3D POINT CLOUD CORRESPONDENCES USING DEEP LEARNING

A Degree Thesis
Submitted to the Faculty of the
Barcelona School of Telecommunications Engineering
Universitat Politècnica de Catalunya
by
Javier de la Rica Palma

In partial fulfilment
of the requirements for the degree in
AUDIOVISUAL SYSTEMS ENGINEERING

Advisor: Javier Ruiz Hidalgo

Barcelona, May 2018
ABSTRACT

The current project arises from the motivation of studying the tools of neural networks, along with the application of the knowledge acquired during the courses in the degree in a work that brings together the training of neural networks using three-dimensional point clouds.

In the work, a system has been created, which from a previous database, computes a larger dataset making use of the information obtained from each of the point clouds key points. The objective of the project is to find correspondences between points from two three-dimensional point clouds using neural networks training.

On the one hand, as mentioned, the database of three-dimensional point clouds of the same scene has been generated from different points of view with ground truth, which later is used to compare patches of all the point clouds, in order to find correspondences between them.

On the other hand, a neural network is generated to be trained with the previously computed database as input, and observe the behavior of the comparison of point clouds given the key points mentioned above.

All in all, the main objective of the project is the study of the behavior of neural networks in a specific case, for a possible application of augmented reality (AR). Therefore, the present work is a good basis for a possible alternative work, capable of generating an augmented reality application in which three-dimensional objects are inserted into two-dimensional images.
RESUM

El present projecte sorgeix de la motivació de l'estudi de les eines de xarxes neuronals, juntament amb l'aplicació dels coneixements adquirits durant els cursos en el grau en un treball que ajunti l'entrenament de xarxes neuronals mitjançant núvols de punts tridimensionals.

Al treball s'ha realitzat un sistema que genera, a partir d'una base de dades prèvia, una base de dades més gran fent ús de la informació obtinguda a partir dels punts clau de cada núvol de punts. L'objectiu principal del treball és trobar correspondències entre punts de dos núvols de punts tridimensionals fent ús de l'entrenament de xarxes neuronals.

D'una banda, com s'ha comentat, s'ha generat la base de dades de núvols de punts tridimensionals d'una mateixa escena des de diferents punts de vista.

D'altra banda, s'ha generat una xarxa neuronal per a ser entrenada amb aquesta base de dades com a entrada, i observar el comportament de la comparació de núvols de punts donats els punts clau esmentats anteriorment.

En general, l'objectiu principal del projecte és l'estudi del comportament de les xarxes neuronals en un cas concret, per a una possible aplicació de realitat augmentada (AR). Per tant, el present treball és una bona base per a un possible projecte alternatiu capaç de generar una aplicació de realitat augmentada en la qual objectes tridimensionals són inserits en imatges bidimensionals.
RESUMEN

El presente proyecto surge de la motivación del estudio de las herramientas de redes neuronales, junto con la aplicación de los conocimientos adquiridos durante los cursos en el grado en un trabajo que junta el entrenamiento de redes neuronales mediante nubes de puntos tridimensionales.

En el trabajo se ha realizado un sistema que genera, a partir de una base de datos previa, una base de datos mayor haciendo uso de la información obtenida a partir de los puntos clave de cada nube de puntos.

Por un lado, como se ha comentado, se ha generado una base de datos de nubes de puntos tridimensionales de una misma escena desde distintos puntos de vista.

Por otro lado, se ha generado una red neuronal para ser entrenada con dicha base de datos como entrada, y observar el comportamiento de la comparación de nubes de puntos dados los puntos clave mencionados anteriormente.

En general, el objetivo principal del proyecto es el estudio del comportamiento de las redes neuronales en un caso concreto, para una posible aplicación de realidad aumentada (AR). Por tanto, el presente trabajo es una buena base para un posible proyecto alternativo capaz de generar una aplicación de realidad aumentada en la cual objetos tridimensionales son insertados en imágenes bidimensionales.
Acknowledgements

Let me begin by thanking the advisor of this thesis, Javier Ruiz Hidalgo, for all the support and advice throughout it. Also, for giving me the opportunity to deepen my knowledge in this field, being always a great help.

I would also like to thank Alba Pujol, for all the information and data provided without which I could not have reached the objectives set at the beginning of the thesis, Josep R. Casas, for all the support during the development stage of both the database and the neural network, and Albert Gil, who was a great support with the image server. And at the end, thank to my family, who at all times has trusted me, and for the constant support they have given me throughout the degree, and to all those people who have contributed, in one way or another, to the development of this project.
Revision history and approval record

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>26/03/2018</td>
<td>Document creation</td>
</tr>
<tr>
<td>1</td>
<td>04/04/2018</td>
<td>Document revision</td>
</tr>
<tr>
<td>2</td>
<td>07/04/2018</td>
<td>Document revision</td>
</tr>
<tr>
<td>3</td>
<td>15/04/2018</td>
<td>Document revision</td>
</tr>
<tr>
<td>4</td>
<td>29/04/2018</td>
<td>Document revision</td>
</tr>
<tr>
<td>5</td>
<td>10/05/2018</td>
<td>Final document revision</td>
</tr>
</tbody>
</table>

DOCUMENT DISTRIBUTION LIST

<table>
<thead>
<tr>
<th>Name</th>
<th>e-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Javier de la Rica Palma</td>
<td><a href="mailto:javier.de.la.rica@alu-etsetb.upc.edu">javier.de.la.rica@alu-etsetb.upc.edu</a></td>
</tr>
<tr>
<td>Javier Ruiz Hidalgo</td>
<td><a href="mailto:j.ruiz@upc.edu">j.ruiz@upc.edu</a></td>
</tr>
</tbody>
</table>

