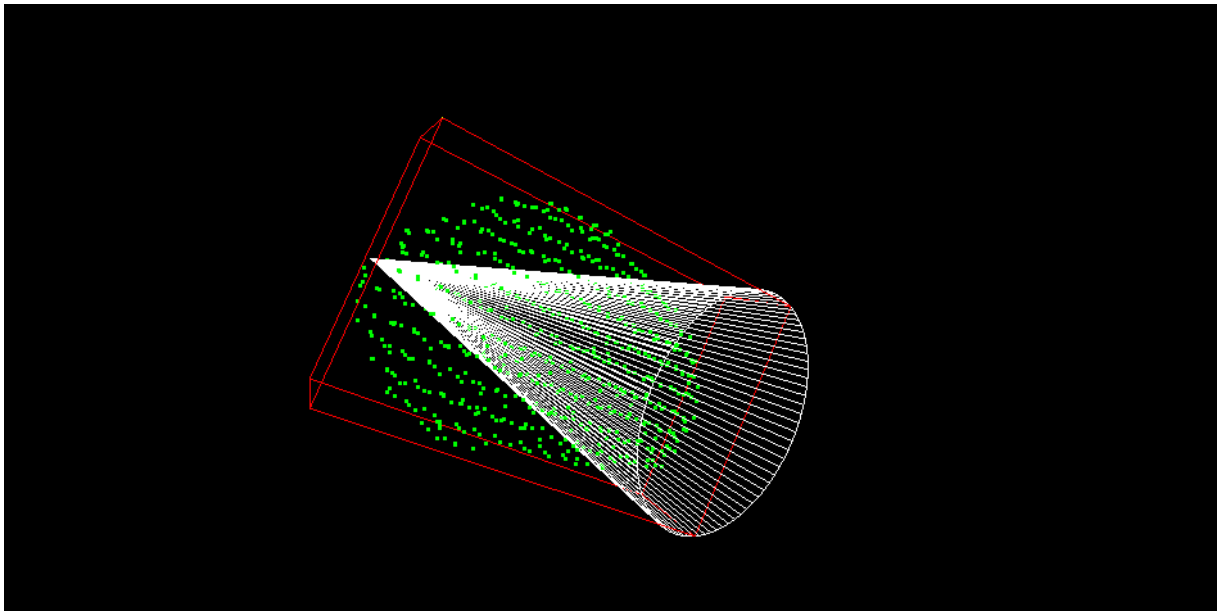

FIGURE 4.14: Failed reconstruction on amphora.

A particular case occurred with the cloud of the skull. As it is possible to see from the Table 4.1 the neural network wasn't able to test correctly classify the skull generating de facto a vector of class values equally distributed ([0.250.250.250.25]). In this case a the

object has been segmented again following the routine described in Section 3.2. However the obtained result from this segmentation didn't improve the final classification scores. In Figure 4.15 it is possible see the results of the further segmentation performed on the skull.

