RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITÉ DES SCIENCES ET DE LA TECHNOLOGIE
HOUARI BOUMEDIENE

FACULTÉ D'ÉLECTRONIQUE ET D'INFORMATIQUE
DÉPARTEMENT D'INFORMATIQUE

Thèse

DOCTORAT 3 ème cycle (LMD)

Spécialité : INFORMATIQUE

Option : Intelligence Artificielle

Par :

Chayma Zatout

Thème :

Codage sémantique de scène par traitement d'images RGB-D pour une interaction homme-scène

Soutenue devant le jury composé de :

Mme. A. SERIR	Professeur	USTHB	Présidente
M. S. LARABI	Professeur	USTHB	Directeur de Thèse
Mme. H. NEMMOUR	Professeur	USTHB	Examinatrice
M. M. HACHAMA	Professeur	USDB1	Examinateur
M. M. GUIATNI	Professeur	EMP	Examinateur
Mme. N. LAICHE	M.C.A.	USTHB	Examinatrice

Abstract

The aim of this thesis is to provide a scene semantic labeling using RGB-D images for human-scene interaction. We focus on scene semantic labeling for scene understanding. The system takes in consideration the scene geometry, the objects in the scene, their nature, their positions in the scene and the relationship between them.

In this thesis, two main contributions are made. First, a touch-based 3D interface that generates 3D scene labels, is designed. The 3D labels rely on 3D shape recognition and geometric features computing. Second, an end-to-end scene understanding system for human-scene interaction is developed. The system produces 3D labels on the designed 3D interface. The proposition of these latter is motivated by the limitations of current systems for human-scene interaction especially assistive systems.

The pipeline of our main system is: first, the acquired depth image is segmented and each segment is classified using geometric features and/or a deep learning network for semantic classification. Second, inspired by the Braille system and the Japanese writing system Kanji, the obtained classes are coded with semantic labels. The 3D interface is then generated using these labels and the extracted geometric features. Our final system is able to predict more than 17 classes only by understanding the provided illustrative labels. For the remaining objects, their geometric features are transmitted.

Each system's component is validated using public and local datasets. The obtained results are reported, discussed and compared to the state-of-the-art works.

Keywords- *Human-Scene Interaction; Scene Semantic Labeling; 3D Interface; Scene Understanding; 3D Shape Recognition; Assistive Systems; Deep Learning.*

Résumé

Le but de cette thèse est de fournir un étiquetage sémantique de scène utilisant des images RGB-D pour l'interaction Homme-Scène. Nous nous concentrons sur le codage sémantique de la scène pour la compréhension de la scène. Le système doit prendre en considération la géométrie de la scène, les objets de la scène, leur nature, leurs positions dans la scène et la relation entre eux.

Dans cette thèse, deux contributions principales sont proposées et motivées par les limites des systèmes actuels pour l'interaction Homme-Scène en particulier les systèmes d'assistance. La première, est une interface 3D tactile qui génère des codes 3D des objets de la scène, est conçue. Les codes 3D reposent sur la reconnaissance de formes 3D et le calcul de leurs caractéristiques géométriques. La seconde, est le développement d'un système de compréhension des scènes de bout en bout pour une interaction Homme-Scène. Le système produit des codes 3D sémantiques sur l'interface conçue.

Le pipeline de notre système principal est composé des étapes suivantes: (1) L'image de profondeurs acquise est segmentée et chaque segment est classifié en utilisant des caractéristiques géométriques et/ou un réseau d'apprentissage profond. (2) Inspirées du système braille et du système d'écriture japonais Kanji, les classes obtenues sont converties en des codes sémantiques. (3) L'interface 3D est ensuite générée à l'aide de ces codes et des caractéristiques géométriques extraites. Notre système final est capable de prédire plus de 17 classes seulement en comprenant les codes fournis. Pour les autres objets, leurs caractéristiques géométriques sont transmises.

Chaque composant du système est validé à l'aide des datasets publics et de notre Laboratoire. Les résultats obtenus sont rapportés, discutés et comparés à des travaux de l'état de l'art.

Mots clés- *Interaction homme-scène; Étiquetage sémantique de la scène; Interface 3D; Compréhension de la scène; Reconnaissance de forme 3D; Systèmes d'assistance; Apprentissage profond.*

Acknowledgements

First and foremost I am extremely grateful to my supervisor, Professor Slimane Larabi, whose expertise was invaluable in formulating the research questions and methodology. I would also thank him for his invaluable advice, continuous support, and patience during my PhD study. His immense knowledge and plentiful experience have encouraged me in all the time of my academic research and daily life. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

My gratitude extends to my thesis committee president, Professor Serir, for making her time available to read my thesis. I also thank Professor Nemmour, Professor Hachama, Professor Guiatni and Doctor Laiche for their availability in reading and analyzing my work.

I would like to acknowledge my family. My parents, Abi and Oummi, never failed to say that they were proud of me, especially my father who keeps telling me: "Tu es ma fierté". My sister, Lily, gave quiet encouragement and positive belief in my success that kept me going regardless of the challenge that I faced. She is a sister, a friend, a confidant and a brother. I am thankful to my cousins Khadidja, Rahil, Zineb, Hiba and Lyna and my aunt Zozo for their support and for always inviting me to take a break.

I am grateful to my friends and colleagues, Chems Eddine and Farah, I'll never forget your willingness to help me, to encourage me, to believe in my ability to achieve my goals. I really can't find the words to express my gratitude for you for everything

you have done for me, Chems Eddine. I am also thankful to my friends Hayat and Lamia who provided stimulating discussions as well as happy distractions to rest my mind outside of my research.

Table of Contents

List of Algorithms

List of Tables

List of Figures

List of abbreviations

AE Absolute Error

BASISR Blind Assistive System for Intelligent Scene Reading

CAD Computer Aided Design

CDBN Convolutional Deep Belief Network

CNN Convolutional Neural Network

conv. convolutional

DBN Deep Belief Networks

DBSCAN Density-Based Spatial Clustering of Applications with Noise

DCGD Depth Cut based Ground Detection

FN False negative

FP False positive

FPFH Fast Point Feature Histograms

GDIS Ground Detection in Indoor Scenes

HT Hough Transform

ILSVRC ImageNet Large-Scale Visual Recognition Challenge

IR Infrared

LiDAR Light Detection and Ranging

MAE Mean Absolute Error

MSE Mean Square Error

PCL Point Cloud Library

PFH Point Feature Histograms

POI Points Of Interest

PPF Point Pair Feature

RANSAC RANdom Sample Consensus

RMRC Reconstruction Meets Recognition Challenge

RMSE Root Mean Squared Error

ROC Receiver Operating Characteristic

ROS Robot Operating System

RSD Radius-based Surface Descriptor

SVM Support Vector Machine

TN True negative

TP True positive

VFH Viewpoint Feature Histogram

Chapter 1

Introduction

To accomplish daily tasks, humans are in daily interaction with their surroundings. They interact by actions or/and by contact. For instance, we sit on chairs, we eat on tables, we walk on the ground. Some of the actions are easy to accomplish like lying on the bed; some others are more or less difficult and they require an intermediate device or systems like the traffic lights to trace the route or a navigation software to navigate in an unfamiliar places. While obvious, it is important to state that in order to accomplish a given task, humans perceive the scene, understand it and then act accordingly. We call these actions and contacts: Human-Scene Interaction.

To enhance the daily life, many systems for human-scene interaction have been proposed. These systems process the captured scene in order to understand it (identify the captured objects, their geometry and locations and the relationship between them) and to generate the required output like orientation cues. Generally, these systems rely on image processing, artificial intelligence and external sensors to understand a given scene and generate the output. The nature of outputs and the feedback interface have a great importance . In fact, delivering information or instructions is crucial, they have to be clear, concise, simple to understand and without needing high concentration and efforts. In addition, the system interface has to be simple to use.

To transmit instructions, scene description or any generated output, most of systems

for Human-Scene interaction use audio-based or vibration-based interface. In many systems, an audio device is used to transmit information and instructions. It's true that this latter enhances the life quality, but sometimes it prevents the hearing sense to do its usual tasks like detecting instant sound events that can make the user life in danger. An alternative is to use interface based on vibration. It releases the hearing sense from this task, however, it is less informative and can provide a modest number of possible instructions. Furthermore, in order to reduce the amount of the transmitted output, the applications that rely on scene interactions provide, whether local navigation information like the presence of an obstacle straight from the user, short instructions to follow or semantic information for grasping objects. However, we believe that with a good scene semantic labeling transmitted by an interface that is exploitable by touch, a better human-scene interaction reached.

1.1 The goal

Through this thesis, our goal is to propose a scene semantic labeling for a better Human-Scene interaction. We are, more precisely, interested in proposing an aid system for the visually impaired for a better interaction with the scene. The system transmits the output through a 3D interface that allows to have a clear idea on the captured scene. The semantic labeling takes into consideration the free space, the type of objects, their locations and properties. Such coding should also make it easier for the visually impaired and the blind people to understand the content of the scene and also allows them to navigate. In addition, it should be exploitable by the touch, the sense that the visually impaired rely on to recognize objects, in order to release hearing that can be used to detect unsafe instant events.

The touch sense for the visually impaired and blind people, is often used to recognize people, obstacles, objects, and many other stuffs. It simply plays the role of sight. Since long time ago, the blind children have used the touch sense to learn and study

Figure 1.1: Blind children use touch to learn. By touch, they study animals, their shape and their size. SOURCE: Copyright Tyne and Wear Archives and Museums [3].

different subjects in different fields such as science, geography, history, etc. as shown in Figure 1.1.

1.2 Challenges

Our main challenge is to design labels that semantically represent each selected object class. We can see the names of the objects as the ideal labels; however, transmitting the name of every detected object so that can be understandable by touch is crucial and can be ambiguous. Another possibility is to use the first characters as labels, but this latter does not represent the object semantically and can be ambiguous when two classes starts with the same characters. In addition, the labels have to be clear, concise and easy to understand to ensure a good understanding of the scene.

Another challenge is to understand the scene, associate to each object a 3D label and map the 3D scene into a designed device. Exploiting this device by touch provides an interface for Human-Scene interaction. The mapping (scene-device) has to take into consideration the geometry of the scene, the objects present in the scene, their nature, their positioning in the scene and the relationship between them.

Finally, designing and implementing an end-to-end scene semantic labeling system for a human-scene interaction using only a depth image or a point cloud instead of an RGB-D data as the system input makes the task more challenging. Furthermore, it reduces the computation time and the system cost in terms of the required hardware.

1.3 Overview of our system

In this thesis, we propose a 3D interface for scene semantic labeling, Figure 1.2. From the depth image, the 3D scene is processed and semantically mapped into the 3D interface using the computed labels and the extracted geometric features of the input point cloud. Three main modules are proposed: the ground detection module, the classification module and the scene semantic labeling module. The first module detects the ground and extract the occupied space which represents the objects appearing in the scene. The second module is based on computing geometric features and on a deep learning architecture to classify the objects into nine geometric features and seven semantic classes respectively. The scene semantic labeling is inspired from Braille and Kanji systems. This latter is mapped into a touch-based device that can be used for many applications such as indoor navigation and searching and grasping objects.

1.4 Summary of contributions

Our main contributions are:
– **A 3D interface for human-scene interaction** that can be used as an output interface in many applications that are devoted for interacting with the scene. For this purpose, our subcontributions are:

- Two objects classification approaches which are suitable for many applications,

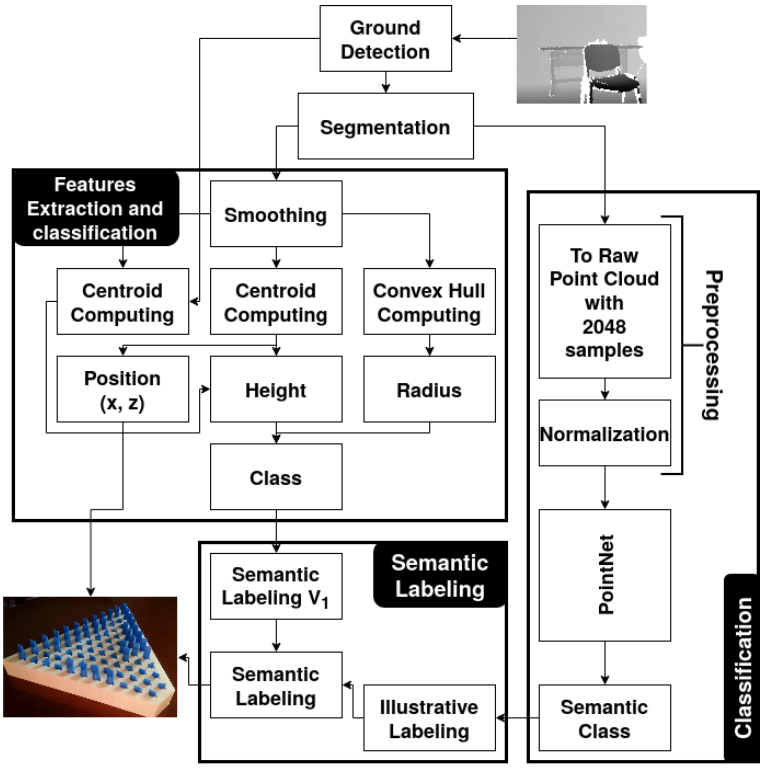

Figure 1.2: Scene semantic labeling system for human-scene interaction.

especially for the visually impaired and blind people. The first approach is based on computing geometric features. The second approach is based on deep neural networks to classify objects into semantic classes. Although we will be using a state-of-the-art neural network, we will show that grouping object classes that have the same usage improves the system accuracy by 5%.

- A scene semantic labeling that is similar to the Braille and Kanji systems, can be mapped on the 3D interface. The proposed scene semantic labeling will provide for users the scene geometry and its content in a simple yet effective way.

- The conception and the design of a device which serves as our 3D interface. Furthermore, it can be used as an output device in many visually impaired assistive systems and for many applications. It is designed to be exploitable

by the touch sense in order to provide concise and informative labels while releasing hearing.

– **An end-to-end semantic labeling system for human-scene interaction** that describes the captured scene and provides geometric features and the nature of the detected obstacles mapped as the touch-based semantic labels. The system takes only a depth image as input to reduce the computational complexity and the system's cost in terms of the required hardware. For this purpose, we also contributed by:

- Depth Cut based Ground Detection (DCGD) algorithm, an efficient algorithm for ground detection. It is based on computing a set of depth cuts from the given depth image. The ground is then detected by removing pixels that correspond to objects after dividing each cut into sub-cuts. The validation has shown that our results exceed the state-of-the-art results.

1.5 Outline of the dissertation

The remaining chapters are structured as follows:
– **Chapter 2** is devoted to review and analyze systems with Human-Scene interaction and their different components. In this chapter, we first present current works in many fields such as indoor navigation, semantic labeling and object avoidance. It allows us to draw their architecture and thus design ours.
– **Chapter 3** introduces the different steps of point clouds processing since one of our challenge was to only use depth images. It first formulates point cloud and then, point cloud filtering, point cloud descriptors, point cloud segmentation and 3D shape recognition are presented respectively.
– **Chapter 4** describes our proposed 3D interface. It first introduces the design of our proposed device and its first prototype that is especially designed for a better Human-Scene interaction. Then, it shows how this latter can be used for as an out-

put device in 3D scene semantic labeling systems.

– **Chapter 5** is dedicated to the proposed 3D scene semantic labeling for a Human-Scene interaction and its different components. Indeed, it describes our proposed scene semantic labeling that is especially devoted for the visually impaired. These labels are mapped on the device to be exploited by touch.

– **Chapter 6** concludes this thesis by summarizing our proposed system and the obtained results. In this chapter, we also discuss the system strength and limitation, and we suggest some future works to improve it.

1.6 List of Publications

As part of this thesis, the following works have been published:

– **An article:** Zatout, C., Larabi, S. Semantic scene synthesis: application to assistive systems. The Visual Computer (2021). https://doi.org/10.1007/s00371-021-02147-w

– **A conference paper:** Chayma Zatout, Slimane Larabi, Ilyes Mendili, and Soedji Ablam Edoh Barnabé. Ego-semantic labeling of scene from depth image for visually impaired and blind people. In 2019 IEEE/CVF International Conference on Computer Vision Workshops, ICCV Workshops 2019, Seoul, Korea (South), October 27-28, 2019, pages 4376–4384. IEEE, 2019.

– **A conference paper:** Zatout, Chayma, and Slimane Larabi. A Novel Output Device for visually impaired and blind people's aid systems. 2020 1st International Conference on Communications, Control Systems and Signal Processing (IEEE CCSSP), March 16-17, El Oued, Algeria, 2020.
https://ieeexplore.ieee.org/abstract/document/9151820.

Chapter 2

Systems for Human-Scene Interaction

2.1 Introduction

We can classify computer systems into two categories: systems with Human-Scene interaction, or more generally systems with Human-Computer interaction, and systems which do not require Human-Computer interaction such as scripts. The systems with Human-Computer interaction are those that offer to users the ability to interact with computers and smartphones such as inserting information and gaming. In the same way, the systems with Human-Scene interaction are those that generate outputs after processing the captured scene in order to offer to the user the ability to interact with the scene such as understanding its content and navigating.