Written by:                          Reviewed and approved by:

<table>
<thead>
<tr>
<th>Date</th>
<th>11/05/2018</th>
<th>Date</th>
<th>11/05/2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Javier de la Rica Palma</td>
<td>Name</td>
<td>Javier Ruiz Hidalgo</td>
</tr>
<tr>
<td>Position</td>
<td>Project Author</td>
<td>Position</td>
<td>Project Supervisor</td>
</tr>
</tbody>
</table>
Table of contents

ABSTRACT ................................................................................................................................................. 1

RESUM....................................................................................................................................................... 2

RESUMEN.................................................................................................................................................... 3

Acknowledgements....................................................................................................................................... 4

Revision history and approval record ........................................................................................................... 5

Table of contents ......................................................................................................................................... 6

List of Figures ................................................................................................................................................ 9

List of Tables ............................................................................................................................................... 10

CHAPTER 1 INTRODUCTION ....................................................................................................................... 10

1.1. Project Background ............................................................................................................................. 11

1.2. Project Requirements and Specifications ............................................................................................ 12

1.3. Work Plan ............................................................................................................................................ 13

1.4. Incidents and Delays ............................................................................................................................ 13

CHAPTER 2 FUNDAMENTALS .................................................................................................................... 15

2.1. Artificial Neural Networks .................................................................................................................. 15

2.2. Activation Functions and Dropout ...................................................................................................... 17

2.2.1. Activation Function: Rectified Linear Unit (ReLU) ........................................................................ 17

2.2.2. Activation Function: Sigmoid ........................................................................................................ 17

2.2.3. Dropout .......................................................................................................................................... 18

2.3. Fully Connected Neural Networks ..................................................................................................... 18

CHAPTER 3 STATE OF THE ART ................................................................................................................ 20

3.1. Frameworks ......................................................................................................................................... 20

3.2. Related Works ..................................................................................................................................... 21
CHAPTER 4 METHODOLOGY ........................................................................................................... 22

4.1. Dataset generation .................................................................................................................. 22
4.2. Fully Connected Neural Network design .................................................................................. 26
4.3. Software .................................................................................................................................. 27
4.2. Computation ............................................................................................................................ 28

CHAPTER 5 RESULTS ..................................................................................................................... 30

5.1. Training with freiburg1_desk dataset ...................................................................................... 30
5.2. Test with freiburg1_desk dataset .............................................................................................. 32
5.3. Test with freiburg1_teddy dataset ............................................................................................ 33

CHAPTER 6 BUDGET ....................................................................................................................... 34

CHAPTER 7 CONCLUSIONS .......................................................................................................... 35

Bibliography .................................................................................................................................... 37
Glossary ............................................................................................................................................ 39

APPENDICES .................................................................................................................................. 40
List of Figures

Figure 01. Gantt Diagram with the time and delivery project plan 13
Figure 02. Activation function for an ANN 16
Figure 03. ReLU activation function 17
Figure 04. Sigmoid activation function 17
Figure 05. Standard Neural Net and after applying dropout 18
Figure 06. Fully Connected Neural Network Architecture 19
Figure 07. Two-dimensional and three-dimensional images 23
Figure 08. N points taken within the radius R in two different point clouds 24
Figure 09. Original database structure 25
Figure 10. Network input dataset structure 25
Figure 11. Network architecture 27
Figure 12. [512, 120, 84, 1] and [512, 100, 50, 1] network architectures 31
Figure 13. Training accuracy graphs with [512, 120, 85, 1] architecture and [512, 100, 50, 1] architecture 32
List of Tables

Table 01. Best accuracy of the different freiburg1_desk dataset network architectures. 31
Table 02. Test loss of the different freiburg1_desk dataset network architectures. 32
Table 03. Test accuracy of the different freiburg1_desk dataset network architectures. 32
Table 04. Test loss of the different freiburg1_teddy dataset network architectures. 33
Table 05. Test accuracy of the different freiburg1_teddy dataset network architectures. 33
Table 06. Final Thesis total budget 34
Chapter 1

INTRODUCTION

The design of artificial neural networks (ANN) often consumes a large amount of time and requires the experience and know-how of a limited community of scientists and engineers, as well as huge amounts of data. Reason why, computer vision is one of the most challenging issues of research in the artificial intelligence field.

This thesis is carried out at the Barcelona School of Telecommunications Engineering ETSETB, with the teacher’s help and under the supervision of a teacher advisor.

The main objective of the research is to observe the behavior of an ANN for a possible augmented reality (AR) application in which three-dimensional models can be inserted into two-dimensional images. This application could be very useful in fields such as architecture, where the user could observe a three-dimensional model of a building to be built on an empty lot, or be able to see, using the application, the final result of buildings that are currently and will be a time under construction, as the known building in Barcelona, the Sagrada Familia.

Two-dimensional images from a dataset were transformed into three-dimensional point clouds in which every point had the information about the correspondent image as the location coordinates (x, y, z) and the color information (r, g, b). The dataset used consisted of different images of the same scene, having taken the images from different perspectives. This way, the database provided a necessary ground truth for a comparison of these images in a neural network.

One of the main issues is to create a three-dimensional point cloud database, in which the point clouds correspond to the same object, but from different points of view. Using the points and the information they provide, patches from point clouds referring to the same scene can be compared training a neural network, looking for correspondences between them. That is so, a comparison of three-dimensional points using 3D point clouds is developed.
There are three main goals of the current project, as detailed below:

1. Creating a 3D Point Cloud database to work with. The database used for the project is a given point cloud database, which is implemented and enlarged to be more useful for the current thesis. This database will be computed to be classified between matches as the input of the ANN.