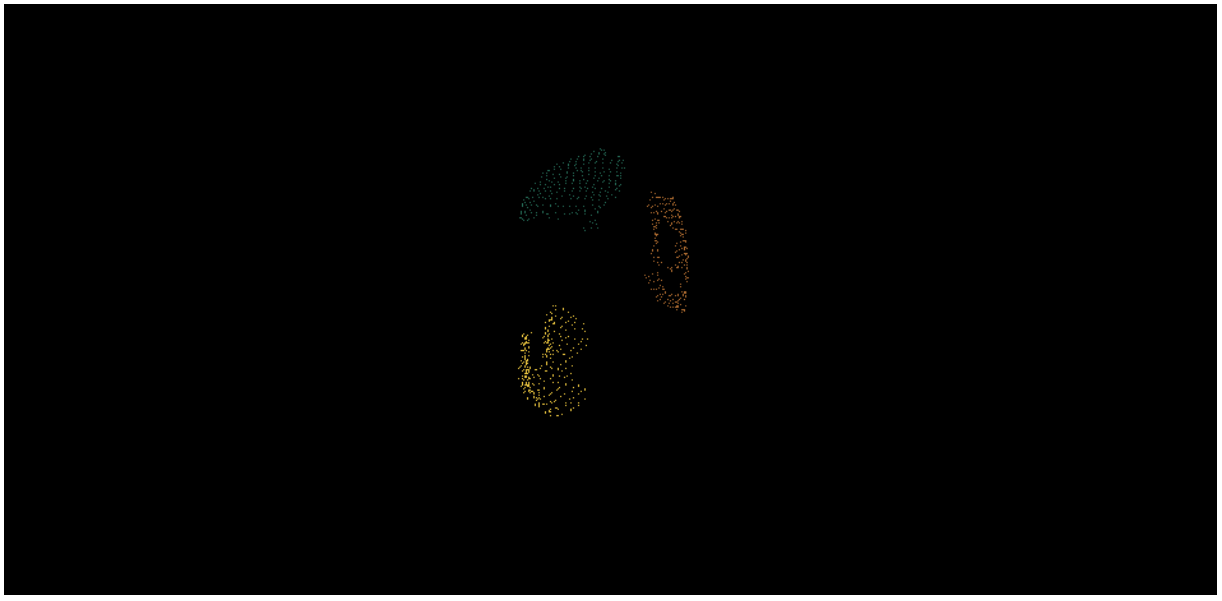

FIGURE 4.15: Region growing segmentation on skull.

From this result it is clear that the most of information coming from the segmentation have been lost due to the particular curvature values of the cloud. The skull case, finally proved to be the most difficult to manage due to its particular topology.

4.3.2 Bad Reconstruction

Another bad classification occurred with the cloud named *amphora_1* showing an amphora bigger than the previously analyzed. In this case the cloud was input entirely to the neural network and the first result was its classification as a cone. Watching the point cloud actually the cone is the most similar shape that a human eye can find. Although the classification succeeded in its part, the reconstruction failed due to the particular shape that the amphora has. In Figure 4.16 is shown how the amphora has been reconstructed in a wrong way.

In this case the amphora had a center of mass that proved to be closer to the wrong side of the bounding box, this lead to a bad reconstruction behavior where the orientation error is easy to notice.

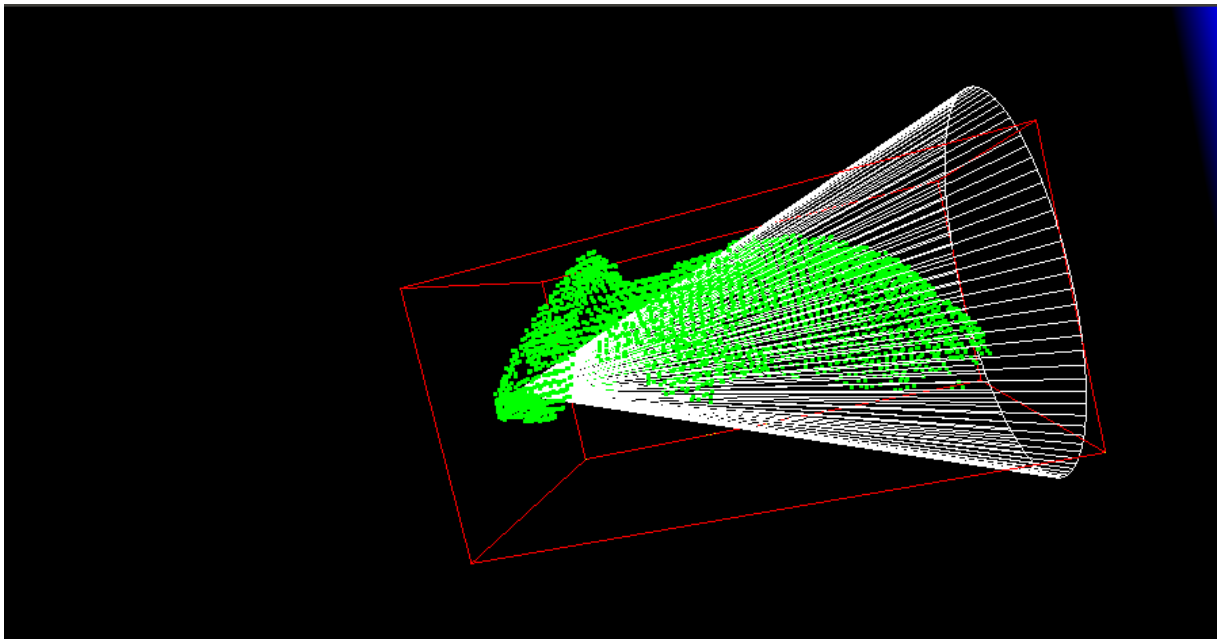

FIGURE 4.16: Failed reconstruction on amphora.