In this chapter, we study and analyze the related works, from capturing the scene to the Human-Scene interface. Indeed, this is important in order to determine their limitations and to design our own system. For this purpose, in Section 2.2, we review the state-of-the-art systems in different fields, namely: scene understanding and object detection, indoor navigation and obstacle avoidance, outdoor navigation, ground detection, semantic labeling, human computer interaction and auto-positioning. As Human-Scene interaction is more delicate in aid systems, most of the studied sys-

tems are of this type. Then, we study the different components of these systems in Section 2.3. We conclude this chapter by discussing the limitations of the presented systems in Section 2.4.

2.2 Human-Scene interaction: A review

The computer systems with Human-Scene interaction consist of a set of techniques whose goal is to enhance the user life in different activities such as route finder, indoor navigation, etc. Generally, these systems process the received data from the real world using sensors and transform it into instructions and signs. They use depth or RGB sensors and techniques of image processing, computer vision and machine learning.

2.2.1 Scene understanding and object detection

In general, the computer systems for scene understanding depend on object classification and detection: detecting objects is important in most applications like image captioning. In assistive systems for example, knowing the objects nature will provide for the users the ability to interact with the scene such as auto positioning and creating free space by moving some kinds of objects like chairs.

In [7], they proposed a wearable system for recognizing and locating some types of obstacles using a depth camera. After detecting the ground, they used a linear classifier to classify the point cloud features into: chairs, tables, stair up, stair down and walls. The obtained class is coded and mapped into a braille display; so by touching the braille device, the user will understand what type of objects is in front of him.
In [16], they proposed a system that uses computer vision techniques (such as Visual Odometry, Region-growing and Euclidean cluster extraction) and depth data to determine if the horizontal planes are a valid step of a staircase. After the segmentation

step, they transform the scene orientation into the Manhattan system [17]; so the extracted planes can be classified into vertical or horizontal planes.

In [6], they proposed a wearable system for scene understanding using deep learning techniques. The scene is captured using an RGB-D camera and the results are displayed using an earphone and a smartphone that serves as a haptic device. They adapted FuseNet [18] and GoogleLeNe [19] to provide semantic segmentation and orientation instructions respectively. The semantic segmentation model was designed to predict 40 different classes; however, these latter are transmitted using an audio device.

In [20], they proposed point cloud segmentation based on cascaded decision tree using RGB-D images. They showed good results; however, the system only classifies a given segment, whether it is the ground, a wall or a table (a horizontal plane that is not the ground). As in [21], they proposed a lightweight Convolutional Neural Network (CNN) architecture that is able to be executed on smartphones for traveling assistive system. They adopted PeleeNet for object detection to cover 80 different classes using the RGB images as input.

Some other works were devoted for specific types of indoors such as public indoors [22] and mesums [23]. In [22], they proposed an assistive system for object recognition in public indoors using RGB-D camera as an input device. Unlike the previous systems, they developed a comprehensive method based on machine learning techniques for object multi-labeling. In [23], they designed an exhibitor to communicate museum samples to the visually impaired and blind people. The system generates audio descriptions with adjusted volume according to the number of exhibits visitors. It also displays close-up photos of the exhibits to offer to the visitors with low vision to appreciate the most significant pieces' parts and their original colors.

Other researches were devoted to a special kind of classes instead of static objects and furniture: in [24], they proposed a system for face and text detection based on video analysis techniques. The system is able to detect faces using the binary

classification of sparse face representations. As for text detection, they opted for a cascade of SVM classifiers and filters.

2.2.2 Indoor navigation and obstacle avoidance

Many researches are devoted for indoor navigation [6, 25, 26]. Indoor navigation includes detecting obstacles, the floor (or free space), providing orientation instructions and/or computing the distance between the user and the obstacles. In navigation assistive systems in particular, the navigation is not only about delivering orientation instructions to move from a position/place to another, but also about ensuring the safety of the user during the navigation; in other terms, it's about avoiding obstacles.

Regarding obstacle detection, [27] applied a combination of elementary image processing operations. They, first, extracted edges using the Canny edge detector. Second, they connected the broken edges using the dilatation operation. With the assumption of the closed boundaries are objects and since some boundaries may remain unclosed, they performed erosion and dilatation operations. After that, they filled the closed boundaries with white pixels using the flood fill operation. Finally, the area of each object (each white area) is computed and then if this area is below a given threshold it will not be recognized as an obstacle.
In order to detect free space, in [28], they detected corners from RGB image using Harris and Stephens corner detector. Then, they divided the image into three regions of interest: center, left and right. After that, wall-like obstacles detection algorithm is applied to identify if the center region is safe. If this latter is not safe, they identify the free walkable space (left or right) and provide the user with the right orientation instructions. In [29], they proposed a multi-level recognition system for indoor positioning and navigation. The system provides location recognition, object recognition, semantic recognition and orientation instructions to navigate from a location to another.

When the input image is taken from an RGB camera of a smartphone, other sensors as the accelerometer and gyroscope can be used [30, 31]. In [31], they proposed a navigating system based on obstacle detection. The system combined the RGB camera, accelerometer, gyroscope, and the magnetometer to observe the user and the smartphone motion, extract the foreground and cluster it. In [32] and based on the processing of images from head-mounted RGB-D, the navigation is achieved using optical flow and tracking of certain points, or landmark identification that are indicative of a user's position along the crowded paths, or real-time 6-DOF ego-motion estimation using sparse visual features, dense point clouds, and the ground plane.

2.2.3 Outdoor navigation

In outdoor navigation computer systems, several approaches are used. They are generally based on the GPS and maps in order to get the user's and the target's positions and to find the best path regarding some fixed criterion such as the shortest path.

In [33], they proposed an algorithm for path planning that compute the optimal path from the user's current position to the target location regarding the number of turns. The system receives the graph corresponding to the supermarket layout , the system current position and the target position to provide orientation instructions. An additional module is added for obstacle detection to perform obstacle avoidance while navigating. In [34], they proposed an assistive system based on NavCog [35], an open-source turn-by-turn navigation assistant. To construct a map, they extracted the layout from OpenStreetMap[1] and populated it with Points Of Interest (POI) from Yelp[2] and FourSquare[3] POI retrieval geolocation services. The navigation in-

[1] `https://www.openstreetmap.org/`
[2] `https://www.yelp.com/`
[3] `https://foursquare.com/`

structions and the information details about POI are transmitted via an audio-device (smartphone).

Some other applications are based on image processing and computer vision techniques to detect potential obstacles in the outdoors. In [36], they proposed a system for obstacle avoidance in outdoor environments based on RGB camera. In order to detect the most common obstacles in outdoors, namely people, bicycles, cars, motorbikes, buses, traffic lights and traffic signs, they opted for YOLOv2 [37], a well known CNN network. To transmit the generated output, the system used a vibration-based device having different intensity levels.

2.2.4 Ground detection

Ground detection is generally the first step to be performed in many computer systems such as in navigation, scene understanding and augmented reality (like putting 3D models in free space) systems. For this purpose and based on depth images, many approaches have been proposed.

In [38], they use the fact that if a pixel is from the ground plane, its depth value must be on a rationally increasing curve placed in its vertical position. However, this solution causes problems if the floor has significant inclination or declination. RANdom Sample Consensus (RANSAC) algorithm is used in many approaches assuming that the space position of the floor is within z-max value [39], or distinguishing the floor from obstacles and walls based on hue, lighting and geometry image features [40] or using the ground's height determined by using the V-disparity [41]. Also, in [42], the RANSAC algorithm is used to find planes, and the relative distance and orientation of each plane with respect to the camera are then tested to determine whether it is the floor or not. *Wang et al.* [20] proceed differently, they distinguished free from occupied space by detecting planes. They proposed scene segmentation based on

cascaded decision tree using RGB-D image to classify segments, whether the plane is the ground, walls or tables.

Strong constraints are also used in [43] for floor detection assuming that ground plane must be large enough and Kinect is mounted on the human body, such that the distance between the ground plane and Kinect (y-axis coordinates) must be in a range of 0.8 to 1.2m.

2.2.5 Semantic labeling

Semantic labeling consists of assigning to each input a class label as an output. For instance, in semantic segmentation, it consists in labeling each pixel with a class. For Human-Scene interaction, providing a clear and simple to understand semantic labeling will save time and efforts. For example, labeling the set of pixels with the object's name is more effective than labeling them with different colors.

Horne et al. [4] proposed a semantic labeling for prosthetic vision for obstacle avoidance and object localization. They proposed a 2D pattern of phosphenes with different discrete level of intensity. In case of navigation, the phosphenes are activated to represent potential obstacles locations, thus free space was represented by gaps. So the user can have an impression about his surroundings and navigate without receiving audio instructions. Whereas for object localization (see Fig. 2.1), they selected a class (intensity level) to be used as landmark for maintaining orientation. To do this, they attributed for each pixel a discrete value that corresponds to its semantic class.

In [5], they proposed a navigation system based RGB image processing. The system transmits the generated scene and the navigation instructions on the Senseg TM device. They used the electrostatic signs to generate codes and form textural instructions for the visually impaired and blind people. They also used colors to

Figure 2.1: *Horne et al.* [4]: (left) the task of finding the garbage. (right) the produced semantic labeling: the garbage is represented with high-intensity phosphenes and the obstacles with low-intensity phosphenes. The user can follow the gaps to the target.

encode additional visual feedback for the sighted and with low vision people (Fig. 2.2).

In [7], they encoded the considered object classes using the first character of the class's name: o, c, t and a space to represent obstacles, chairs, tables and free spaces respectively. These codes are simple to understand, but it can be ambiguous while covering a large set of objects, especially with objects having the same first character.

2.2.6 Human Computer Interaction

Some assistive systems were devoted to enhancing the visually impaired experience in Human-Computer interactions. In [44], they proposed a system to analyze the structure of a web page based on visual information. The system uses a hierarchical segmentation of the web page after converting it to image. Using image processing and computer vision techniques, the system detects edges and segment the input page into semantic categories without the need to know the programming languages or the tags used to write these components.

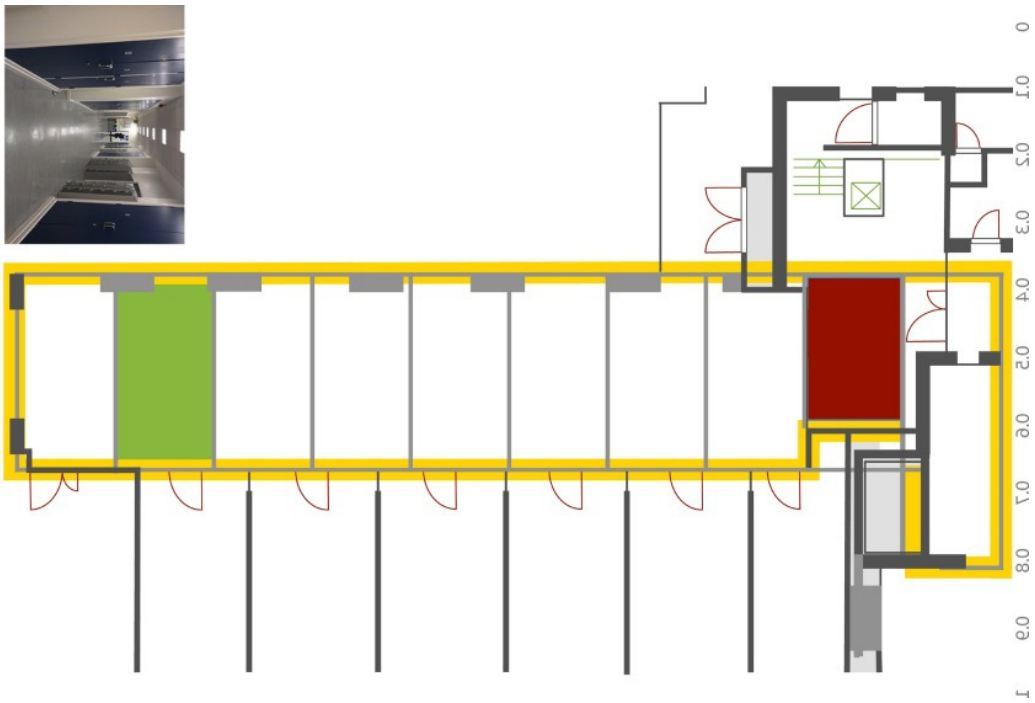

Figure 2.2: *Rubio et al.* [5]: the Senseg TM device screen. Walls are represented by the yellow outline, the user location is represented by the green box and the user's touch is represented by the red box.

2.2.7 Auto-positioning

Auto-positioning is another important task in everyday life. The auto-positioning is difficult to accomplish in unfamiliar environments, especially for the visually impaired. Several auto-positioning systems are proposed in the literature. In [45], they proposed an indoor positioning system to locate users based on artificial intelligence and computer vision techniques. They opted for the VGG16 and a region proposal network for feature extraction, object classification and bounding box extraction. The pose estimation were based on matching SIFT features after applying the RANSAC algorithm to remove outliers. Similarly, in [46], they proposed an auto-localization system based on deep neural network and computer vision techniques. The CNN features are used for localization; as for pose estimation, they computed the Hamming distance between the ORB features.

2.3 Human-Scene interaction: the systems' components

In general, for Human-Scene interaction, the computer systems consist of four main components: two devices (namely input and output devices) and two modules (namely the preprocessing and the processing modules). The input device, that most of the time is a sensor, delivers numerical data from the real world. In scene understanding, cameras are the most used sensors. The preprocessing module is the first step in this process due to the noisy nature of the manipulated data. The processing module is responsible to transform the input into an understandable output that can be transmitted via the output device.

2.3.1 Input device

Regarding the input device, we distinguish four types of aid systems: Depth-based systems, RGB-D based systems, Stereo-based systems and RGB-based systems. The depth sensors provide a depth map that supplies how far is an object (pixel set) from the user, if we suppose that the camera is carried by him. The depth camera can be structured light based such as Microsoft Kinect v1 [47] and Asus Xtion [48] or time of flight scanners, such as Microsoft Kinect v2 [47, 49] and ultrasonic [50].

An alternative to 3D sensors, is to use stereo cameras. In the stereo-based systems, the depth is obtained by finding corresponding matches between two images captured from two different cameras at the same time. With the absence of the described sensors, RGB cameras are used [27, 51, 52] as an input device in assistive systems. The type of the input device is important for the accuracy and the running time in real-time applications. Depth sensors are more accurate compared to stereo cameras

and RGB cameras. In terms of computational running time, depth sensors are suitable mainly when the processing phase is considerably slower.

2.3.2 Data preprocessing

Data preprocessing is a set of techniques including data smoothing, data down-sampling and data cleaning, that aims to reduce noise and redundancy to produce accurate data with reduction if necessary. It is applied when the data are noisy and huge regarding the algorithms in processing stage and thus it increases temporal and spatial complexity. Depending on data nature, several techniques can be applied. Regarding depth images, smoothing filters are used such as 2D-EEMD [53] filter and bilateral filter [54]. As for point clouds, passthrough filter and voxelization are widely used to reduce temporal and spatial complexity as in [16, 48] and many other works that are based on 3D data.

2.3.3 Data processing

The processing phase can be split up into four major categories: techniques based on image processing [27, 28], computer vision based techniques, artificial intelligence based techniques [55], and augmented reality based techniques.

Among the computer vision techniques, segmentation techniques such as RANSAC [51, 56, 57, 58] and region-growing [16]; computing descriptors such as SURF [59] [55], and Visual Odometry [16] are widely used. With regards to artificial intelligence approaches, cascaded decision tree [20], SVMs [59], multiple layer perceptron (MLP) [31] and deep learning techniques [6, 22] are used. Augmented reality based techniques on the other hand, are generally used for outdoor navigation [60] like navigation in unfamiliar places, and especially proposed to enhance the perception of people with a low vision as proposed in [61, 62].

2.3.4 Output device

In general, the generated output can be transmitted through audio-based interface [6, 21, 28, 34, 51, 55, 59, 61, 63], vibration-based interface [58] or a combination of them [52, 53]. The audio feedback can be stereo tone [63], beeps [63], recorded instructions [63] and text-to-speech [52, 53]. It can deliver some local information (like obstacle in front of you, etc.) or instructions to navigate (like go straight, etc.), to grasp objects (like up, down, left), etc.

Audio output is simple to understand and provides clear and concise information or instructions if these latter are well coded. However, it can be sometimes distracting and especially for the visually impaired. It also occupies ears and thus prevent them from doing their job like detecting unsafe instant events such "a car passing by". It can be seen when pedestrians cross the road while using their smartphones or listening to music.

The vibration-based devices such as vibration sensors, phones and vibrotactile gloves [57] is an alternative to audio feedback since it does not occupy the sense of hearing. However, the number of possible instructions provided by vibration can be modest (4 or 8 possible instructions are usually used). Military code as used in [52], enriches the possible directions to take but can be ambiguous like distinction between 5 and 10 o'clock.