2. Design a deep learning network deciding the necessary parameters and architecture to compare patches from two point clouds of the previous database. The main implemented network is a Fully Connected Network (FCN), in order to observe its performance, and to be able to compare it with another kind of networks.

3. Train the computed neural network providing the database as the input, and test it with a portion of the same database to observe the network performance. At the end, the network is evaluated using a second database that meets the same specifications as the one used during the training, to observe the real performance of the network providing completely new information at the network input.

1.1. Project Background

The initial idea of the thesis was to compute an AR application in which inside a two-dimensional image, a three-dimensional model could be attached. This could be useful in so many applications, as explained before, such as architectural ones.

This early thought was later shared with the project supervisor, with whom deliberating the complication that the work could take to reach the final application, it was decided to do a research project in order to generate a good base, so that in a possible future project, the final AR application could be developed.

In order to know how to focus the project, different fields were studied, as the design, computation, training and test of neural networks, and the design and generation of an enough large database to input the network.

The results from the final master thesis of Alba Pujol[15] were studied in order to integrate them in the current thesis. In that project, a registration between a 2D color
image and unorganized 3D point clouds is proposed, leading to a database in which the current project is based.

1.2. Project Requirements and Specifications

The current project is originated in order to compute a deep learning network able to compare patches from a 3D point cloud dataset and study the performance of different neural networks architectures. However, there are some conditions and requirements the project and the computed scripts may accomplish.

The research is carried out making use of a server provided by the Barcelona School of Telecommunications Engineering ETSETB. All the scripts and files are saved and used in the server.

The neural network (NN) must be able to compare patches from different three-dimensional point clouds, as long as the different possible neural networks must be able to have as input the same dataset, in order to make a proper comparison of the different performances.

At the end of the project, a deep learning network is computed, able to compare patches from different three-dimensional point clouds. This network is mainly computed in Python, using Pytorch, inside the Barcelona’s GPI server.

The final purpose of the project is to compare different network architectures such as FCN with different parameters. It is desired a previously trained network able to compare patches from three-dimensional point clouds. This network should be able to have a better hit rate than failures, being able to compare different point cloud patches by classifying them according to the scenes they belong.
1.3. Work Plan

This section presents the structure of the project, tasks to overcome, a Gantt diagram with the temporary planning (Figure 01) and, in addition, the incidents that arose during its execution.

The project was divided into four main work packages being the database implementation, the neural network implementation, the network evaluation and the documentation. Dates estimations were made at the beginning of the project. However, as the time went by and the project was progressing, some adjustments had to be made with the aim of delivering the project in time and obtaining the desired results.

![Gantt Diagram](image)

**Figure 01. Gantt Diagram with the time and delivery project plan.**

1.4. Incidents and Delays

As mentioned above, as the project was progressing, some difficulties and incidences were found, which delayed the desired delivery date and led to a non-completion of the original work plan. However, the delay was necessary in order to obtain the desired results.
First, one of the main difficulties encountered during the project was the database generation. The first idea was to compute the database from scratch. Nevertheless, a provided database was remodeled in order to reuse it, making it quite bigger so the NN had enough input data for both the training and testing stages. The main data consisted of 595 point clouds, and two files in order to enlarge the database: a keypoints dictionary with all the key points from all the point clouds and a matches dictionary containing all the matches whether they were true or false. All in all, the generation of the database took more time than expected, as the whole database was meant to be finished in October. The first idea was the scripts to be computed in Matlab. However, with the aim of learning a new programming language, and as the development of the NN was going to be developed using Python – and the whole project was to be developed using the same programming language – it was decided to not compute it in Matlab but in Python, which would ease the following network computation. That was an important decision, which was one of the main reasons why the database computation took longer.

To begin, python basics had to be studied, in order to compute a whole Python script. Then, the generation of the whole database took almost one week to compute. To make sure the script worked properly, two different datasets were computed. One test was made with only 64k matches, which took one day to compute. Then, the whole database with approximately 400k matches, which took four days to compute.

The second main difficulty found along the project was the computation of the NN, as the database had to be adapted in order to be able to read it as the neural network's input. The whole project was being developed using TensorFlow, although it had finally been decided to use PyTorch to develop it, which delayed a bit the project, as the NN had to be remodeled to be able to compute it using this framework. Instead of TensorFlow, PyTorch was finally used as it is relatively new comparing it to TensorFlow, and the documentation and support were found better, as long as several implementations related to the current thesis were found more useful, and PyTorch was found clearer and developer-friendly, providing the ability to define reusable modules and being more flexible and powerful.

The last difficulty in terms of time was the combination of the project development with a full-time job, which also delayed the original plan.
Chapter 2

FUNDAMENTALS

The related work in this thesis is centered in different fields. First, an overview of the general studies on ANN is provided in order to completely understand the computer vision technology and its purposes, specifically in FCN.

In order to understand what deep learning is, a previous definition of what artificial intelligence (AI) is, is needed: AI gives computers the chance to reason by themselves. In order to achieve AI, computers can be either manually or automatically instructed via machine learning.

2.1 Artificial Neural Networks

Deep learning visual systems are based on ANN\cite{8}, a structure vaguely inspired by the biological NN that constitute animal brains, which provides a framework to solve computer vision problems.

By iteratively feeding data information into an ANN, the network is able to learn features of different hierarchical natures. It is based on a collection of connected nodes called artificial neurons. Each connection between them can transmit a signal from one to another. The artificial neuron that receives the signal with information can process it and then signal it to artificial neurons connected to it.