Chapter 5

Conclusions

The project realization allowed me to face the fundamental problems in designing a neural network and in point cloud processing. In particular, I became aware of all those problems regarding the choices that defines the structure of a network for a specific purpose such as the classification could be. To this aim the knowledge acquired along the master, and in particular in the last year, about artificial intelligence and machine learning proved to be fundamental for the development of the project. In fact the previous work reported in [30] offered me the basis to approach the neural network problem in a organized way to better asses the learning phase. Moreover the TensorFlow libraries offered me the occasion to face in a more focused way the Python language that proved to be particular efficient for our specific purposes. On the other hand working with PCL made my knowledge deeper in either point cloud processing and in the supported C++ language.

Although the approach to the problem is innovative and may seem effective, the obtained results at the end didn't satisfy completely the initial expectations. Probably the best suggestion for future works could be to put more efforts on the neural network in order to guarantee a robust and effective classification in order to have a solid base for the 3D reconstruction routine to work on.

In summary this project presented a new approach to a recent problem that nowadays proves to be still a critical point for robotics: detecting optimal grasping points of unknown object. A neural network able to classify shapes within four classes has been presented and proved to be effective on real data in situation where the shapes in the cloud are well defined. In fact the neural network failed few times in classifying those shapes that are confusing or that are difficult to classify even to an human operator. In addition the neural network proved to be very precise either for the good classification or the bad. This is due to the fact that during training the cost function tended to get so low that

the output of it has always a high peak on one of the four class values leaving no margin of doubt in the classification. This would be a nice feature of the network if it proved to be effective in all case. However this has to be seen as an obstacle in classification since, having no lower values than the peaks, it is not possible to apply the point cloud processing routine liable to detect further feature in order to better classify the shapes of objects.

On the other hand, regarding point cloud processing, many algorithms used for segmentation and reconstruction have been presented and they proved to be effective in almost all the cases that have been taken into account or the tests. Only a case proved to be a failure (shown in Figure 4.16) due to the particular shape of the cloud.

In conclusion a theoretical approach to detect optimal grasping points has been presented and a first practical algorithm has been provided. This approach involved the use of either machine learning and point cloud processing concepts taking advantage of the most recent instruments like PCL and TensorFlow. In particular with the latter one it has been decided to try to use a new feature that have been already mentioned before: the 3D convlutional layers which allowed us to manage with data structure such as point cloud.

In the next section some suggestion for further analysis will be provided for a future development of the project.

5.1 Future Development

After our conclusions we wanted to point out some possible future development for further improvements of this project. For this reason in this section will be briefly show some possible concepts that may help to increase the effectiveness of the process.

Different neural network structure In this project has been tested a neural network easy to set up made mainly of convolutional, pooling and densely connected layer. As aforementioned, the convolutional layers used were new in the TensorFlow libraries and were providing a good analysis regarding data having a 3D structure such as point clouds. However it is possible to set infinite different structure for the networks. A possible solution in this case would be to set three different networks working with 2D convolutional network analyzing the dimension of the clouds like a sequence of images. In this case it would be possible to accelerate the learning process and to detect further features contained into clouds.

Dataset In our work the dataset used, described in Section 2.1.4, was generated by a C++ routine exploiting PCL to create all the different shapes used. A suggestion for further works on this project could be to try to work with different dataset. During development, more datasets have been used and few of them proved to be unsuccessful especially those ones containing too many elements. In those cases the training led the network to be overtrained being unable to classify anything, looking for features too precise to appear in a real scene. For this reason a good idea for the future would be to use a rich dataset without using too many figures in order to find a good trade of between overtraining and overfitting due to lack of data.