The discussed types can sometimes be annoying, distracting and can not be used in some locations as using audio feedback in hospitals when silence is needed or supermarkets when there is huge noise. In these cases headphone [51, 61] or earphone [59, 63] can be used, but again, this holds hearing and isolates it relatively from the real world. With a simple informative semantic labeling for Human-Scene interaction, the navigation, the grasping and other tasks can be done without dictating instructions. Furthermore, such a labeling allow to have an impression about the real world and their surroundings, especially for the visually impaired and blind people

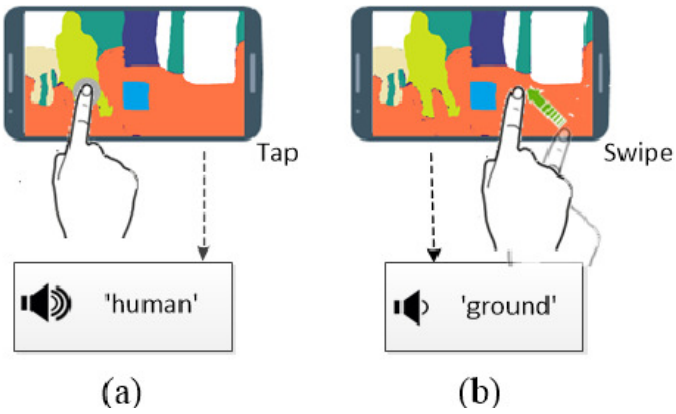

Figure 2.3: *Lin et al.* [6] for Human-Scene interaction: The smartphone screen.

. For example, coding the free space and the occupied space permits navigation, searching for an object, self positioning and other tasks.

Recently, tactile interfaces were proposed [6]. In [6], they proposed an output interface for Human-Scene interaction using smartphones. The semantic segmentation of the captured scene is mapped to the smartphone's screen. After touching a specific area (Figure 2.3), the proposed system transmits its label via wireless earphones with a volume that is proportional to the object distance, the closer the object, the higher the volume. This solution is based on touch sense; however, it still occupies hearing. A braille device is also used in [7] in combination with vibration sensors and an audio device (Figure 2.4). The braille device was used to transmit the occupancy grid to the user and to deliver the character that represents the class of the detected object in Braille system. They used the equivalent in the Braille system of: o, c, t and a space to represent obstacles, chairs, tables and a free spaces respectively.

2.4 Discussion

In addition to the discussed limitations of the Human-Scene interfaces based on audio and vibration. The state-of-the-art systems present many limitations:

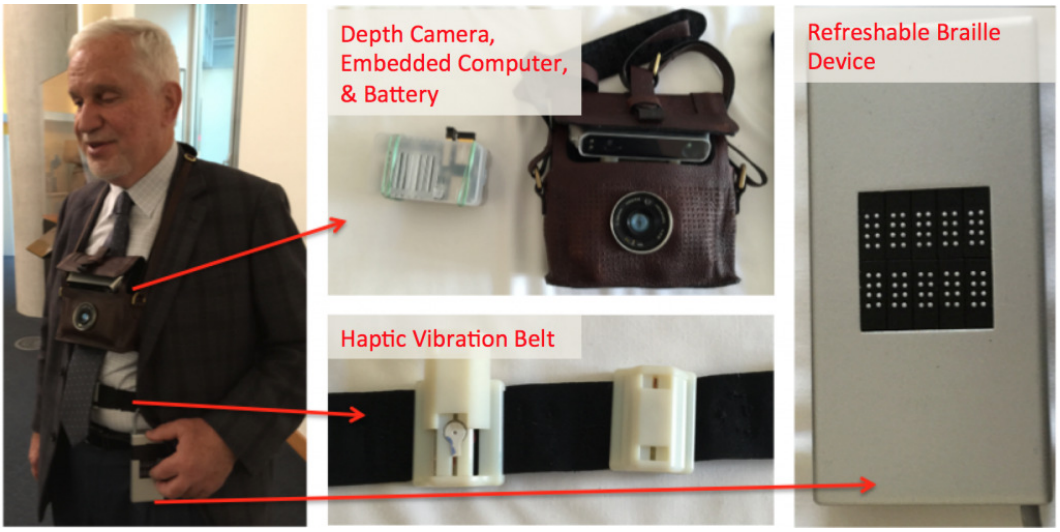

Figure 2.4: *Wang et al.* [7] for Human-Scene interaction: the hardware overview. The labels and the occupancy grid are mapped in the refreshable braille display. The haptic vibration belt produces pulse indicating the location of an obstacle.

- In indoor navigation, for example, they provide only a global description of the captured scenes (free space and obstacles without describing their nature). In some proposed systems when objects are detected, they usually use an RGB camera as an additional input sensor [64]. In other works, they provide obstacle classification, but by considering only a few classes such as classifying the scene components into the floor, objects that are parallel to the floor and objects that are perpendicular to floor without considering other features such as the object's height and its occupied area.
- The proposed semantic labels are less informative and can generate ambiguity and do not consider some geometric features that can be helpful [7].
- The actual aid systems are vibration-based or audio-based. For the visually impaired, the hearing replaces the sight in different tasks such as detecting some events, thus, the importance of releasing it.

2.5 Conclusion

In this chapter, we viewed and presented some of the state-of-the-art computer systems that are especially designed for Human-Scene interaction. These systems have a similar pipeline even if they have not similar input and similar output. This latter ensures generating the required output from the given input while reducing noise. We will use the same pipeline for our proposed system. However, unlike these systems, we will generate an output that is exploitable by the touch sense and thus we will overcome the discussed limitations. In other words, the system is designed based on a touch interface for a better Human-Scene interaction. In our system, we will be using point clouds as input. Therefore, in the next chapter, we will present different techniques for point cloud processing.

Chapter 3

Point Cloud Processing

3.1 Introduction

Nowadays, point clouds are widely used as inputs for many applications. They are a 3D representation that is suitable for dealing with real-world data, especially when the 3D scene geometry is required, such as the distance from the input sensor, the shape of objects and their size.

In this chapter, after defining the point clouds in Section 3.2, different types of 3D scanners will be introduced in Section 3.3. In Section 3.4, we present the most common filters employed to reduce the noisy nature of point clouds. Some descriptors will be also presented in Section 3.5; they are used to represent a point cloud differently depending on the application. The segmentation and the 3D shape recognition techniques will be introduced in Section 3.6 and Section 3.7 respectively. Finally, we will explain in more details deep learning techniques for point cloud classification in Section 3.8.

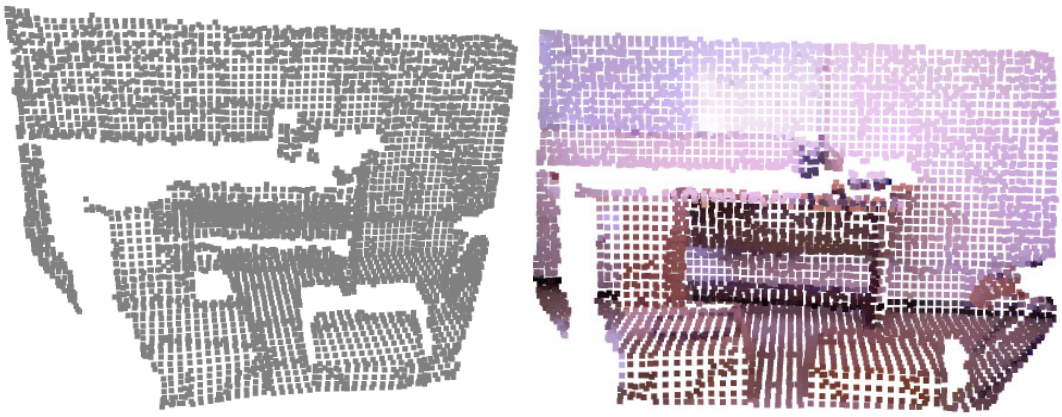

Figure 3.1: (Left) point cloud with only 3D coordinates. (Right) point cloud with 3D coordinates and RGB colors.

3.2 Definitions

A point cloud (Figure 3.1 (right)) is a collection of points that represent the captured scene in real-world or objects in space. It is a discrete representation of the geometry of scenes (objects). Point clouds are generally generated using 3D scanners or from Computer Aided Design (CAD) models. In a more formal way, a point cloud is a set of points *pcd* (eq. 3.1) where each point Pi is represented by its 3D coordinates $P(x, y, z)$, namely the $x - coordinate$, the $y - coordinate$ and the $z - coordinate$. The latter generally, represents the distance from the input sensor

$$PCD = \{P_i : i = 1, ..., n\}. \tag{3.1}$$

Note that some other features can be added to describe point clouds such as RGB colors, normals, etc. The RGB colors, for example, can be added in order to provide more information (See Figure 3.1 (right)). This is done after aligning the depth image and the RGB image.

Organized point clouds is a particular type that is provided by some scanners such as Microsoft Kinect V1 and Microsoft Kinect V2. The organized point clouds are point clouds that have an information about the points order: they are stored in a

row or a grid. Microsoft Kinect V1, for example, provides a depth map D (a matrix of depth information) that can be used to create an organized point cloud PCD(a matrix of points): each point P_{ij} at the row i and the column j is computed using the depth of D at the row i and the column j. The advantage of organized point cloud is the fact that some algorithms can be optimized and thus, the execution time can be reduced such as searching for the point neighbors. Indeed, in this case, instead of checking each point if it is a neighbor of a given point, the latter will be limited around the point's cell.

3.3 3D Scanners

3D scanners are a set of devices that use different technologies and techniques in order to convert the real-world scene into a numeric measurements named point clouds. 3D scanners can be devised into three main categories, namely: laser scanners, time of flight scanners and structured light scanners.

In this section, we will cover these types of scanners in Section 3.3.1, Section 3.3.2 and Section 3.3.3 respectively. Since we will be using Microsoft Kinect as the input device, Section 3.3.4 is devoted to review it in particular. We will also explain how to calibrate a depth camera for point clouds computation in Section 3.3.5. Finally, we will present the possibility to estimate point clouds from a single RGB image in Section 3.3.6.

3.3.1 Laser scanners

Laser scanners use a laser beam in order to compute point cloud. They project the laser beam on the target (scene/objects) and after that, derive the distance to the target based on the triangulation technique. This is done by adding a camera that captures the intersection between the laser beam and the target. The laser scanner,

the camera and the target form a triangle with its size defined by the distance between the camera and the laser scanner, the angle of the laser corner and the angle of the camera corner. This type of scanners is accurate, the distance that can be measured is up to certain kilometers, but unfortunately they are highly expensive.

Laser scanners are also used in Light Detection and Ranging (LiDAR) technique which is a time of flight based approach. In LiDAR based solutions, the distance between the target and the laser scanner is represented by the time elapsed between the emission and the reception of the laser beam. The LiDAR technique is usually used for large and distant targets such as mountains and buildings.

3.3.2 Time of flight scanners

As for time of flight scanners, such as Microsoft Kinect v2 and ultrasonic, to compute the depth data, they project light rays and then compute the time elapsed between the projection and the reception of these later. These scanners are less expensive, but also less accurate and noisy compared to laser scanners. The difference between time of flight scanners and the LiDAR technique is that LiDAR uses only a single laser beam.

Ultrasonic is a time of flight low-cost sensor, that computes the depth using the time elapsed between the emission and reception of the ultrasonic waves [65]. It is used not only for indoor navigation [63] but also for outdoor navigation in contrary to Microsoft Kinect v2 that is more suitable in indoors.

3.3.3 Structured light scanners

Structured light scanners, such as Microsoft Kinect v1 and Asus Xtion, are less expensive, lightweight but provide noisy data compared to the previous one. In

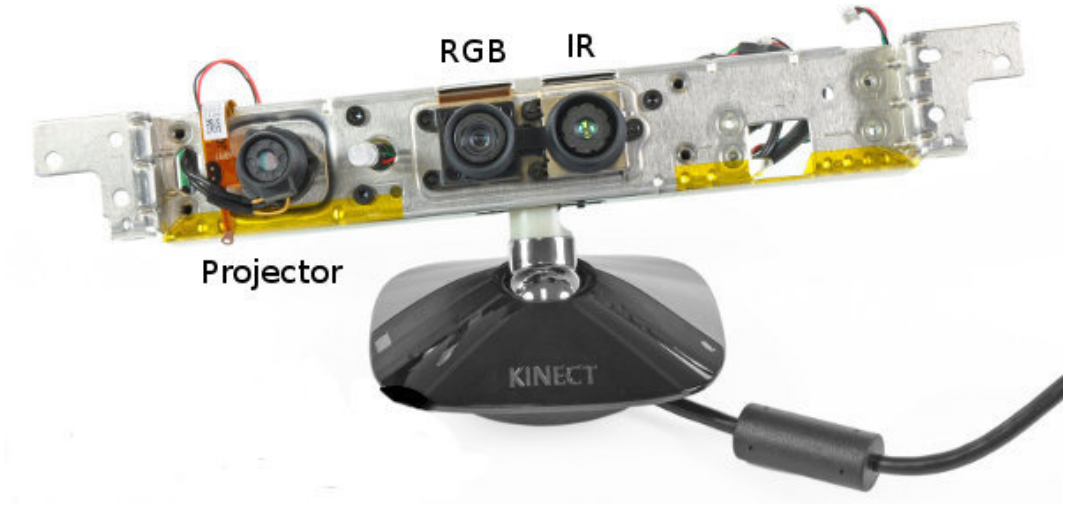

Figure 3.2: Microsoft Kinect (V1) [8].

order to calculate the depth data, they compare the distortion between the projected pattern light and the received one.

3.3.4 Microsoft Kinect V1

Microsoft Kinect (V1) (Figure 3.2) is a low-cost RGB-D sensor that provides, simultaneously, the RGB image and the depth image with (640x480) as resolution and streams output video with 30 frames per second. Kinect is popular in the research field, is involved in many applications such as indoor navigation, obstacle avoidance, etc. It consists of an RGB camera, Infrared (IR) camera and IR projector. The IR camera captures the pattern projected by IR projector (see Figure 3.3 (middle)) to generate the depth image (see Figure 3.3 (right)).

Each pixel of the depth image denotes its distance from the camera. The depth data provided is noisy, in particularly beyond a particular range fixed according to the sensor's state. Furthermore, with the nature of IR radiation, Kinect is sensible toward transparent objects and strong sources of light. To prevent these problems, other sensors, such as ultrasonic sensor, can be used alongside with the Microsoft Kinect as in [63].

Figure 3.3: Microsoft Kinect's produced data. From left: RGB image, IR image and Depth image.

3.3.5 Depth camera Calibration

To compute the point cloud from the depth image, calibrating the depth camera is required. In general, there are three methods for calibrating a camera : using the standard parameters provided by the factory, using the results obtained in calibration research or calibrating the Kinect manually. The first two methods are simple but less accurate since the factory parameters can be changed by time and can be slightly different from a camera to another. The third method is more accurate, but it requires using one of the calibration algorithm such as the chess-board algorithm proposed by Zhang and Huang in [66]. This algorithm is implemented in Robot Operating System (ROS) and OpenCV. The principle of the chess-board algorithm is to compute the position of each corner of the black square (called pattern) and the compute the difference between the obtained results and the positions in the real-world. This later is finally used to extract and infer the distortion vector and the camera matrix to correct the x and y values and convert them to the camera system. This process is explained in [67].

In RGB camera calibrating, this is done by extracting the corners from the RGB image; however, this is not the case of the depth camera since all the corners in the chess-board have the same depth and thus it will look like a plane: the corners cannot be seen. Thus, instead of using the depth image shows the distorted RGB image input (Figure 3.4 (left)). The algorithm detects the chess-board to extract

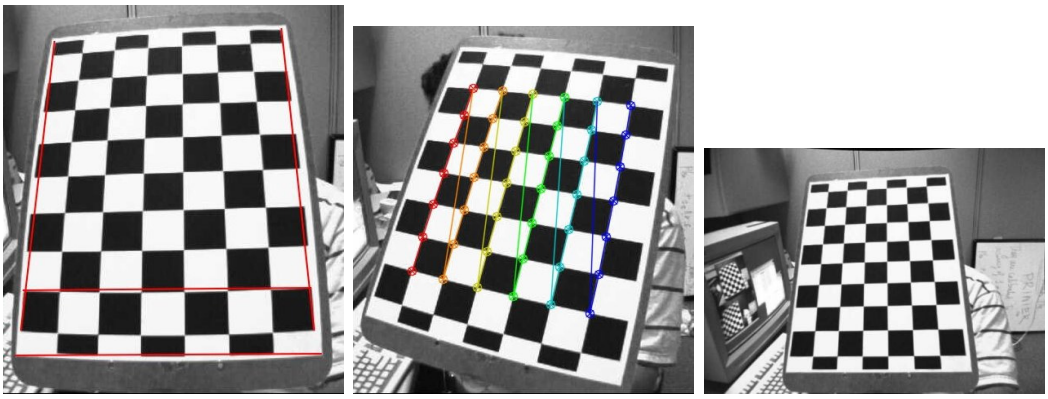

Figure 3.4: Camera calibration [9]. From left to right: an image with distortion, the chess-board pattern used to compute the distortion parameters, and the image results after calibration (the distortion is eliminated: the edges are straight).

the pattern (Figure 3.4 (middle)). After that the algorithm compares the size of the extracted pattern with its size in real-world to compute the distortion vector. This vector is, after that, used to remove distortion from images captured with the same camera (Figure 3.4 (right)).