The information in an ANN flows in two ways:

- When the network is learning (training stage).
- Normal operation (once the network is already trained).

Each neural unit individually operates using additional functions. There may be a limited function or threshold in each connection or unit such that the signal must exceed a certain threshold to propagate the information to another neuron.
There are three different layers in a neural network:

1. Input Layer (All inputs are fed into the model).
2. Hidden Layers (There can be more than one hidden layer, used for processing the inputs).
3. Output Layer (The data after processing).

The **input layer** is the communication with the external environment that presents a pattern to the neural network. Its main job is to deal with all the inputs. These inputs get transferred to the hidden layers, and every input neuron should represent some independent variable that has an influence over the output of the neural network.

The **hidden layer** is the collection of neurons which have an activation function applied (Figure 02), being the intermediate layer found between the input layer and the output layer. Its job is to process the inputs obtained by its previous layer, so it is the responsible of extracting the required features from the input data. Although there may be more than a single hidden layer, it certainly does not mean that the more number of layers, the more accuracy it will provide, as a stage comes when the accuracy becomes constant or falls if an extra layer is provided. The number of neurons in each network should also be calculated, as if unnecessary neurons are present in the network, overfitting may occur. Several methods are used which do not provide the exact amount of hidden layers as well as the number of neurons in each hidden layer.

The **output layer** collects and transmits the information according to the designated way to provide it. Its pattern can be directly traced back to the input layer. The amount of neurons in the output layer should be directly related to the type of work the neural network is performing.

![Activation function for an ANN](image)

**Figure 02. Activation function for an ANN.**
2.2 Activation Functions and Dropout

In order to deepen the knowledge about how an ANN work, the definition of the
activation function, and specially those used in the project are detailed below, as well as
the definition of the dropout, also used in the NN improvement step of the project.
Activation functions are used to determine the output of NN computing a weighted sum of
their input, adding a bias and deciding whether the signal should be fired or not.

\[ Y = \sum (\text{weight} \times \text{input}) + \text{bias} \quad (1) \]

2.2.1 Activation Function: Rectified Linear Unit (ReLU)

ReLU is one of the most used activation functions due to the NN performance
improvements by speeding up the training. This function exists in the range of [0, ∞)
(Figure 03), returning 0 if it receives any negative input, and returning back any positive
value (2).

\[ f(x) = \max(0, x) \quad (2) \]

Figure 03. ReLU activation function.

2.2.2 Activation Function: Sigmoid

The main reason why the sigmoid function is used in NN is because it exists
between 0 and 1 (Figure 04). Therefore, it is used for models where the probability as an
output has to be predicted, since the probability only exists between the range of 0 and 1
(3).

\[ f(x) = \frac{1}{1 + e^{-x}} \quad (3) \]

Figure 04. Sigmoid activation function.
2.2.3 Dropout

Dropout refers to randomly dropping out units in a NN during the training stage (Figure 05), which means the network does not consider certain units during the training. Avoiding these random neurons, the network learns more robust features that are useful in conjunction with many different random subsets of those neurons that are not considered, doubling the amount of iterations required to converge, although it reduces the training time for each epoch.

The main objective the usage of the dropout looks for is the reduction of the overfitting, which is when the performance on test set is much lower than the performance on train set because the model fits too much to seen data, and does not properly generalize.

![Figure 05. Standard Neural Net (left) and after applying dropout (left).](image)

2.3 Fully Connected Neural Networks

In this thesis, the order in which the information is provided at the NN input is not relevant. In fact, the order should not be considered when training the NN, which is why a FCN is computed as, unlike a Convolutional Neural Network (CNN)\(^7\), does not take into account the input order. Furthermore, unlike CNN, a FCN does not understand the input data as a two-dimensional matrix, but as a one-dimensional vector.

A FCN is composed, as any ANN, by three different layers; input layer, hidden layers and the final output layer. Although there must be just one input and one output layer, there can be different hidden layers. These hidden layers have their own internal parameters, as the number of neurons and their weights, which influence the behavior of the network and come between the results.
One of the most important features to take into account when working with a FCN is that all the nodes are interconnected, being the simplest FCN a two-node network. Since the number of connections grows quadratically with the number of nodes, this kind of topology does not trip and affect other nodes in the network, which makes it impractical for large networks.

**Figure 06. Fully Connected Neural Network Architecture.**
Chapter 3

STATE OF THE ART

The work in this thesis is focused in a specific field, which is the artificial intelligence and machine learning. The state of the art of this project references first the different software used in the development of the thesis, and then the different projects in which this thesis is based, which provided the necessary information and basis in order to be able to carry out the network implementation.

3.1 Frameworks

In the machine learning and neural networks computation, different frameworks and computing languages can be used. In this section, these different frameworks are detailed, and both TensorFlow and Pytorch were finally used.

As explained above, different frameworks and packages can be used in order to compute, train and test ANN:

Caffe[1] is a deep learning framework from Berkeley (BVLC) implemented in C++, which uses Google’s protocol buffers based model specification and parameter format, supporting several data formats (file systems, leveldb, lmdb, hdf5).
Although it is based in portable models and provides a simple command line interface for training and fine-tuning, it requires lots of dependencies, which makes it tricky to install. Furthermore, it provides no automatic differentiation and is less flexible than other frameworks.

Torch[2] is a scientific computing framework for Lua. Although it has no automatic differentiation built-in, it is very fast and flexible, and used by different important companies such as Facebook, Deepmind and Twitter.