Reconstruction and voxel resolution As far as reconstruction is concerned a good idea for the future would be to use a Random Sample Consensus (RANSAC) to better reconstruct shapes basing on more feature of the point cloud and not only on their occupation in space.

Another suggestion regarding point cloud processing could be to try different shapes for voxel conversion of the clouds. In fact, as aforementioned in Section 4.3, there is reason to believe that more of the classification errors are due to the voxel conversion that, in our case, led to a distortion of the cloud making impossible to classify correctly the shapes coming from the scene.

Bibliography

[1] Jeannette Bohg, Antonio Morales, Tamim Asfour, and Danica Kragic. Data-driven grasp synthesis-A survey. *IEEE Transactions on Robotics*, 30(2):289–309, 2014. ISSN 15523098. doi: 10.1109/TRO.2013.2289018.

[2] Russell H. Taylor, Arianna Menciassi, Gabor Fichtinger, Paolo Fiorini, and Paolo Dario. *Medical Robotics and Computer-Integrated Surgery*, pages 1657–1684. Springer International Publishing, Cham, 2016. ISBN 978-3-319-32552-1. doi: 10.1007/978-3-319-32552-1_63. URL https://doi.org/10.1007/978-3-319-32552-1_63.

[3] Pedro J. Sanz, Pere Ridao, Gabriel Oliver, Giuseppe Casalino, Carlos Insaurralde, Carlos Silvestre, Claudio Melchiorri, and Alessio Turetta. Trident: Recent improvements about autonomous underwater intervention missions. *IFAC Proceedings Volumes*, 45(5):355 – 360, 2012. ISSN 1474-6670. doi: http://dx.doi.org/10.3182/20120410-3-PT-4028.00059. URL http://www.sciencedirect.com/science/article/pii/S1474667016306280. 3rd IFAC Workshop on Navigation, Guidance and Control of Underwater Vehicles.

[4] Jürgen Schmidhuber. Deep Learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. ISSN 18792782. doi: 10.1016/j.neunet.2014.09.003. URL http://dx.doi.org/10.1016/j.neunet.2014.09.003.

[5] Edward Rosten and Tom Drummond. *Machine Learning for High-Speed Corner Detection*, pages 430–443. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-33833-8. doi: 10.1007/11744023_34. URL https://doi.org/10.1007/11744023_34.

[6] A. A. Moosavian H. Mesgari, F. C. Samavati, H. E. S. Jazeh. A Neural Network Approach for Optimal Grasp Planning. In *2011 IEEE International Conference on Control, Instrumentation and Automation (ICCIA)*, volume 8, pages 859–864, 2011. ISBN 9781467316903.

[7] LeCun Y., Bengio Y., and Hinton G. Deep learning. *Nature*, 521(7553):436–444, 2015. ISSN 0028-0836. doi: 10.1038/nature14539.

[8] I. Lenz, H. Lee, and A. Saxena. Deep learning for detecting robotic grasps. *The International Journal of Robotics Research*, 34(4-5):705–724, 2015. ISSN 0278-3649. doi: 10.1177/0278364914549607. URL http://ijr.sagepub.com/content/34/4-5/705.short.

[9] Edward Johns, Stefan Leutenegger, and Andrew J. Davison. Deep Learning a Grasp Function for Grasping under Gripper Pose Uncertainty. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016. ISBN 9781509037612. URL http://arxiv.org/abs/1608.02239.

[10] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. Software available from tensorflow.org.

[11] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[12] K. Huebner, S. Ruthotto, and D. Kragic. Minimum volume bounding box decomposition for shape approximation in robot grasping. In *2008 IEEE International Conference on Robotics and Automation*, pages 1628–1633, May 2008. doi: 10.1109/ROBOT.2008.4543434.

[13] Andreas ten Pas and Robert Platt. *Using Geometry to Detect Grasp Poses in 3D Point Clouds*, pages 307–324. Springer International Publishing, Cham, 2018. ISBN 978-3-319-51532-8. doi: 10.1007/978-3-319-51532-8_19. URL https://doi.org/10.1007/978-3-319-51532-8_19.