Once the distortion coefficients and the camera matrix are obtained using one of the methods mentioned above, the x and y are corrected and the 3D coordinates are then computed. Let D be the distortion vector, depending on the distortion nature (if it is radial or tangential distortion), the corrected values of x and y are computed respectively using one of the following formulas [9]:
in case of radial distortion:

$$x' = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6), \tag{3.2}$$

$$y' = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6), \tag{3.3}$$

in case of tangential distortion:

$$x' = x + [2p_1 xy + p_2(r^2 + 2x^2)], \tag{3.4}$$

$$y' = y + [p_1(r^2 + 2y^2) + 2p_2 xy], \tag{3.5}$$

knowing that D the vector of the distortion coefficients is given as:

$$D = (k_1, k_2, p_1, p_2, k_3). \tag{3.6}$$

Note that the same formulas can be applied to correct x and y of the RGB camera, the only difference is to compute and use the distortion coefficients of the RGB camera.

Once the x and y values are corrected, the 3D coordinates (X, Y and Z) are computed. Let M be the estimated calibration matrix (or the camera matrix) of the depth camera in the calibration process defined as:

$$M = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$$

Wehre (fx, fy) and (cx, cy) are the focal lenghth and the optical centers respectively. Finally, the 3D coordinates are computed using the following formulas [67], where $depth(x', y')$ is the depth value at the raw y' and column x':

$$X = (x' - cx) * depth(y', x')/fx, \tag{3.7}$$

$$Y = (y' - cy) * depth(y', x')/fy, \tag{3.8}$$

$$Z = depth(x', y'). \tag{3.9}$$

Figure 3.5 shows the generated point cloud from the depth image using the previous equations (equation 3.7, equation 3.8 and equation 3.9).

3.3.6 Deep learning for point clouds estimation

Many works have investigated the use of deep learning for point cloud reconstruction from RGB images. These works can be divided into two categories: multi-view based depth estimation and monocular based depth estimation. In the first category, the

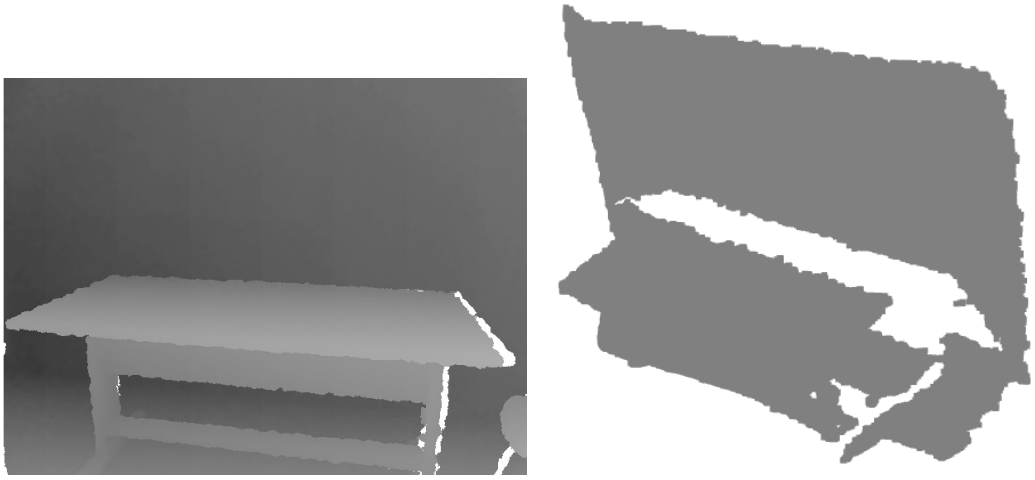

Figure 3.5: Left: the depth image. Right: the computed point cloud.

depth map is produced from a set of posed RGB images [68, 69]. The point cloud is then generated using the intrinsic parameters of the camera as explained in Section 3.3.5.

As for the second category, the depth image is generated using a single RGB image. They are generally, based on CNN networks such as fully convolutional residual network [70], attention guided network [71], deep ordinal regression network [72]. The point clouds generated from the works [70, 71, 72] are distorted due to the quality and the resolution of the estimated depth maps. DenseDepth [73] is a recent state-of-the-art network that is a straightforward convolutional encoder-decoder network. It generates depth images more accurate and with higher resolution; however, the generated point cloud still distorted (see Figure 3.6).

3.4 Point cloud filters

The computed or the gathered point clouds can sometimes be noisy due to the nature of the used 3D scanners or the captured scene. On the other hand, some algorithms or computer vision techniques are sensitive to noise: the output can be affected

Figure 3.6: Point cloud estimation using DenseDepth neural network. First row: the input image (left) and the generated depth map (right). Second row: two views of the generated point cloud. The depth image is over smooth; thus, the point cloud is distorted.

by noise and thus the system accuracy is decreased. In order to reduce noise, some filters are used. These filters are also used to reduce the point cloud density and thus reduce the computation time. In this section, the most common filters, namely: pass-through filter, voxelization, statistical outlier removal filter, radius outlier removal filter and parametric models for filtering, will be introduced. Other filters can be found in this review [74] and a comparative study can be found in [75].

3.4.1 Pass-through filter

The Pass-through filter applies constraints on the input point cloud along a particular axis. Each point passes through a defined constraints: it is removed if it is a non-finite point or if it is outside the defined interval along the specified axis. This interval is fixed according to the nature and the state of the input device. Like Microsoft Kinect V1 for example, the interval is set to $[800, 4000]mm$ along the $Z - axis$ by many researchers: the depth data is more accurate inside this interval and become more noisy otherwise. The pass-through filter can be used not only for filtering the input from noise, but also to reduce data such as considering the nearest points (see Figure 3.7).

3.4.2 Voxelization

This voxel grid filter (Figure 3.8) transforms the point cloud into a 3D voxel grid. The voxel grid divides the input into a set of l^3 voxels, where k is the voxel size. Each voxel includes points that belong to the same intervals regarding the 3 axes. The points belonging to the same voxel are then down-sampled and replaced with their centroid. This filter is used to reduce the point cloud's size and to smooth it. However, it is time consuming since it computes the centroids after reorganizing the point cloud into voxel, and it is sensitive to outliers.

Figure 3.7: Pass-trough filter. The red points will be neglected since they are out of the fixed interval $[1500, 2000]mm$.

Figure 3.8: Left: the point cloud. Right: the point cloud after voxelization (in this example, we took $l = 5mm$). The point cloud is smoothed and its size has been reduced.

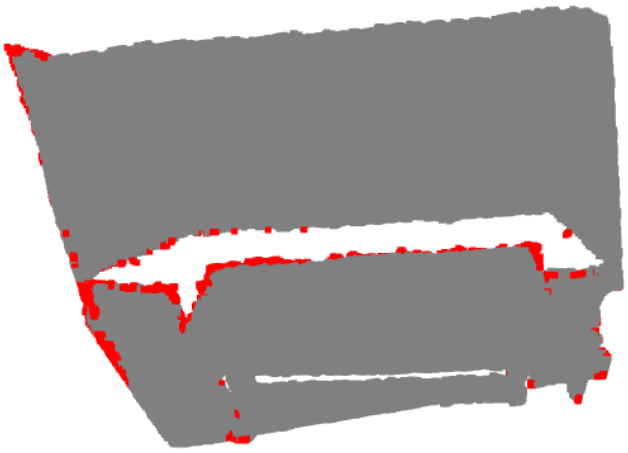

Figure 3.9: Statistical outlier removal filter: the red points are outliers (in this example, we took $std = 0.03$ and $k = 500$).

3.4.3 Outlier removal filter

Statistical outlier removal filter (Figure 3.9) is based on a statistical analysis on each point and removing it if it does not meet a certain criterion. It provides a Gaussian distribution of the point cloud with a given standard deviation, *std*, by computing the mean distance from a given point to all its k neighbors and removing it if its mean distance is outside an interval defined by the global distances mean. This process is applied for every point of the point cloud.

Radius outlier removal filter (Figure 3.10) is a conditional filter that removes every point that has less than a certain number, k, of neighbors within a certain range, r.

3.4.4 Parametric models for filtering

Parametric models such as the RANSAC algorithm or even simpler plane fitting can be used as a filter. They fit the input point cloud into a defined 3D model (plane, cylinder, etc.). The points that don't belong to the defined model are considered as

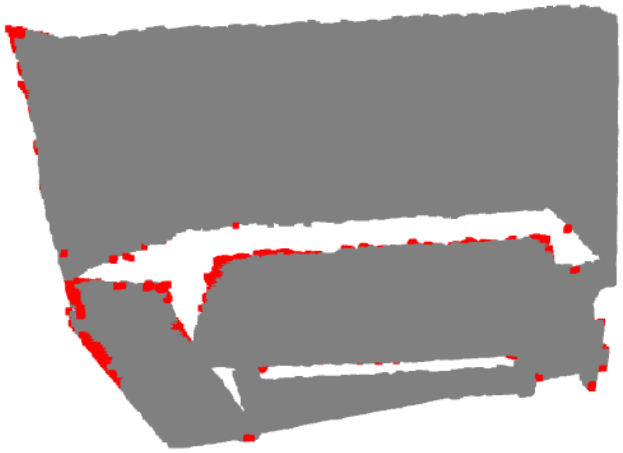

Figure 3.10: Radius outlier removal filter: the red points are outliers (in this example, we took $k = 300$ and $r = 10mm$.

noise. Parametric models can also be used to smooth the point cloud by projecting the points into a given 3D model.

3.5 Point cloud descriptors/features

In some applications, using the basic point cloud representation can lead to ambiguity and misleading results such as in comparing between two captured frames. Due to the nature of some 3D scanners, the same static scene captured at time t_1 and t_2 ($t_1 \approx t_2$) using the same scanner with respects to the same origin, a same point p from real-world can have two different computed values v_1 and v_2 from t_1 and t_2 respectively. In contrast, in a dynamic scene two different points p_1 and p_2 in the real-world can have the same value v at t_1 and t_2 respectively: p_2 took the place of p_1 at time t_2. As a consequence, in this type of applications, it's better to find other features and descriptors for a better representation to distinguish between geometric surfaces.

Point cloud descriptors/features are widely used in many fields, especially when comparing between different point clouds or searching for the point cloud keypo-

ints as in point cloud registration, visual positioning and scene classification. Point cloud descriptors can be divided into three main classes: local-based descriptors, global-based descriptors and hybrid descriptors. The selection of a good descriptor is dependent on the performed task. In addition, a good descriptor should remain efficient and invariant in the presence of rigid transformations (3D rotations and 3D translations), density variation and noise. It should be highly descriptive with an efficient computation. In this section, we will bravely introduce some of local and global descriptors in Section 3.5.1 and Section 3.5.2 respectively. We can find more descriptors in these surveys [76, 77, 78, 79].

3.5.1 Local descriptors

Local descriptors techniques estimate a set of feature vectors, generally based on geometric information such as normals and curvatures. Each vector, is estimated based on the geometry of the local neighborhood of a given keypoint. They describe the characteristics and the geometry of their neighborhood. In other words, each local feature vector describes the local neighborhood and their combination describes the entire point cloud geometry. This later makes them robust to occlusion and clutter. Spin Image [80], Point Feature Histograms (PFH) descriptors [81] and Fast Point Feature Histograms (FPFH) descriptors [82] are among the most common local descriptors.

Spin Image [80] consists of computing for each neighborhood of the keypoint that is represented by its coordinates and its normal, two distances: the distance to the key-point's plan and the distance to the line holding the keypoint's normal. Then the descriptor (the spin image) is generated by accumulating the computed distances of keypoints in discrete 2D bins. This descriptor has the property of the local descriptors (robustness to occlusion and clutter); but it is not robust to local noise and outliers.

Unlike Spin Image, PFH consists of capturing the interactions between the points in k neighborhood and their surface normals. It generates a multi-dimensional histogram of values by generalizing the mean curvature. PFH is robust to noise and invariant to the 6D pose; however, it is time consuming and not suitable for real time applications. Therefore, FPFH was proposed to reduce the temporal complexity from $O(nk^2)$ to $O(nk)$ by computing the weighted simplified PFH. These descriptors are suitable for 3D registration.

3.5.2 Global descriptors

Global descriptors compute a single feature vector for the entire input point cloud. They rely on the observation of the entire geometry and fail in environments with occlusion and clutter. Global descriptors are suitable for 3D object recognition, geometric categorization and shape retrieval applications. Several global descriptors have been proposed over time: Point Pair Feature (PPF) [83], Global Radius-based Surface Descriptor (RSD) [84], Viewpoint Feature Histogram (VFH) [85], etc. Recently, deep features are widely used as global descriptors. Deep features are represented by the output or the input of dense layers of deep neural networks like the classification models such as PointNet [13].

3.6 Point cloud segmentation

Point cloud segmentation consists of a set of techniques and algorithms that aim to regroup raw 3D data into non overlapping regions. These regions may correspond to a specific object structure or a specific structure. In point cloud segmentation, contrary to semantic segmentation, the extracted segments has no semantic information since the technique does not require prior knowledge. In this section, we categorized point cloud segmentation methods into 3 main classes: Region growing based methods

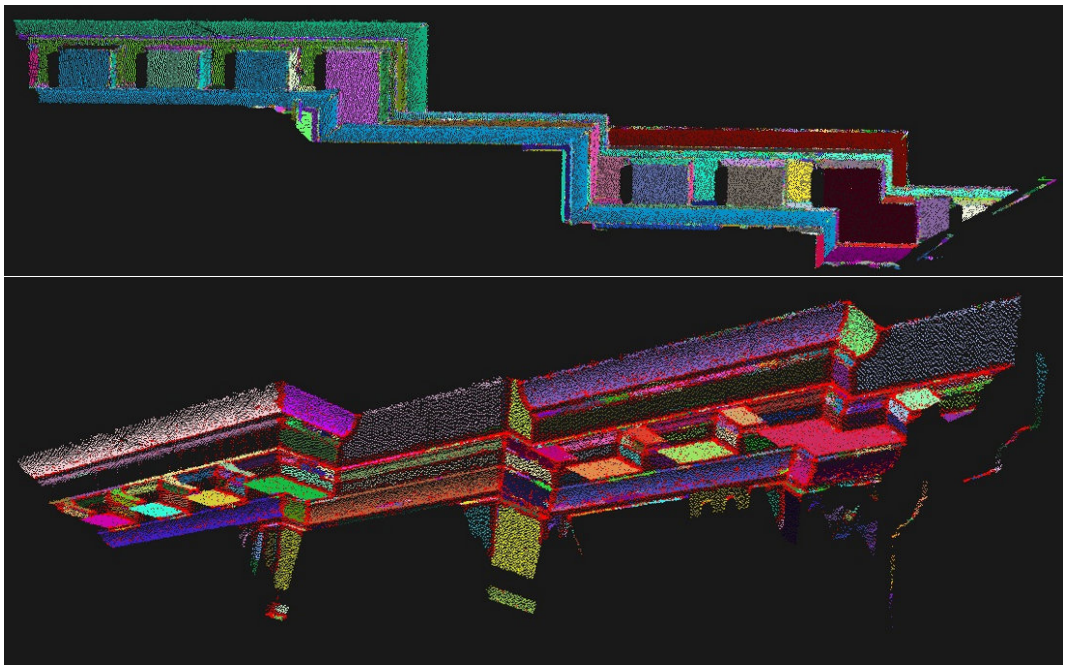

Figure 3.11: Result of point cloud segmentation using region growing algorithm [10] for two different scenes: each color represents a single segment.

(that will be presented in Section 3.6.1), model fitting based methods (that will be presented in Section 3.6.2) and clustering based methods (that will be presented in Section 3.6.3). More details and more classes can be found in [86, 87].

3.6.1 Region growing based

Region growing methods were first introduced for 2D images (rgb and intensity images) [1]. Then, it was applied on range/depth images in [88] and soon adapted for point clouds (Figure 3.11). After that, several variations were presented in the literature and still widely used today [89].

Before describing how they works, 4 factors should be presented: seed unit, seed unit selection, criteria and growth unit.

- Seed unit: represents the initial entity. It can be pixels in 2D, points or voxels in 3D.

- Seed unit selection: represents the algorithms used to select seed units like selecting points having the minimum curvature as seeds.
- Criteria: represents the similarity measures. Generally, in point cloud region growing algorithm, they are based on the geometric features such as: normal vectors or Euclidean distance.
- Growth unit: the amount of units to grow the regions with. It can be a single pixel/point, a voxel or other other structures like octrees.