TensorFlow[3][10] is a Google’s new deep learning library, which counts with several constant updates. It is C++ based with first class Python bindings, counting with distributed computing support since April 13, 2016 and a good documentation, being also flexible, which enables defining complex networks quickly and concisely.
3.2 Related works

To develop the project, different works and projects related to the ANN computation and training, along with three-dimensional point clouds segmentation were studied in order to learn to compare patches via ANN. Specifically, Fully Connected Networks are studied for this thesis, basing the research on different developed projects such as:

PointNet\(^5\). Submitted in 2016, this work is carried out in the Stanford University, basing its research on an effective network which provides a unified architecture for applications ranging from object classification, part segmentation, to scene semantic parsing. This work explores different deep learning architectures capable of reasoning about 3D geometric data such as point clouds or meshes. With the results of this project, a proper network architecture could be established in order to compute the desired FCN for the current thesis.

Learning to Compare Image Patches via Convolutional Neural Networks\(^4\). Project carried out at the East Paris University (Universite Paris Est, Ecole des Ponts ParisTech), based in the learning directly from image data. In order to compare image patches, easing the network computation in the current project, providing all the needed information in order to design the FCN.

Point Clouds Segmentation Processes\(^16\). Project developed in Polytechnic University of Catalonia (Universitat Politècnica de Catalunya, BarcelonaTech). It explores the current techniques and processes in order to properly perform point clouds segmentation, which helped with the point clouds understanding for the current project, as the database implemented was based on three-dimensional point clouds.
Chapter 4

METHODOLOGY

This chapter describes the followed stages during the development of the project in order to achieve the initial objectives. The project has been divided in two main tasks: (1) generating a dataset using a provided RGB database from the Freiburg Computer Vision Group, and (2) design and train a Fully Connected Network with the previously computed dataset as input in order to find correspondences between patches\(^4\) from its three-dimensional point clouds.

4.1 Dataset generation

For the first stage of the project, the *freiburg1_desk* dataset from the Freiburg Computer Vision Group\(^{14}\) is used. It is a sequence which contains several sweeps over four desks in a typical office environment (Figure 07). Nevertheless, this dataset contains two-dimensional images, which do not provide the needed information required for the project. That is why the project was mainly developed with the dataset provided by a previous UPC thesis from Alba Pujol\(^{15}\), which contains all the three-dimensional point clouds with all the needed information based on the Freiburg original dataset. From this dataset, Alba’s thesis provided a file with all the key points from the point clouds of all the images, and a file containing all the matches whether they are true or false, which provide the needed information in order to compare the point clouds. These files contain the points with the four coordinates related to the position of the point and its color information \((x, y, z, R, G, B)\), so in order to simplify the implementation of the database generation and further research, the luminance, \(Y\) \((4)\), was computed from the color information, resulting only four coordinates to work with in each point, \((x, y, z, Y)\).

\[
Y = 0.2126R + 0.7152G + 0.0722B \quad (4)
\]

A main script to find a list of points \((x, y, z, Y)\) from a given dataset was computed, in order to obtain different interesting points to input to the final FCN. To do so, a threshold was set in order to classify the neighborhood points of the key points as matches or not matches. From this dataset, selecting similar key points, a radius was set to study its
neighborhood’s points, which could be from 64 to 200 points, computing a down sampling in order to obtain the final list of interesting points.

![Figure 07. Two-dimensional (left) and three-dimensional (right) images.](image)

Later on, taking into account that the work is based on three-dimensional point clouds, the training should be independent from (1) translation, moving the correspondent key point to the origin (0, 0, 0) to be the reference point and all the neighboring points accordingly, and (2) rotation, either taking the main’s vector direction, or computing a rotation function in the network itself, which rotates the point clouds in the database. This way, the network could have more input information in the training stage, and would not determine the training conclusions on the points position.

When generating the database, N neighbors are searched within a radius R given a key point KP, comparing them with the rest of the point clouds.

The final database contains the same information as the matches file, although for each key point it saves 64 points, which causes the database to be even larger. The original freiburg1_desk database had 595 point clouds, which on average had 200,000 points each. The fact that the database was of such a large volume and each cloud had a large number of points, caused the generation of the final database to be somewhat slow. To be precise, this computed database elapsed 4 days and 15 minutes to be fully completed.

The final database was generated with all the matches which had enough neighbors within the radius R, being N = 64 neighbors, and R = 0.05, discarding all the key points which did not have at least 64 neighbors within the radius 0.05, as they did not provide
any useful information for the project. That is why, from all the 456612 matches contained in the *matches.shelf* file, only 434057 were valuable for the database and so were saved, as 22555 matches did not have enough neighbors.

In the figure below, the neighboring extracting according to the correspondent key point is detailed. For each point cloud, the corresponding key point is used as the center of a sphere with radius $R$, extracting a matrix of $4 \times 63$ containing all the 64 neighbors within that radius (Figure 08). This process is performed for each point cloud, and later, every pair of matrices with all the neighbors are the actual network input.

![Figure 08. N points taken within the radius $R$ in two different point clouds.](image)

The final format of the database was a struct of $N$ values, in which every single one had 7 values corresponding to the key points, the matches, and the $N$ corresponding neighbors for each key point (Figure 09).

The following line refers to the format of the *matches.shelf* file. The example means that there is a match between the 107th key point from the 113th point cloud and the 287th key point from the 398th point cloud, and there is a mismatch between the 141th key point from the 113th point cloud and the 135th key point from the 398th point cloud.

`'0113_0398': [(True, (107, 287)), (False, (141, 135))]`
In order to be able to use the database in the NN, it had to be adapted so that it could be used as the network input. That is so, that instead of a struct of N values as the dataset was saved, a 512xN matrix had to be computed (Figure 10).