[14] G. De Novi, C. Melchiorri, J.C. Garcianda, P.J. Sanz, P. Ridao, and G. Oliver. New approach for a reconfigurable autonomous underwater vehicle for intervention. *Aerospace and Electronic Systems Magazine, IEEE*, 25(11):32 –36, nov. 2010. ISSN 0885-8985. doi: 10.1109/MAES.2010.5638803.

[15] N. Palomeras, A. Peñalver, M. Massot-Campos, G. Vallicrosa, P. L. Negre, J. J. Fernández, P. Ridao, P. J. Sanz, G. Oliver-Codina, and A. Palomer. I-auv docking

and intervention in a subsea panel. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2279–2285, Sept 2014. doi: 10.1109/IROS.2014.6942870.

[16] A. Palomer, P. Ridao, D. Ribas, and G. Vallicrosa. Multi-beam terrain/object classification for underwater navigation correction. In *OCEANS 2015 - Genova*, pages 1–5, May 2015. doi: 10.1109/OCEANS-Genova.2015.7271587.

[17] P. J. Sanz, A. Peñalver, J. Sales, D. Fornas, J. J. Fernandez, J. Perez, and J. Bernabe. *GRASPER: A Multisensory Based Manipulation System for Underwater Operations*, pages 4036–4041. Oct 2013. doi: 10.1109/SMC.2013.689.

[18] M. Prats, J. Pérez, J. J. Fernández, and P. J. Sanz. An open source tool for simulation and supervision of underwater intervention missions. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2577–2582, Oct 2012.

[19] Sanjay K. Dhurandher, Sudip Misra, Mohammad S. Obaidat, and Sushil Khairwal. Uwsim: A simulator for underwater sensor networks. *SIMULATION*, 84(7):327–338, 2008. doi: 10.1177/0037549708096606. URL `http://dx.doi.org/10.1177/0037549708096606`.

[20] Vision-based Object Handling. Grasping the Not-So-Obvious. *Grasping the Not-So-Obvious*, (September):44–52, 2005. ISSN 1070-9932. doi: 10.1109/MRA.2005.1511868.

[21] Johannes Speth, Antonio Morales, and Pedro J. Sanz. Vision-based grasp planning of 3D objects by extending 2D contour based algorithms. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pages 2240–2245, 2008. doi: 10.1109/IROS.2008.4650632.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

[23] Zhe Wang, Hong Liu, Yueliang Qian, and Tao Xu. *Real-Time Plane Segmentation and Obstacle Detection of 3D Point Clouds for Indoor Scenes*, pages 22–31. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-33868-7. doi: 10.1007/978-3-642-33868-7_3. URL `https://doi.org/10.1007/978-3-642-33868-7_3`.

[24] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *Iclr*, pages 1–15, 2015. ISSN 09252312. doi: http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503. URL `http://arxiv.org/abs/1412.6980`.

[25] A. T. Hadgu, A. Nigam, and E. Diaz-Aviles. Large-scale learning with adagrad on spark. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 2828–2830, Oct 2015. doi: 10.1109/BigData.2015.7364091.

[26] Yann Lecun and Corinna Cortes. The MNIST database of handwritten digits. URL `http://yann.lecun.com/exdb/mnist/`.

[27] Y. Ioannou, B. Taati, R. Harrap, and M. Greenspan. Difference of normals as a multi-scale operator in unorganized point clouds. In *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization Transmission*, pages 501–508, Oct 2012. doi: 10.1109/3DIMPVT.2012.12.

[28] Radu Bogdan Rusu. Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments. *KI - Kunstliche Intelligenz*, 24:345–348, 2010. ISSN 09331875. doi: 10.1007/s13218-010-0059-6.

[29] Ruwen Schnabel and Reinhard Klein. Octree-based Point-Cloud Compression. *Spbg*, pages 111–120, 2006. doi: 10.2312/SPBG/SPBG06/111-120.

[30] Javier Perez, Aleks C. Attanasio, Nataliya Nechyporenko, and Pedro J. Sanz. *A Deep Learning Approach for Underwater Image Enhancement*, pages 183–192. Springer International Publishing, Cham, 2017. ISBN 978-3-319-59773-7. doi: 10.1007/978-3-319-59773-7_19. URL `https://doi.org/10.1007/978-3-319-59773-7_19`.