Region growing based methods consist of two main steps. The first step represents the identification of the initial units, called seed units, based predefined metrics. In the first adaptation, they used the points that has the minimum curvature values (representing a flat area) as the seeds. Then, growing the initial seeds with the defined growth unit based on the predefined criteria. The algorithm 1 summarises the original algorithm called seeded region growing Algorithm [1]. Region growing methods can be categorized into 2 classes: the bottom-up methods that start from a set of seeds and grow them into segments with respect of the defined criteria; and top-down methods that start by defining the entire point cloud as a single region and then subdivide it to region.

Region growing based methods are robust to noise compared with traditional segmentation algorithms such as edge-based algorithms; however, they are computationally intensive and their accuracy depends on the location of the selected seeds and the estimation of the criteria like the accuracy of the estimated normals.

3.6.2 Model fitting based segmentation

Model fitting methods are based on matching the point cloud into primitive geometric shapes like planes, spheres and cylinders. In general, model fitting algorithms classify the input points that fit the mathematical representation of the primitive shape.

Algorithm 1 Seeded Region Growing Algorithm [1]

Input: pcd: $\{p_1, p_2, ..., p_m\}$ the point cloud to segment
n the number of seeds
$\{C_1, C_2, ..., C_n\}$ the clusters initialized by initial seeds s_k
D the distance threshold
Output: $\{C_1, C_2, ..., C_n\}$ the set of clusters

```
while pcd is not empty do
  for each cluster C_i do
    for each seed s_k of the cluster C_i do
      Select the neighboring points of s_k;
      for each each neighbor n_j do
        Compute d the Euclidean distance between n_j and s_k;
        if d < D then
          Add n_j to C_i;
        end if
      end for
    end for
  end for
  Update the seeds by the clusters' boundary points
end while
return {C_1, C_2, ..., C_n}
```

These algorithms can be used to segment point clouds into segments that fit the defined primitive shapes like segmenting the captured scene into planes. Model fitting methods are based on 2 classical algorithms: Hough Transform (HT) and RANSAC. Comparing these two methods, the RANSAC algorithm has shown more efficiency in terms of segmentation quality and running time [90]. In this section, we focus on RANSAC.

RANSAC was first proposed by Fischler and Bolles [91] for 2D detection. It was then adapted to detect 3D primitive geometric shapes on point clouds in [92] such as planes (Figure 3.12 (left)) and cylinders (Figure 3.12 (right)). After that, more RANSAC variations and enhancements for point cloud segmentation has been proposed [93]. The original version of the algorithm consists of 2 main steps. First, it generates a hypothesis (parametric model) by randomly selecting n points and then estimating the model parameters. Then, the estimated model is evaluated by testing it against

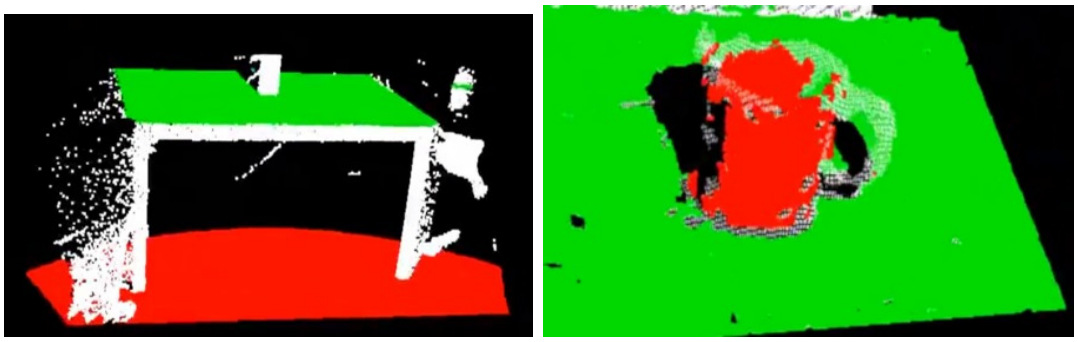

Figure 3.12: RANSAC for segmentation [10]. Green color and red color respectively represent the detected plane and cylinder.

all the point cloud: each point that fit the model is considered as part of the model (inlier). If the model fits well the data (it has more than m inliers), its parameters are saved. These steps are executed iteratively to estimate the best parameters.

The explained steps represent the original algorithm that is used for filtering. An additional criterion can be added to segment the point cloud into a set of 3D primitive geometric shapes. The criterion can be set to: the number of potential shapes, the number of the remaining points to consider as outliers, etc. The algorithm 2 summaries RANSAC for point cloud segmentation.

RANSAC algorithm is efficient and robust to outliers. It is commonly used for plane detection. However, it is a nondeterministic algorithm: it exhibits different behaviors on different runs for the same input.

3.6.3 Clustering based segmentation

In order to segment point clouds into coarse segments, clustering algorithms can be used for irregular object segmentation. K-means [94, 95] (see Figure 3.13 (top left)), mean-shift [96] (see Figure 3.13 (top right)), fuzzy-clustering [97] and Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [98, 99, 100] (see Figure 3.13 (bottom)) clustering algorithms are widely used for point cloud segmentation.

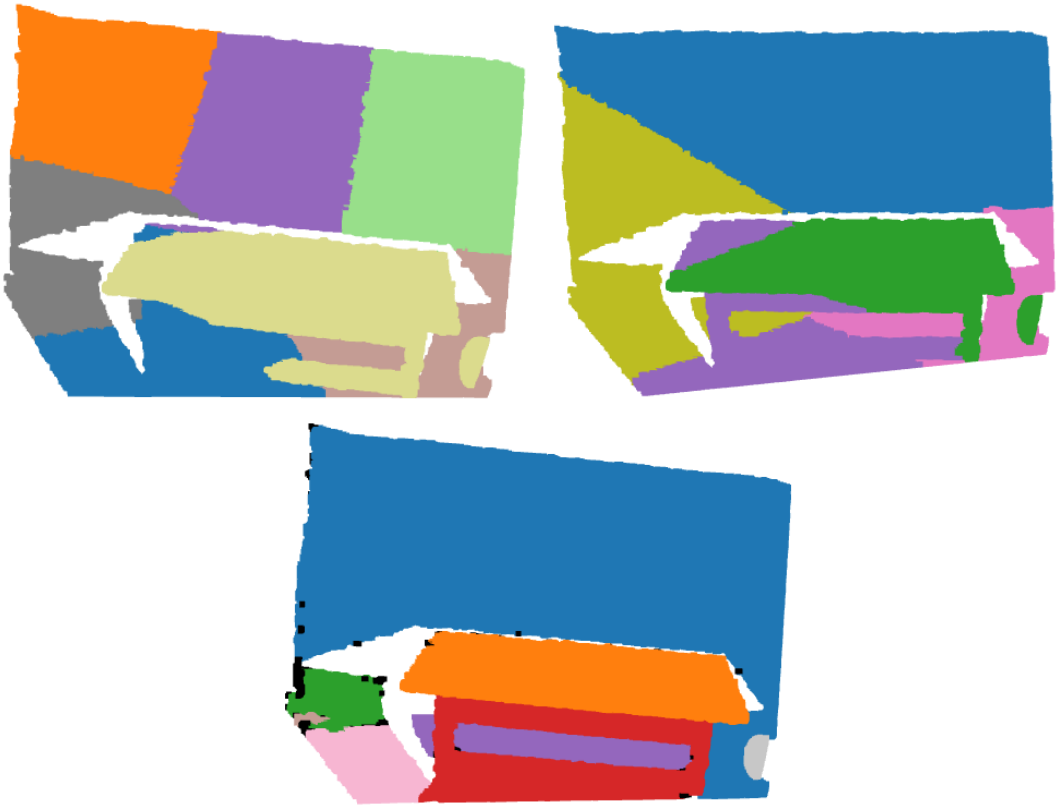

Figure 3.13: Point cloud clustering: k-means (top left), mean-shift (top right) and DBSCAN clustering (bottom). We first applied DBSCAN algorithm; it returned K clusters ($K = 7$) that is used to set the number of the clusters for k-means and mean-shift algorithms.

Algorithm 2 RANdom SAmple Consensus (RANSAC) Algorithm

Input: pcd: $\{p_1, p_2, ..., p_m\}$ the point cloud to segment
criterion : used to stop the segmentation process k the minimum number of points required to estimate model parameters // for planes detection $k >= 3$
n the maximum number of iterations to find the best model
D the distance threshold
Output: $\mathbb{P}$ the set of planes

while criterion is not satisfied **do**
  $BEST_{model} \leftarrow \emptyset$
  while $iterations < n$ **do**
    Pick random k points;
    Estimate the model parameters;
    $\mathbb{I} \leftarrow \emptyset$;
    for each point p in pcd **do**
      Compute the distance d between p and the estimated model;
      if $d < D$ **then**
        $\mathbb{I} \leftarrow p$;
      end if
    end for
    if The size of $\mathbb{I}$ > the size of $BEST_{model}$ **then**
      $BEST_{model} \leftarrow$ the fitted model;
    end if
  end while
  Add $BEST_{model}$ to $\mathbb{P}$;
end while
return $\mathbb{P}$ the set of planes

K-means, mean-shift and fuzzy-clustering are centroid based algorithms: the first divides the input samples into K separate groups with equal variance while minimizing the distances between points. The mean-shift algorithm separates the input samples into blobs with smooth density. As for the fuzzy-clustering, which is based on fuzzy-logic, it assigns for each sample a probability corresponding to each cluster. More the probability is high the more the sample is near to the cluster. One of the disadvantages of these algorithms is that they require the number of clusters as input which is unknown when dealing with real-world data.

On the other hand, the DBSCAN algorithm seeks to separate the samples into high density clusters with low density areas. In addition, the number of clusters does not

have to be defined beforehand. Furthermore, the DBSCAN is robust to noise: during the clustering, the outliers are detected and neglected and thus, the clustering process is not affected by them. It was first proposed by *Ester et al.* in [101]. It requires only two parameters: $minPts$ and eps that represent the minimum number of points to form a cluster and the distance measure that is used to locate the neighbors respectively. The algorithm consists mainly of two steps: it starts by randomly selecting a point to compute its neighbors. Then, these latter are considered as a cluster if there is at least $minPts$ points. The last process is repeated recursively for is neighboring point. The algorithm 3 summarizes the DBSCAN algorithm for point cloud segmentation.

3.7 3D shape recognition

In general, the classification task consists of assigning a class as an output to the given input. It is generally based on machine learning techniques with supervised learning that showed considerable improvement in many fields, especially with RGB data after introducing the CNNs.

At the early stage, researchers investigated the combination of different descriptors or/and handcrafted features with machine learning techniques [102, 103, 104, 105]. In [102], they proposed 3D shape contexts and harmonic shape contexts, two local descriptors for recognizing vehicles in noisy and cluttered scene. In [103], they designed an object classification system based on computing shape features that are orientation-invariant such as the volume, the average height and a spin image descriptor; and contextual features such as the position of an object relative to its environment. The extracted features are fed to a Support Vector Machine (SVM) model to classify them into classes of urban environments. In [105], they investigated the use of histogram based descriptors for 3D laser range data classification in Urban Environments. It was shown that, histograms based on the point distribution, nor-

Algorithm 3 DBSCAN Algorithm

Input: pcd: $\{p_1, p_2, ..., p_m\}$ the point cloud to segment
$minPts$ the minimum number of points in a cluster
eps the distance used to locate the points in the neighborhood
Output: $\mathbb{C}$ the set of clusters

while pcd is not empty **do**
  Select randomly a point a point p_i;
  Compute n_i the neighbors of p_i within the distance eps;
  if the size of $n_i < minPts$ **then**
    p_i is a set of outliers;
  else
    $C \leftarrow n_i$;
    for each point s_j in n_i **do**
      Remove s_j from n_i;
      Compute m_j the neighbors of s_j within the distance eps;
      if the size of $m_j < minPts$ **then**
        m_j is a set of outliers;
      else
        $C \leftarrow C \cup m_j$;
        $n_i \leftarrow n_i \cup m_j$;
      end if
    end for
    $C \leftarrow C \cup p_i$;
    $\mathbb{C} \leftarrow \mathbb{C} \cup C$;
  end if
end while
return $\mathbb{C}$

mal orientations, or spectral values are more suitable in this type of classification. These systems suffer from the same limitations of the used descriptors. For example, the volume and the average height are features that are orientation-invariant, but they are not scale-invariant and they are not robust to noise.

With the introduction of RGB-D and 2.5D scanners, other researchers used the depth image as an additional channel along with RGB channels [106, 107, 108, 109]. These methods are straightforward; however, they do not really exploit the geometric properties. Current works investigate the 3D object classification using multi-view representation (example: Figure 3.14 (middle)), volumetric representation (example:

Figure 3.14: Stanford bunny. From left to right: a single 2D view representation, volumetric representation and point cloud raw representation.

Figure 3.14 (left)) or point cloud raw representation (example: Figure 3.14 (right)). In this section, we will present these different models. Figure 3.14 shows Stanford bunny [1], one of models most commonly used in testing and tutoring, in 2D, volumetric and point cloud representations.

3.7.1 Multi-view based classification

Multi-view based recognition models are based on 2D representations: 2D projection images [110] or depth images [14]. These latter are estimated while performing rotations most of the time around the y-axis. In [110], they designed a multi-view CNN architecture for 3D shape recognition. The model combined multiple 2D views of the input 3D shape. Each view is then fed to an unified CNN network to extract view based features. These latter are then pooled across views and passed through another CNN network to obtain a compact shape descriptor.

The multi-view based architectures preserve fined details unlike voxel based models, for example, that require resolution reduction to reduce temporal and spatial complexity especially when using 3D convolutions. However, they fail when data is incomplete or/and occluded especially in real-world where rotating the captured object is non-informative.

[1] `graphics.stanford.edu/data/3Dscanrep/`

3.7.2 Voxel based classification

Voxel based recognition models are based on volumetric representations. In [14], each 3D shape is represented by 30^3 grid of probability distribution of binary variables where each voxel indicates whether it is inside the shape (assign it the value 1) or not (assign it the value 0). To process the computed grid, they designed Convolutional Deep Belief Network (CDBN), an adaptation of Deep Belief Networks (DBN) [111], a probabilistic models to join probabilistic distribution over pixels, with convolutions. In [112], instead of using a naive volumetric representation as in [14]. They integrated a 32^3 volumetric occupancy grid that takes in consideration not only the free and the occupied space but also the unknown space. By using a straightforward 3D CNN model, this work outperforms the previous work since CNN models have the capacity to learn spatial features.

The presented models were initially trained and tested on 3D CAD models. However, when dealing with real-world data i.e. 3D or 2.5D scanners, these models adapted the multi-view approach to estimate the volumetric representation in different ways. In [14], computed the volumetric representation of the predicted best-next-view from the initial view. Whereas in [112], the model votes over the predicted classes of the volumetric representations computed from the 3D multi-views.

3.7.3 Point cloud based classification

The previous approaches require highly regular data. However, some scanners, such as LIDAR, provide point clouds in their raw representation (a set of 3D points). In this case, researchers transform the point cloud into a voxel grid which reduces the input resolution. In addition, they are time and space consuming due to 3D convolutions.

Point cloud based methods process the point clouds in their raw representation.

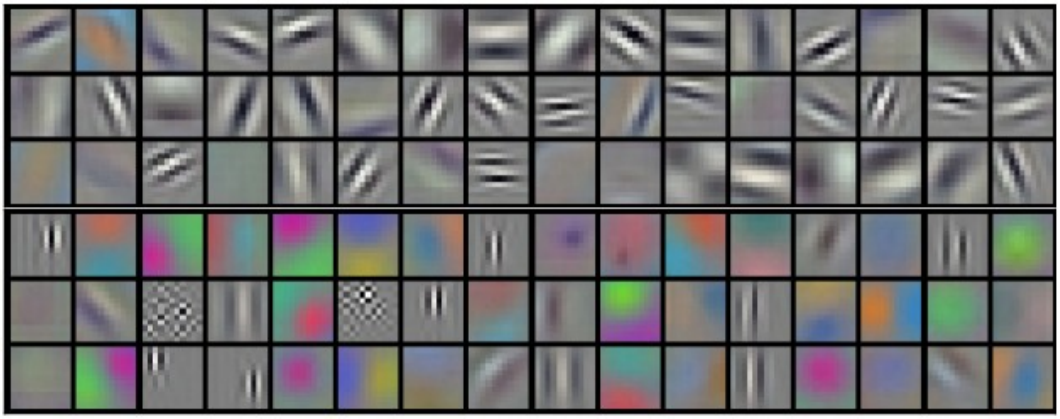

Figure 3.15: The representations learned by AlexNet [11] of large-scale image classification.

These latter exploit the properties of point clouds: they are unordered, invariant under transformations and the points are not isolated. Point cloud based classification models are generally based on deep neural networks. *Qi et al.* proposed PointNet network [13], a pioneer deep neural network for point cloud classification. It achieved a great success, it surpassed the state-of-the-art models with an accuracy equal to 89.5. After its success several models were proposed like PointNet++ [113] which is the enhanced version of PointNet and others [114, 115]. We will present PoinetNet in more details in the next section (Section 3.8).