The network input data was finally two matrices of 4x63, being 64 the neighbors within the radius R, and 4 from the (x, y, z, Y) coordinates. That is so, the input data to the neural network was a matrix of 64x4x2 x N = 512 x N, being N = 434057 the total of matches found on the database generation.

At the end, the database is divided into train, test and validation, in order to be able to evaluate the network behavior. That is why, 80% of the database is saved for the training stage, while the 10% is saved for test, and the final 10% of the database is saved for the validation.
4.2 Fully Connected Neural Network design

Taking into account the needs of the network to be able to provide a good performance, a study of the kind and number of parameters needed to implement a FCN is done. These parameters are such as the number of neighbors N taken to enlarge the database, the radius R, the number of layers L, and the internal network parameters $\omega_L$.

Once the parameters are set and the database is designed and tested, a simple FCN is computed, having two patches from different point clouds as the input. The results of training and validating the network provided the necessary information to be able to play with the neural network and to be able to follow certain criteria when making changes in the network, or apply certain transformations in order to improve the execution and the results obtained.

During the work of the project, it had to be taken into account that the work is done to research the performance of networks with three-dimensional point clouds $(x, y, z, Y)$, being $x$, $y$, and $z$ the location coordinates of each key point, and $Y$ the luminance of the point cloud patch. That is so, the movement and rotation of the point cloud’s objects had to be studied too, as the network should not depend on these cases. To be invariant regarding the translation, the key point of all the point clouds was considered the origin $(0,0,0)$ as a reference. Nevertheless, to be invariant regarding the rotation, two possibilities could be used, as the rotation could be invariant either using the PCA, taking into account the direction of the main vector, or training the neural network rotating the input data.

As explained above, a FCN is implemented, as the order of the inputs are irrelevant. In these networks, the neurons of each layer are connected to every neuron in the previous layer, and each connection has its own weight. Because of the complete connection of every neuron to each other, a FCN is a very expensive neural network in terms of computation, and as every connection has its own weight, it is also expensive in terms of memory.

A three-layers network is computed, the last of which is activated with a sigmoid function, since the network output must be binary, as the result shall be either a match or a mismatch between the key points.

In order to provide the required architecture for the FCN, the following parameters had to be studied to ensure a proper execution of the network:
• Number of layers, $L$
• Number of neurons per layer, $n$
• Internal parameters, $\omega_L$

As explained before, a first three-layers network is computed to observe its behavior, providing the neighborhood of two key points in two different 3D point clouds as the input. The first two layers are fully connected layers activated by a ReLU function, while the last layer is a fully connected layer activated with the sigmoid function. If the training and test results were not as good as expected, the number of layers, together with all the other parameters set at the beginning could be changed.

![Network architecture](image)

**Figure 11. Network architecture.**

To evaluate the proper work of the network, the accuracy and the loss were computed, accumulating the values for each batch, and computing the average for each epoch, saving later all this information in a text file. Furthermore, all the accuracy values are compared with each other, and the best accuracy for each training is saved in a separate file, so the different training architectures can be easily compared.

### 4.3 Software

For all the procedures involving feature extraction using Fully Connected Networks, the PyTorch framework was used. Due to the big amount of data used, in order to exploit all its potential, GPUs are required, which is why in this case, the GPUs from the UPC computing service were used.
The scripts to compute and arrange the database and the computation of the artificial neural network are written in Python, using PyCharm as the main framework, and updated to a public GitHub repository for a better updates traceability.

4.4 Computation

The first step of the project is preparing all the data and adapting it later so that the network is able to read it. As mentioned before, the final database is stored in a text file. Although the network input could be a text file, the dataset has to be arranged so it had the format the network need.

For starters, in the database computation, a cell for each match from the original matches file is kept, although only those matches which their key points have enough neighbours within the radius $R$ are saved. This leads to several empty cells at the end of the dataset, which do not interfere when training the network, as all these last cells are ignored. Nevertheless, the provided `matches.shelf` file includes matches between a key point and itself, which could lead to some overfitting when training the network, reason why the computed database had to be re-arranged in order to provide the best input data possible to the network. To do so, a `database_prepare.py` script is computed, which, after loading the database text file, prepares the data into two numpy arrays, to compare the key points provided, ignoring all the repeated ones. Then, all the data is normalized and splitted into training, validation and test in different files, which later will be the corresponding network inputs.

Next, as mentioned before, a three-layers network is computed in a `networks.py` file. This stage is where the network architecture is decided, providing the internal parameters as the number of neurons and the initial weights of the network. This network is computed as a function to be able to use it from another file either to train or test it. It is then called as from the `train.py` file, which loads the database previously prepared so it can be used as the network input, and provides the necessary information for the network to train. In this file, taking the colour into account, providing dropout or using some changes to the input data as the rotation is possible, using the `datasets.py` file, which is based in a function called `FreiburgRGBDataset(Dataset)`, loading both the data and the labels of the network, previously splitted and saved using the `database_prepare.py` file. Calling this function, the mode such as training, validation and test can be chosen, along with the colour and the desired transform such as rotation or
scale if needed. During the execution of the script, during each epoch a first training is made, and then it evaluates the validation data.

To keep track of the network behaviour, the loss and accuracy are computed for each epoch, accumulating first the results for each batch.

At the end, when the execution is completed, a folder is generated saving two plots in a PNG file, one for the loss, and another for the accuracy, comparing both behaviours for each epoch between training and validation stages. In this folder, a best.txt file with the best accuracy during the computation is saved, and the values of the loss and accuracy for each epoch of both training and validation are saved in a train.txt file.

When the network is trained and all the results are saved, the testing stage gets all the information from the selected training result. The test.py script file is computed, using only those trainings architectures which provided the best accuracy results, as they were the ones with most probability to provide the better testing results.
Chapter 5

RESULTS

This chapter describes the research of the best results and the network behavior, computing different architectures and tests changing the parameters and datasets, computing the loss and accuracy for each epoch in order to properly evaluate the network performance.

On the one hand, to compute the loss for each epoch, the Binary Cross Entropy loss (4) is computed, as the output provided a binary result. Examining the following cross-entropy expression should make the usage of the function clear, for the model parameters $\theta$, labels $y_i$, and predicted probabilities $p_i$:

$$L(\theta) = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)] \quad (4)$$

On the other hand, to compute the accuracy for each epoch, a threshold is set to $\mu = 0.5$, as if the output value is over that threshold, it is labelled with a 1, and if the output value is under that threshold, it is labelled with a 0.

5.1. Training with freiburg1_desk dataset

The internal parameters were changed, although only the parameters of the hidden layers could be modified, as the input data was always the information provided by two 4x63 matrices, so the input parameters had to be equal to 512, and the output could only be equal to 1, as a binary result regarding if it was a match or a mismatch was expected.

While these parameters were changed, also the provided information to the input network was modified. Providing the luminance (Y) information or not, and applying the rotation transform. Due to an overfitting in the first attempts of the training, also dropout was taken into account in different tests.
First, the [512, 120, 84, 1] architecture was trained, changing the different input data as the luminance information, dropout and rotation. Then, the train, test and validation data were changed, in order to train the same architecture with different input data. This way, it was ensured that the results obtained were coherent, since similar results were obtained with different portions of the same database. That is why two results with the same [512, 120, 84, 1] architecture are provided in Table 01. At the end, the internal parameters were set to [512, 100, 50, 1], in order to observe the network behavior with a different architecture.

![Figure 12. [512, 120, 84, 1] (left) and [512, 100, 50, 1] (right) network architectures.](image)

The different accuracy results of the network training using the freiburg1_desk dataset are shown on the following table:

<table>
<thead>
<tr>
<th>Color</th>
<th>Dropout</th>
<th>Rotation</th>
<th>Best Acc [1] 512 120 84 1</th>
<th>Best Acc [2] 512 120 84 1</th>
<th>Best Acc 512 100 50 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>0.6819</td>
<td>0.6712</td>
<td>0.6788</td>
</tr>
<tr>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>0.8013</td>
<td>0.8026</td>
<td>0.8035</td>
</tr>
<tr>
<td>✗</td>
<td>✔</td>
<td>✗</td>
<td>0.6936</td>
<td>0.6922</td>
<td>0.6910</td>
</tr>
<tr>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td><strong>0.8078</strong></td>
<td><strong>0.8075</strong></td>
<td><strong>0.8098</strong></td>
</tr>
<tr>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>0.7979</td>
<td>0.7970</td>
<td>0.7978</td>
</tr>
</tbody>
</table>

Table 01. Best accuracy of the different freiburg1_desk dataset network architectures.
On the one hand, as far as the first results, quite a small difference could be observed, as the obtained accuracy results were similar for both portions of the database, which confirmed a good network training performance. On the other hand, the best results obtained were those networks trained with color information and dropout, but not including the rotation transform, although the difference was quite small between them.

5.2. Test with freiburg1_desk dataset

To do following test stage, as the results in the training stage were so close in both network architectures, the best accuracy for both internal parameters changes were taken into account, as the results were quite similar. That is why, only the dropout and the color information was taken into account in both architectures.

<table>
<thead>
<tr>
<th>Color</th>
<th>Dropout</th>
<th>Rotation</th>
<th>Test Loss 512 120 84 1</th>
<th>Test Loss 512 100 50 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>0.3708</td>
<td>0.3380</td>
</tr>
</tbody>
</table>

*Table 02. Test loss of the different freiburg1_desk dataset network architectures.*

<table>
<thead>
<tr>
<th>Color</th>
<th>Dropout</th>
<th>Rotation</th>
<th>Test Acc 512 120 84 1</th>
<th>Test Acc 512 100 50 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>0.8285</td>
<td>0.8474</td>
</tr>
</tbody>
</table>

*Table 03. Test accuracy of the different freiburg1_desk dataset network architectures.*

Both architectures provided a similar loss and accuracy results, as the less loss the test had, the higher the accuracy was. Nevertheless, the difference between both tests was
quite small, as it provided a difference of about a 0.02 in the accuracy and 0.04 in the loss.

5.3. Test with *freiburg1_teddy* dataset

The *freiburg1_teddy* database was only used for the network test, as the objective was to observe the network behavior with a different input data, which the network had no previous information about, and it was not trained with this database. Nevertheless, only for the architectures which obtained the best accuracy in the training stage, the test with the new database was computed.

<table>
<thead>
<tr>
<th>Color</th>
<th>Dropout</th>
<th>Rotation</th>
<th>Test Loss 512 120 84 1</th>
<th>Test Loss 512 100 50 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>0.4875</td>
<td>0.5040</td>
</tr>
</tbody>
</table>

*Table 04. Test loss of the different freiburg1_teddy dataset network architectures.*

<table>
<thead>
<tr>
<th>Color</th>
<th>Dropout</th>
<th>Rotation</th>
<th>Test Acc 512 120 84 1</th>
<th>Test Acc 512 100 50 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>0.7644</td>
<td>0.7637</td>
</tr>
</tbody>
</table>

*Table 05. Test accuracy of the different freiburg1_teddy dataset network architectures.*

The accuracy during the test with the *freiburg1_teddy* dataset is lower using both architectures (Table 05). As said before, the network had no previous information about this dataset, as it was trained with a completely different dataset, reason why the accuracy during this test stage is lower than testing the network using the same database as in the training stage.
Chapter 6

BUDGET

In this thesis, no prototype realization is being carried out, therefore, this budget section is sparing. All the hours dedicated to the project are taken into account evaluated at cost of an Intern Engineer, and all the expenses in software are also deducted.