3.8 Deep Learning for classification

Deep neural networks are neural networks with several hidden layers. It is one of the most powerful machine learning techniques. It has shown an incredible success, especially for its ability to learn several representations (see Fig. 3.15) and handle several data structures such as 2D data, 3D data, time series data, etc. In this section, we will present VGG-16 and PointNet neural networks the pioneer deep neural networks for image classification and 3D shape recognition respectively.

3.8.1 VGG-16

After the great success of the convolutional neural networks, especially in large-scale image and video recognition, many challenges arose in particular in image classification and object detection such as ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [116]. The ILSVRC challenge provides a large-scale dataset that includes about $1.4M$ images of 1000 classes. Many research teams participated and thus many architectures have been designed a submitted to this challenge like the VGG models [117] that won the second place in the ILSVRC-2014 challenge [116].

Karen and Andrew have investigated increasing the depth (up to 16–19) of neural network architectures while using only small convolution filters (3×3) rather than small networks with larger filter (7×7, 11×11, etc.). They designed several configurations using the same principle. The best configuration was submitted to the ILSVRC-2014 under the name of the team "VGG". The general configuration consists on a set of stacks of convolutional (conv.) layers followed by three fully connected layers and the classification layer. Each stack is followed by a 2×2 max-pooling layer with stride 2. The ReLU function is the activation function off all the layers except for the last layer that uses the soft-max function for classification.

VGG-16 is the best configuration with a minimum of the number of parameters. It consists of 5 stacks of conv. layers (16 weight layers in total) of 64, 128, 256, 512, 512 filters respectively. Fig. 3.16 shows the model's architecture: the input is a 224×224 RGB image which is passed to the first stack of conv. layers (with two conv layers) followed by a max-pooling layer that reduce the input size to half. The remaining stacks consist of 2, 3, 3 conv. layers. Each max-pooling layer produces an output equal to half of its input. The output of the last of the max-pooling layer is flattened and passed to three $1 \times 1 \times 4096$ fully connected layers. The last layer (the soft-max layer) provides the probabilities of the input belonging to each class.

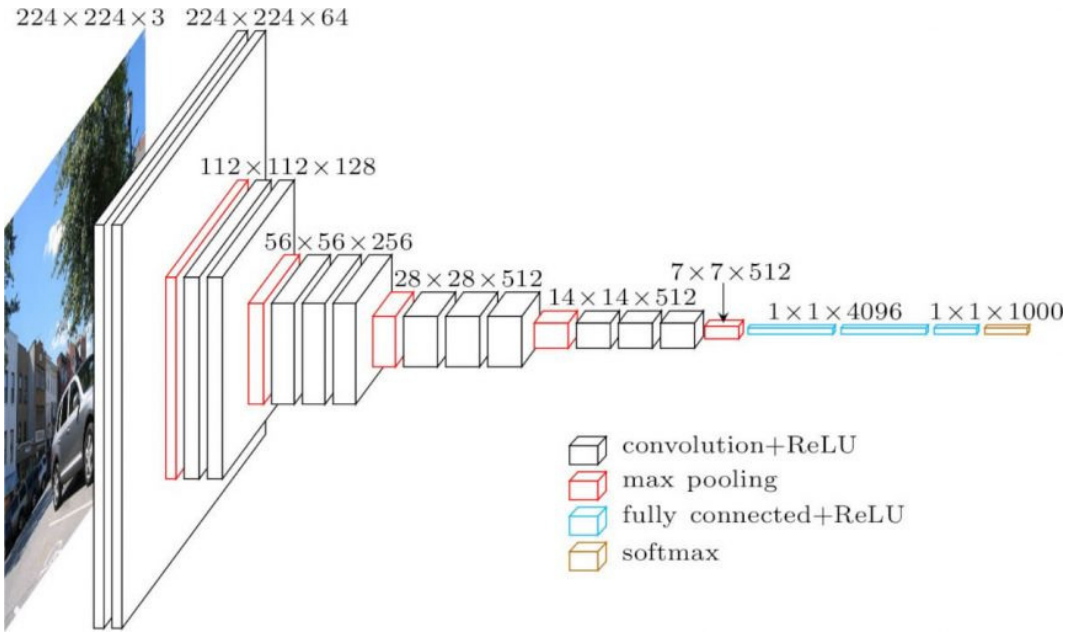

Figure 3.16: The VGG-16 architecture [12].

The VGG-16 model is widely used, The output of its dense layers is used as global descriptors in many applications.

3.8.2 PointNet

PointNet network [13] is the first proposed point cloud based model and a state of the art architecture. It is a deep multi-layer perceptron network that is designed for 3D raw data. It is invariant to permutation and transformation due to the use of max pooling, a symmetric activation function, and T-net that ensures pose normalization. It is also robust to small noise and incomplete data (with a small portion), but it is not able to capture fine local patterns. Although several architectures have been proposed over time such as [114, 115], PointNet is still used in different systems that involve point clouds [118, 119].

PointNet consists of three stacks of fully connected layers separated by an input transform unit (see Fig. 3.17). The first stack includes two layers with 64 perceptrons for each. It receives a $n \times 3$ vector, where n is the number of points, and generates

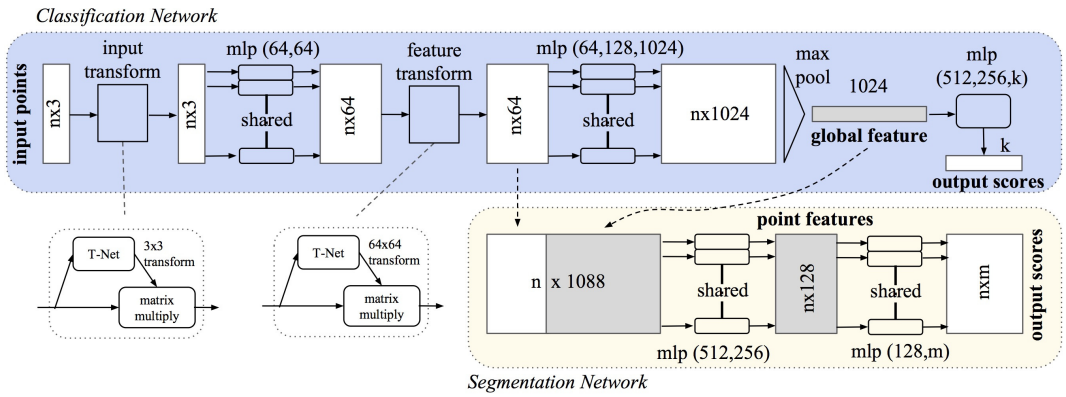

Figure 3.17: The PointNet architecture [13].

a 1×64 vector. The second stack is made of three layers: each layer consists of 64, 128 and 1024 perceptrons respectively. As for the third layer, it is represented by three layers of 512, 256 and k, where k is the number of classes.

The input transform unit, also called T-net, is applied on the model's input and the first stack output. It predicts the affine transformation matrix and applies it to the input points in order to normalize their pose. Like PointNet, T-net consists of a set of fully connected layers with max pooling as the activation function.

Despite the simplicity of the model, PointNet manages to effectively classify the 3D models. Indeed, it was shown that the second stack of layers was able to learns the shape; thus, its output is considered as global descriptors.

3.9 Conclusion

In this chapter, we reviewed different techniques for point cloud processing. These techniques are organized regarding their order in the processing pipeline. We first defined point clouds and how they can be obtained or computed. Since we will be using the Microsoft Kinect scanner as our input device, we explained in more details how to calibrate it and how to compute point clouds from depth images. After that, we present some filters that are used either for reducing the point cloud's size or to

reduce noise. This step is important when using techniques that are not robust to noise. We also presented some descriptors since we will be using them for point cloud classification. We also introduced the segmentation and classification techniques that are the main modules in our system. Finally, the last section depicted some machine learning techniques in more details. In the next chapter, we will present our 3D interface for human-scene interaction that can be used in many applications.

Chapter 4

3D Interface for Human-Scene Interaction

4.1 Introduction

In the previous chapter, we presented different techniques for point clouds processing since our 3D interface requires 3D data for a better human-scene interaction that is built on scene semantic labeling.

In this chapter, we present our designed device named Blind Assistive System for Intelligent Scene Reading (BASISR), that can be used as output device in many human-scene interaction based systems (see Section 4.2). After that, we will introduce how the scene can be mapped to the device, it can be considered as a basic system which maps the geometry of the scene on the device after only detecting the ground, as it will be presented in Section 4.3. In Section 4.4, we will propose an improved scene semantic labeling system based on plane detection. Section 4.5 will be devoted to validating each component of the proposed system including our proposed algorithm for ground detection. Finally, we will discuss the obtained results in Section 4.6.

4.2 BASISR: 3D interface for human-scene interaction

In this section, we will present in details our proposed device BASISR. For this purpose, we will first explain the advantage of having a 3D interface for human-scene interactions in Section 4.2.1. After that, the design of our device will be introduced in Section 4.2.2. We will show finally how a given scene can be mapped to this device by synthesizing or coding the point clouds.

4.2.1 Interests of the 3D interface

In chapter 2, we presented and discussed the limitations of the output interface of the presented systems, especially that are designed for the visually impaired and blind people. Therefore, our first challenge was to provide a scene description as it is captured by a depth camera. This description should initially indicate to the user his location in relation to the captured scene. In addition, it should provide the location of the scene's components as it is captured. Different information of different tasks for different scenarios should be encoded and provided in a simple way, so the user can use it to accomplish the desired task. For the second challenge, this information should be perceivable by the touch. By touching the output interface, the user should understand the provided information easily and without ambiguity.

Let's take the indoor navigation as an example, if the scene does not contain obstacles, we indicate to the user that the space is free from obstacles; so he will understand that he can navigate freely. If the scene contains an obstacle, we should provide the obstacle's location, so the user can locate it in the scene by touching and then can take the free space to navigate. Now, suppose that the scene contains different obstacles, we should provide their locations on the scene, so the user can also

take the free space to navigate. Additional information can be provided such as the obstacle's height and the occupied area and can be perceived only by touch without external devices. Labeling and mapping objects' locations and these additional information can be used for scene understanding; so the user can have an impression of his surroundings.

4.2.2 Design of the device BASISR

Our aim is to provide for human-scene interaction an area with a trapezoid shape that corresponds to the scaled view field of the camera. On this area, we will map the **coding** of point clouds of the 3D scene which is in the camera's view field as shown in Figure 4.1.

The parameters of the depth camera are used to design the synthesis area. We chose the ratio between the smaller and the longer bases of the trapezoid representing the camera's view field equal to 5 (for $L = 400cm$), the scaled area of scene synthesis verifies the same ratio as indicated in Figure 4.1 (right).

Let $b(x_b, y_b, z_b)$ be a given point from the point cloud that represents the scene geometry. The position of $b'(u_{b'}, v_{b'})$, the mapping of b on the synthesis area is calculated from the geometric relationship between the scene and the synthesis area. The values of $u_{b'}, v_{b'}$ are given by the equations 4.1 and 4.2 where d' is the small base of the synthesis area. The values of l, L for Kinect sensor are $80cm$ and $400cm$.

$$u_{b'} = d' \times x_b/d, \tag{4.1}$$

$$v_{b'} = d' \times (z_b - l)/d. \tag{4.2}$$

In order to reflect the scene's content, its correspondent semantic labeling or whatever we want to transmit to the visually impaired, the tangible area is provided with pins

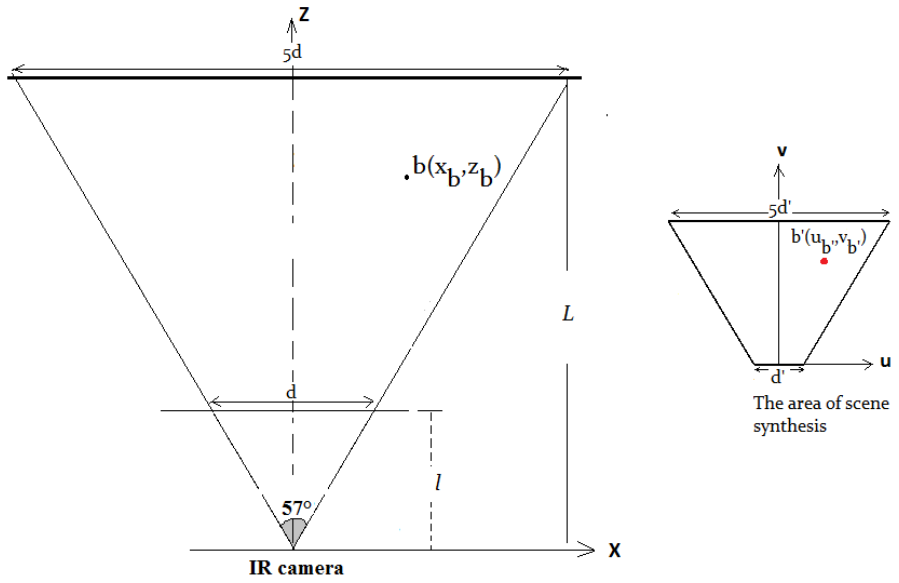

Figure 4.1: (Left) Top view of the depth camera's view field, only x-z axes are represented. (Right) To view of the area of scene synthesis.

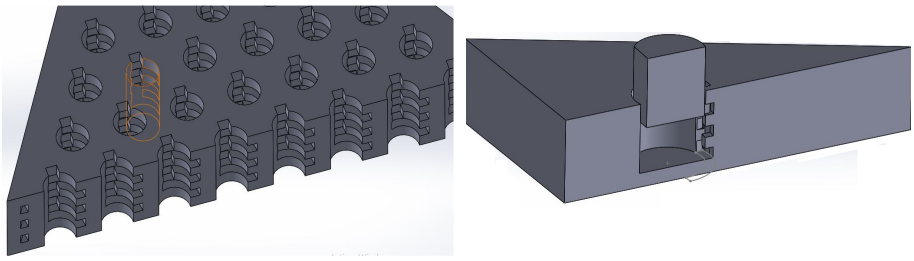

Figure 4.2: The proposed design allows to give to each pin different heights.

that can have different heights and form different shapes. Figure 4.2 and Figure 4.3 show the device's design. The tactile area of the device has a set of lines and columns of pins which can be raised to a different level. The 3D printing of the device is shown in Figure 4.4. An electronic layer will be added below the device as an additional component to control pins' height each configuration.

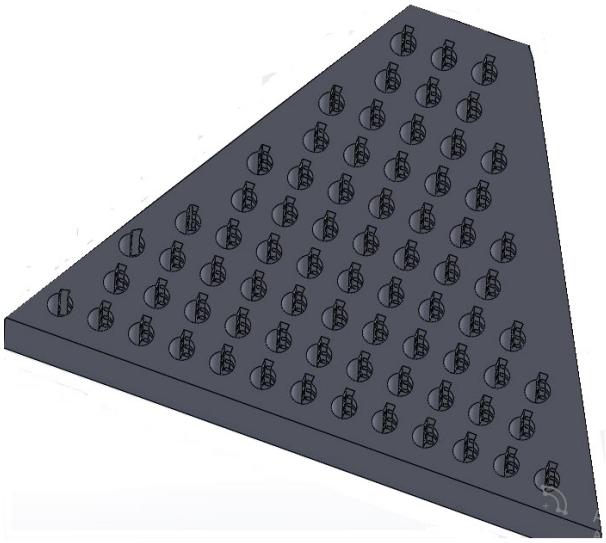

Figure 4.3: The bottom view of the device where a mechanical piece will be linked to each pin to allow to push it towards the top of the device with different distances. The motion of the mechanical piece will be performed via an electronic component.

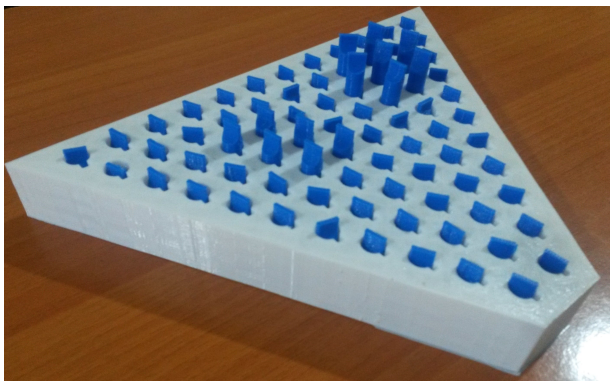

Figure 4.4: 3D printing of the proposed device. Note that some pins have set up at different heights.

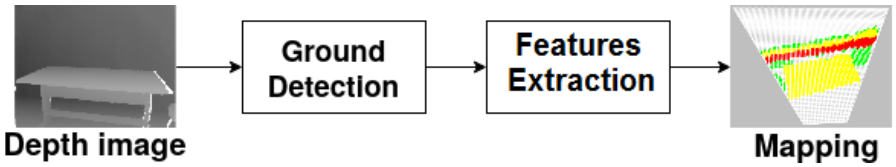

Figure 4.5: Scene coding on BASISR: a basic system.