The thesis started in September 2017, although some organization issues were established before. Considering an amount of 876 hours dedicated to the whole project, knowing that it took 128 hours to prepare and study the subject, and 108 hours were devoted to documentation, writing and preparation of the final defense, a total of 640 hours have been exclusively devoted to the project. With this information and supposing that an Intern Engineer earns 8 euros per hour, results in a cost of 5.120€.

To carry out the thesis, different software is used, and different tasks and costs have to be established. The main software used to compute the scripts is PyCharm, being this one used inside the Barcelona’s School of Telecommunications Engineering Image Department server. PyCharm has a monthly price of 19,90€ the first year, so for the whole project of 8 months, the total cost is of 159,20€.

All the work has been developed in the Barcelona’s School of Telecommunications Engineering Image Processing Group, using the image server which provides GPUs in order to compute, train and test the network. The access to this server with GPU has a cost of 0,90 euros per hour. For the current thesis, an amount of 380 hours have been devoted to run the software using the GPU, and the CPU has been used a total of 1083 hours. That is so, the total cost of the server usage is about (1083+380)·0,90 = 1.316,70€.

<table>
<thead>
<tr>
<th>ITEM</th>
<th>PRICE</th>
<th>FINAL COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours worked as an Intern Engineer</td>
<td>8€/hour · 640 hours</td>
<td>€ 5.120,00</td>
</tr>
<tr>
<td>PyCharm monthly fee</td>
<td>19,90€/month · 8 months</td>
<td>€ 159,20</td>
</tr>
<tr>
<td>Image Server</td>
<td>0,90€/hour · 1463 hours</td>
<td>€ 1.316,70</td>
</tr>
<tr>
<td>Final Budget</td>
<td></td>
<td>€ 6.595,90</td>
</tr>
</tbody>
</table>

Table 06. Final Thesis total budget.
Chapter 7

CONCLUSIONS

This thesis final project has been motivated by the growing research and need of artificial neural networks over the past years.

One of the objectives at the beginning of the project consisted in acquiring a solid fundamental knowledge in order to identify, form and resolve problems related to the artificial intelligence and machine learning. The main aim of this thesis was to compute a big database from three-dimensional point clouds in order to compare patches from these point clouds using artificial neural networks.

This thesis has first presented the fundamentals and guidelines to compute a fully connected neural network using python with Pytorch as the main framework. Then, the design of both the database and the neural network is presented, using the freiburg1_desk and freiburg1_teddy datasets for the database creation.

The Fully Connected Neural Network is trained using the 80% of the freiburg1_desk dataset, while it is tested and validated with the rest 20% of the dataset, 10% for each. Then, the Network is tested again using the freiburg1_teddy dataset, in order to study the network behavior if a dataset which the network has no information about is used as the input data.

The first training results obtained an accuracy of a bit more than the 50%, which could be considered an arbitrary result, as half of the times the network was failing the matches. That is why different network architectures were designed in order to improve the accuracy. With these different architectures, an accuracy improvement was obtained, from which the one with the best hit rate was chosen for the following testing step. Nevertheless, using two different architectures a similar accuracy was obtained, which led to a testing stage with these two architectures.

By and large, the final test results obtained were quite successful, as the best accuracy was of 0.76 in both network architectures using the freiburg1_teddy dataset. Although these accuracy values were quite smaller than the ones obtained using the original freiburg1_desk dataset, the results were good enough, as the network had no previous information of this freiburg1_teddy database. Nevertheless, these results could be improved, as the 24% of the times the network is failing.
On the one hand, enlarging the data used at the input network, could increase the accuracy. Nevertheless, the database used for the project is provided by another thesis, which could not be modified or enlarged enough in order to test these improvements.

On the other hand, as tried in this thesis, different network architectures and implementations could be tested, as enlarging the amount of layers used, and changing the network internal parameters.

All in all, the objectives proposed at the beginning of the project have been met, since the desired database has been generated, and a neural network with a correct performance has been designed. This has led to a good result of the tests, since it has been achieved, on the one hand, 80% of the hits using the same database in the training and tests stages, and most importantly, 76% hits using a database only in the test stage.

This leads to conclude that the obtained results are satisfactory. Although the accuracy could be improved, there is no possible improvement without a setback, either due to a training slowdown due to a modification of the neural network, or due to an expansion of the database, which would entail a higher information charge, and a computation of a whole new database from scratch.

The results obtained in this thesis could be the starting point of a new possible future work, in order to compute the AR application discussed at the beginning of the document.
Bibliography


## Glossary

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR</td>
<td>Augmented Reality</td>
</tr>
<tr>
<td>PCA</td>
<td>Principal Component Analysis</td>
</tr>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>NN</td>
<td>Neural Network</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
</tr>
<tr>
<td>FCN</td>
<td>Fully Connected Network</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td>KP</td>
<td>Key Point</td>
</tr>
<tr>
<td>ReLU</td>
<td>Rectified Linear Unit</td>
</tr>
</tbody>
</table>
APPENDICES

All the scripts coded for the whole project along with the graphs and results obtained can be found in a GitHub repository by the name of 3DPCDL. It all has been developed in python using Pycharm with Pytorch backend.