4.3 Mapping the scene on BASISR device

In order to transmit to the BASISR device the 3D raw data or labels, three steps are performed (Figure 4.5): (1) we detect the ground as it will be presented in Section 4.3.1. (2) we extract features from the 3D scene, (3) we map extracted features on BASISR as it will be explained in Section 4.3.2.

4.3.1 Ground Detection

As mentioned before, detecting the ground is generally a first step in many applications with human-scene interaction. For this purpose, we will present DCGD, our proposed algorithm for ground detection.

The DCGD algorithm is designed as follow:
Let $(Oxyz)$ be the coordinate system attached to the RGB-D sensor mounted on the human body. The RGB-D sensor can perform any rotation around the three axes and can also perform a translation following three directions. We assume that the Kinect sensor can have any pose without constraints on its orientation.

Let $p(l, c)$ be the pixel located at row l and column c in the depth image $I_d(n \times m)$ and let $P_{l,c}(x, y, z)$ be the associated 3D point in the scene where the coordinates x, y, z are computed using the sensor parameters obtained after calibration.

The first step of our method is to select for each depth z_i and for each column c in the depth image I_d, the pixel $p^*(l, c)$ such that its associated 3D point has the $z - component$ equal to z_i and a minimal value of $y - component$ for all $P_{l,c}(x, y, z_i)$,

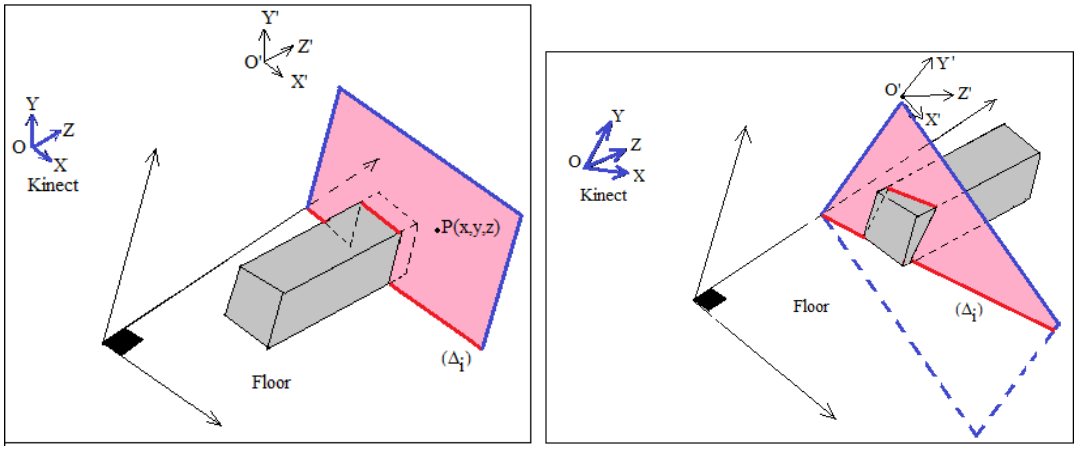

Figure 4.6: The obtained curve (G_i) in cases where (left) the $xz-$plane is parallel to the ground, (Right) the $xz-$plane has an unconstrained orientation.

$l = 1..n$. This allows to determine images of all projected points P of the scene having the same depth z_i as indicated with red color in Figure 4.6. The set of the obtained pixels for a given depth z_i defines a curve noted (G_i).

The curve (G_i) may be considered as the intersection of the $x'y'-$plane with the scene, where the coordinate system $O'x'y'z'$ is the translation with a vector $(0, 0, z_i)$ of the coordinate system $Oxyz$ attached to the Kinect sensor.

Depending on the orientation of the $xz-$plane relatively to ground, the curve (G_i) may change as it is shown in Figure 4.6. Consequently, the curves (G_i) are considered as cuts on the point cloud of the scene.

For each curve (G_i), the set of aligned pixels constitutes candidate pixels of the ground. Indeed, the second step consists to remove parts of (G_i) corresponding to obstacles.

In order to facilitate the geometrical illustration, we draw the curves G_i considering that the Kinect sensor performs only a rotation around x-axis. If the ground is a plane without any convexity or concavity, all curves (G_i) for all z_i will be lines (see Figure 4.7(a)). We assume now that the presence of an object on the ground. The curve (G_i), in case of an isolated object on the ground, will have the shape shown

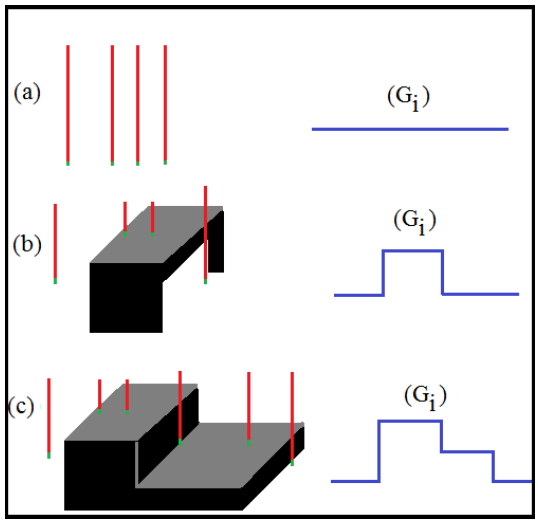

Figure 4.7: Example of 3D points at a given depth (colored in green) projected and define the curve (G_i) in case where the curve (G_i) is a line (a), contains two concave parts (b), contains three concave parts (c).

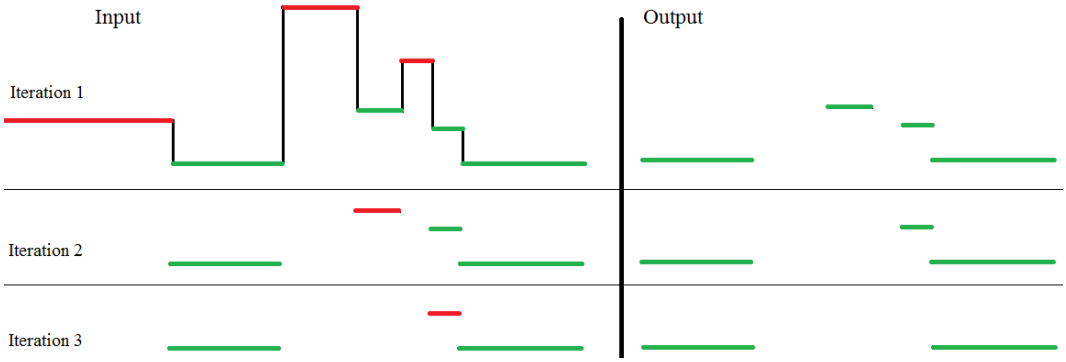

Figure 4.8: Removing iteratively convex parts (in red color) from G_i to keep only the ground corresponding to concave parts (green color)

Algorithm 4 Algorithm *DCGD* (Depth-cut based Ground Detection)

Input: $I_d(n \times m) = \{p(l,c), l = 1..n, c = 1..m\}$,
The point cloud $\mathbb{P} = \{P_{l,c}(x,y,z)\}$
Output: The set $\mathbb{G}$ of ground pixels $p^*(l,c)$

$\mathbb{G} \leftarrow \emptyset$;
Determine $\mathbb{Z} = \{z_i / \exists p(l,c) \in I_d,\ P_{l,c}(x,y,z)\ verify\ z = z_i\}$;
for *each* $z_i \in \mathbb{Z},\ i = 1..Card(\mathbb{Z})$ **do**
  $G_i \leftarrow \emptyset$;
  for *each column* $c = 1..m$ **do**
    for *each* $p(l,c),\ l = 1..n$ **do**
      Determine $p(l,c)$ *such that* $P_{l,c}(x,y,z)\ verify\ z = z_i$;
    end for
    Select $p^*(l,c)$ *associated to* $P_{l,c}(x,y^*,z^*)$ *such that* $z^* = z_i$ *and* $y^* = Min(y - coordinate\ of\ P_{l,c})$;
    $G_i \leftarrow G_i \cup \{p^*(l,c)\}$;
  end for
  Remove from G_i convex parts and keep only concave parts;
  $\mathbb{G} \leftarrow \mathbb{G} \cup G_i$
end for
return $\mathbb{G}$

in Figure 4.7 (b) with a convex part (lines curved outward) and two concave parts (lines curved inward). However, when some objects are grouped, then the curve (G_i) will contain many convex and concave parts as shown in Figure 4.7(c). Note that the slope of all (G_i) is the same and depends on the rotation of the sensor around z-axis.

Each (G_i) is composed by a set of convex and concave parts. By scanning the curves (G_i) from left to right, we associate a label (*cv* for convex or *cc* for concave) to each part. All *cv* parts are removed. The labelling and convex parts removal are repeated until there will be no convex parts. We give in Figure 4.8 an example of how the ground is determined from (G_i) . The algorithm 4 summarizes the steps to be performed for the computation of the curves (G_i).

4.3.2 Scene synthesis on the BASISR device

After detecting the ground, we extract the occupied space and compute the ground height. From the points constituting the ground, the ground height is represented by the y coordinate of the 30^{th} percentile to avoid potential noise. Finally, we compute the height of each point of the occupied space and map them on BASISR. The height of a given point (h_i) is represented by the distance between the point and the ground y coordinates, y_i and y_g.

Figure 4.9 displays two examples of this system. From the capture scene (Figure 4.9 (top)), the system generates the scene coding (Figure 4.9 (bottom)) from the depth image (Figure 4.9 (middle)). Each point of the occupied space is mapped individually regarding its height and its location. This system can be used for scene understanding and for navigation by providing the scene geometry and distinguishing between the free space and the occupied space respectively. However, it does not consider objects as separate individual units.

In Figure 4.10, the point cloud of the captured scene is directly mapped on the area. The same point can be mapped into the synthesis area using different heights of the pins (see Figure 4.11). This is done if the pin is set up to **5** different levels: the level zero is used to represent the holes on the ground. The level 1 represents the ground or the neutral element in some applications. The height of the illustrative labels that are mapped in the center of the convex hull, is set to the computed level (i.e. level 2, 3 or level 4) and the height of the remaining area of the convex hulls associated with the segment is set to level 2. For more visibility, we assigned the grey color, the green color, the red yellow color and the red color to represent objects with level 2 (ground), 3, 4 and 5 respectively.

This representation is simple and basic: the points are directly mapped on the device (see Figure 4.10). However, it cannot be adapted in the real world. Indeed, mapping all points in their raw representation can generate ambiguity. In addition, it cannot

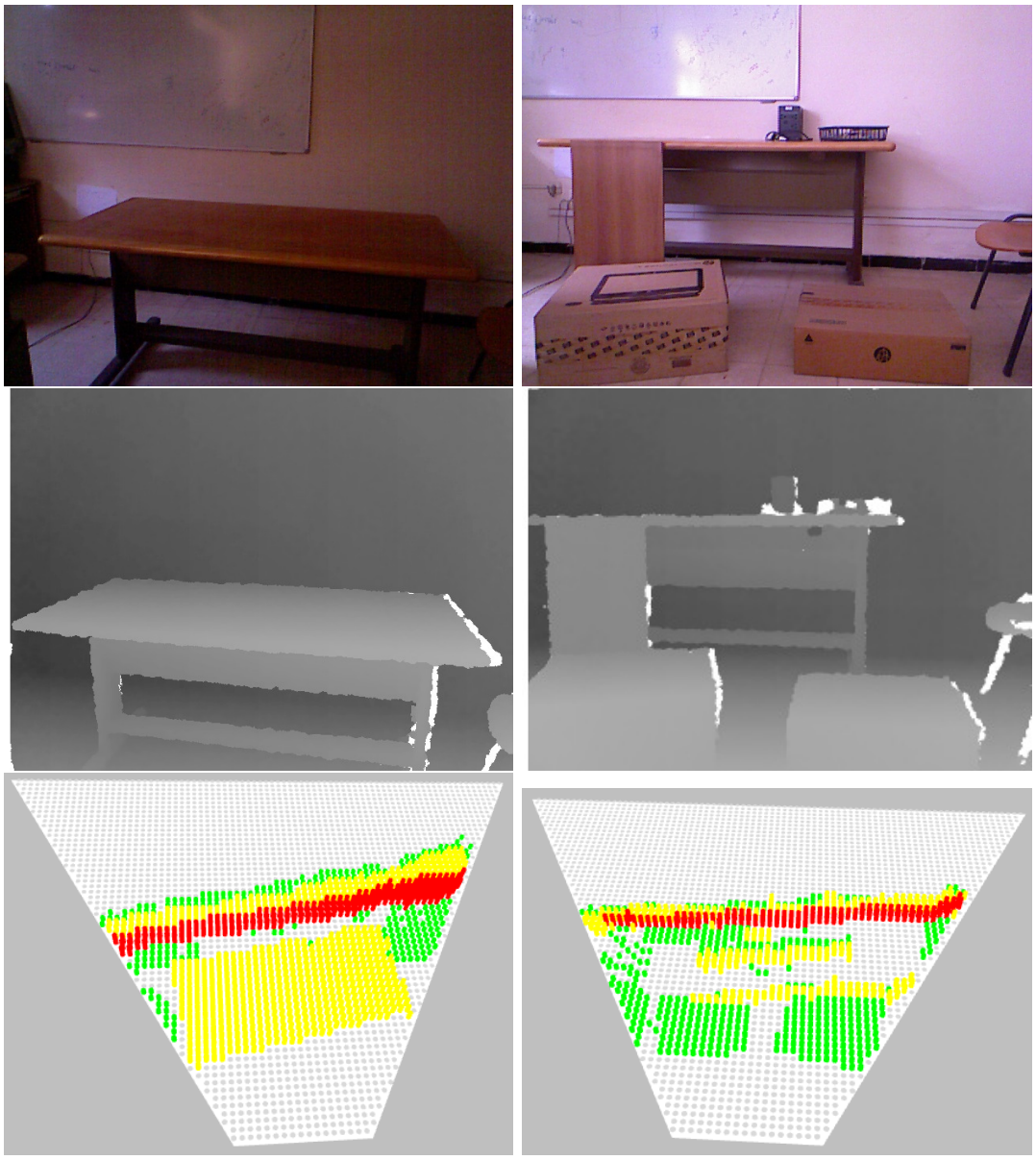

Figure 4.9: Scene coding on BASISR device. From top to bottom: the captured scene, the depth image (the system input) and the synthesized scenes (left and right). For more visibility the color green, yellow and red represent level 1, 2 and 3 respectively.

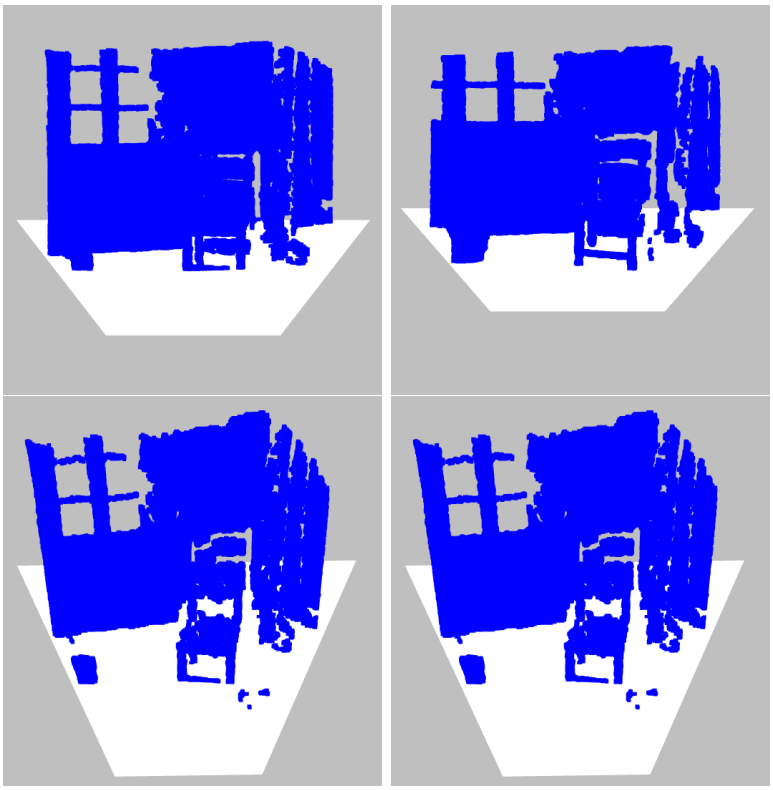

Figure 4.10: Mapping the point clouds on the area of the scene synthesis: all points are inside this area.

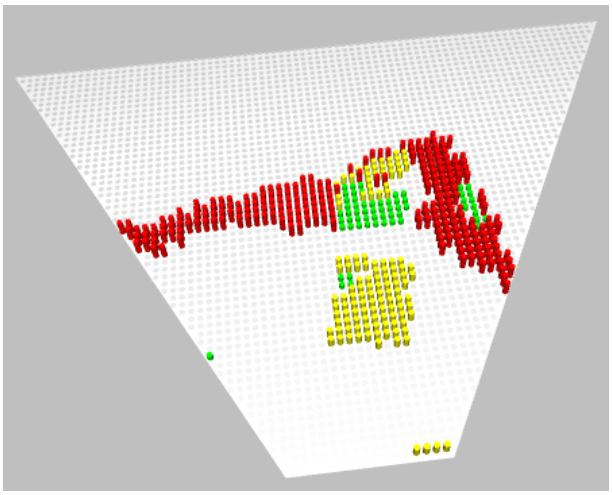

Figure 4.11: Scene synthesis from raw data (point cloud) using pins. The color gray, green, yellow and red represent levels 0, 1, 2 and 3 respectively.

be generated physically using a device: points require support, they cannot be placed in the air, and the concave surfaces cannot be represented using pins. Another drawback is that there is no real semantic meaning: although it is perceivable to the eye, it is more difficult to understand by touch and it takes effort and time.

4.4 3D scene labeling on BASISR device based on plane detection

As the previous scene labeling does not detect and classify objects, in this section, we introduce an improved system that consists mainly of: ground detection (that is already presented), point cloud segmentation that segments the occupied space into planes (it will be presented in Section 4.4.1), point cloud classification that classifies each plane into the proposed classes (it will be presented in Section 4.4.2), scene labeling that generates the 3D labels of each detected class (it will be presented in Section 4.4.3) and the 3D interface, BASISR, that is used to provide the user with the labels for a human-scene interaction. Our presented system is based on plane detection and classification. The entire system pipeline can be summarized in Figure 4.12.

4.4.1 Plane detection

After breaking down the depth image into the free space (ground) point cloud and the occupied space point cloud (Figure 4.12 (a)) and to avoid the noise produced by the nature of Micorsoft Kinect, we first applied the pass-through filter to only accept points that their depth is between $800mm$ and $4000mm$. Then, we down-sampled the occupied space point cloud (Figure 4.12 (b)) to reduce the number of points to be processed and thus decrease the computational complexity of RANSAC. For the

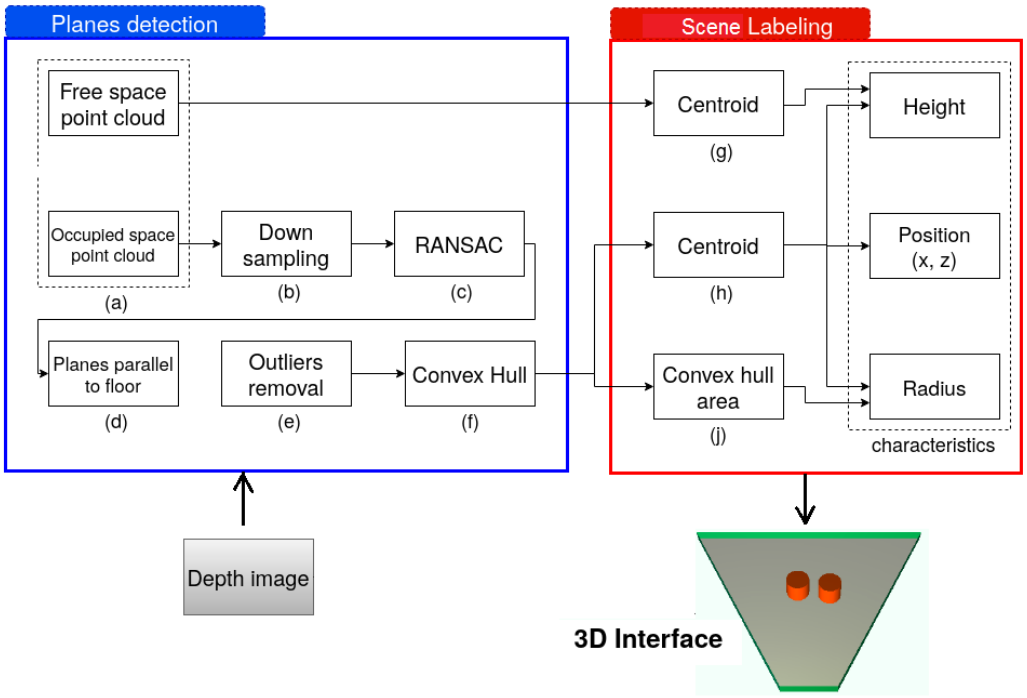

Figure 4.12: Scene labeling for human-scene interaction: First prototype. It receives a depth image as input, detects the ground, extracts horizontal planes and generates a 3D interface as output based on the provided scene labeling.

down sampling, we used the voxelization filter with a grid of $3 \times 3 \times 3cm^3$ cells, each voxel is replaced by its centroid. The hyperparamters were set by experiments.

After that, we applied RANSAC (Figure 4.12 (c)) for plane segmentation on the reduced occupied space point cloud. To reduce the RANSAC's run time and the number of insignificant possible planes, we set the distance error threshold up to $2cm$. This latter may affect the result by accepting some outliers, but it will be proportionally handled later.

4.4.2 Planes classification

In our surroundings, every object has a location, a shape, a geometry and dimensions. However, the eye of the human being is not capable to estimate the exact size of objects, the estimate of the size is approximate. Therefore, we propose a promising yet simple to compute object classification regarding two features: the object height and the occupied area. This latter can also be extremely helpful for the visually impaired and blind people.

Regarding the height, we define three classes: the first class defines objects with a height less than $0.3m$; it represents objects that can be traversed by feet. The second class defines objects with a height greater than $0.3m$ and less than $1.6m$; it represents objects that can be touched by hands. The objects with height greater than $1.6m$ represent the third class; it represents objects that are higher than the average height of humans.

As for the occupied area, we also define three classes: the first class represents the objects occupying small area with a radius less than $25cm$ that is can be explored only by moving hands without effort. The second class represents the objects occupying a medium area with a radius larger than $25cm$ and less than $50cm$. This type of objects can be explored with hands, but may need stretching the arm. The third class represents the objects with a huge area, with a radius larger than $50cm$, which

cannot be fully explored just by stretching the arms, but may also require movement around the plane.

The object's location is another important factor for classification. Indeed, it represents the relationship between objects and their arrangement in the scene. For instance, the night-stands are usually placed alongside the wall and next to the bed rather than in the center of the room unlike tables which are generally placed in the center.

With this approach, we designed 3 classes for each feature; but while combining them, we actually designed 9 classes for instance. Indeed, each object is classified according to its height and surface; for instance, the chair belongs to the 2nd and 1st class according to its height and width respectively. In addition, the location will be taken in consideration in a straightforward way as it will be explained in Section 4.2.

In order to classify the planes into the designed classes, we proceed as follows: after extracting the planes, we identified parallel planes to the ground (Figure 4.12 (d)). A plane π_i is parallel to the ground plane π_G when the cross product of their respective normal n_i and n_G is equal to the null vector:

$$n_i \times n_g = \vec{0}. \tag{4.3}$$

We only considered these planes, assuming that are the objects' surface and they occupy a space in the captured scene unlike vertical planes for example.

In order to get the occupied space, we extracted the convex hull (Figure 4.12 (f)) encompassing each plane parallel to ground. To enhance the RANSAC's result and due to the sensibility of the convex hull technique toward noisy data, we first projected the plane into its equation and then applied the statistical outlier removal filter (Figure 4.12 (e)) to refine the projected plane boundaries before extracting the plane's convex hull.

At the end of the planes detection process, parallel planes to the ground are represented by their convex hulls. These latter are used later to extract occupied space characteristics.

4.4.3 Scene labeling

In Section 4.4.2, we classified the input regarding their heights and occupied space. In this approach, we aim to represent these features simultaneously. Thus we represented each class (of 9 classes) with a cylinder with a specific radius, height and location. The reason why we chose a cylinder and not other 3D shapes is the fact that it only admits these two properties and that is invariant to rotations since we do not take into consideration the input pose.

The inputs are represented by a cylinder having a specific characteristics that are related to the input characteristics in the real world. Each cylinder has its position (the coordinates x and z of its center), its height that represents the class's height and its radius to represent the class's occupied area.

To conduct this, we compute the centroid and the area of the concerned convex hull; and the free space point cloud centroid. The center's position is represented by the plane's centroid coordinates (x and z coordinates). The height is found by subtracting the y-coordinates of the plane centroid and the ground's centroid. As for the radius, we searched for the radius of a circle having the same area as the area of the convex hull. This is done for each plane that is parallel to the ground plane. The computed cylinders are then transmitted to the user via the 3D interface as shown in Figure 4.12 (3D interface).

Figure 4.13 illustrates the generated cylinders corresponding to the provided labeling of three scenes. The first scene was taken in our first position and second scene was taken few steps ahead. In the two scenes, the generated scene indicates that there is a free space followed by an object having a height less than $300mm$ and it can be

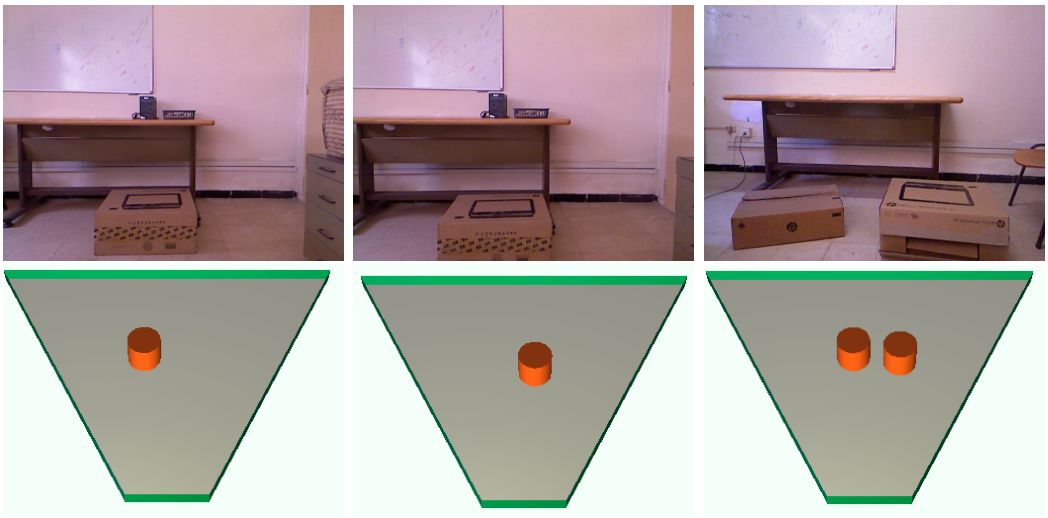

Figure 4.13: First row: scene 1: initial position. Scene 2: after few steps ahead. Scene 3: with two obstacles. Second row: the coding on the generated scene.

explored with hands, but may need stretching the arm (its radius is less than $500mm$ and larger than $250mm$). Note that in the second scene, the position of the cylinder changed to let the user understand that the obstacle became closer. In other words, the area of free space has become smaller. As for the third scene, the generated scene indicates that there are two objects, after a free space, having as height less than $300mm$ and they can be explored with hands but may need stretching the arm.

4.4.4 BASISR: use case scenarios

The proposed device can be used as an output device for many applications that are based on human-scene interaction, including free space detection, scene understanding, indoor navigation and locating and grasping objects. It depends on the interpretation that we give for different device's configurations.

It can be used for scene understanding as shown in Figure 4.14, 1st row: a small object and a large object are detected; the large object is behind the small object and the remaining space is a free space. In case of indoor navigation (Figure 4.14 2nd row), the free space can be represented by pins with level 0 and level 3 represents

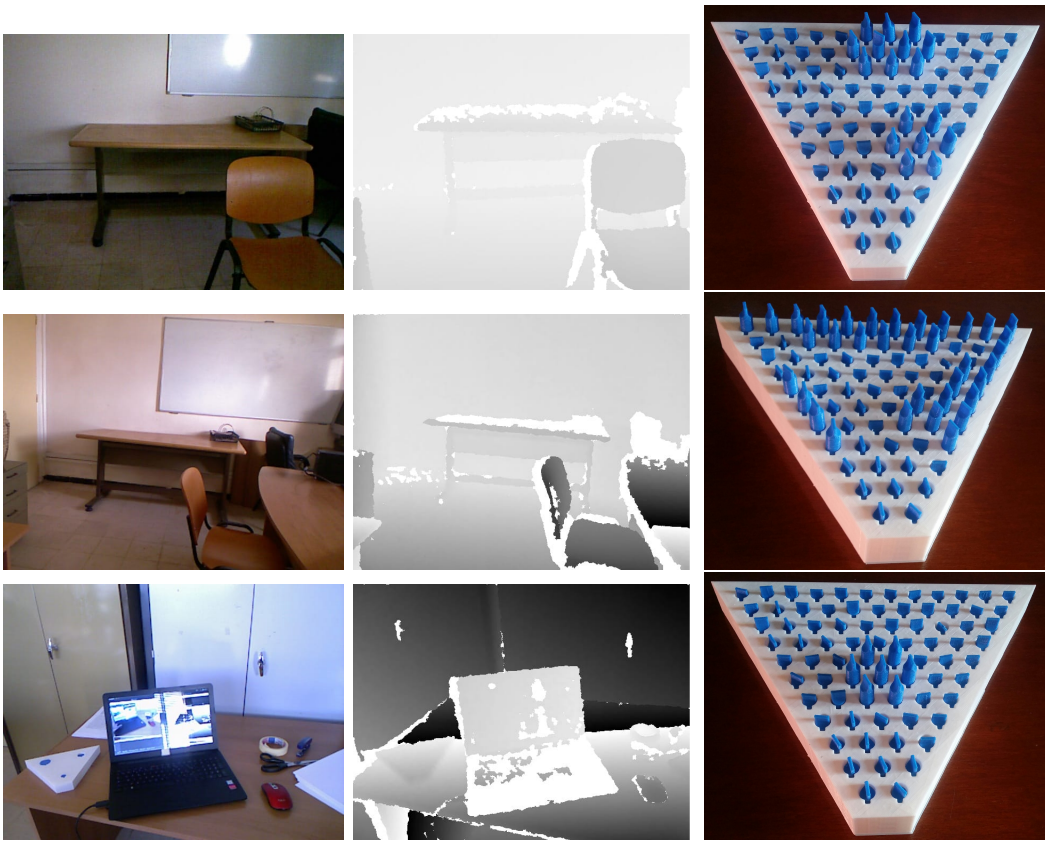

Figure 4.14: Our proposed framework for: scene labeling, indoor navigation and object searching (from Top to Bottom). From left to right: RGB image of the captured scene, depth image as input data, the output mapped to the device.

the occupied space. As for object searching (Figure 4.14 3rd row), the target object can be represented by pins with level 3. Figure 4.14 3rd row shows an example of searching object application where the target object was the laptop.

4.5 Validation

In this section, we evaluate and test each designed module, namely: the ground detection algorithm in Section 4.5.5 and the segmentation module in Section 4.5.6. These modules will be validated using our constructed dataset and public datasets. The validation metrics are presented in Section 4.5.1.

4.5.1 Validation metrics

The model/algorithm validation involves computing metrics to illustrate the behavior of the latter against different types of data (training data, validation data or/and test data). The obtained results will be displayed/visualized and interpreted. These metrics consist in comparing the results obtained by the model with the real values/classes (the ground truth).

The metrics to be computed strongly depend on the nature of the model output: if it is a regression (or a continuous variable) the Mean Square Error (MSE) or Mean Absolute Error (MAE) can be used; If it is a classification (or a discrete variable) the precision, recall, f-measure and confusion matrix can be used.

Before presenting these metrics, we present 4 important terms, namely:

- **True positive (TP)** is where the model correctly predicts the positive class.
- **True negative (TN)** is where the model correctly predicts the negative class.
- **False positive (FP)** is where the model incorrectly predicts the positive class.
- **False negative (FN)** is where the model incorrectly predicts the the negative class.

4.5.2 Evaluation metrics for regression

For what follows in this section, we note the size of the dataset, the instance class and the predicted class by N, y, y_{pred}.

- MAE is the mean absolute difference between the predicted and the actual values it represent the mean distance between them. It is widely used for its simplicity:

$$MAE = \frac{1}{N}\sum_{j=1}^{N}|y_j - y_{pred_j}| \tag{4.4}$$

- MSE is represented by the average of the square of the error that is difference between the predicted and the actual values:

$$MSE = \frac{1}{N}\sum_{j=1}^{N}(y_j - y_{pred_j})^2 \tag{4.5}$$

- Root Mean Squared Error (RMSE) is the root of MSE. It is a way to normalize the MSE since it changes the units because of the exponent operation. Its formula is:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{\sum_{j=1}^{N}(y_j - y_{pred_j})^2}{N}} \tag{4.6}$$

4.5.3 Evaluation metrics for classification

- Accuracy: is the fraction between the correct predictions, true positive and true negative, and all predictions.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{4.7}$$

- Precision: represents the positive class predictions that belong to the positive class. It provides how much examples are actually positive out of all the positive classes that have been predicted correctly.

$$Precision = \frac{TP}{TP + FP} \tag{4.8}$$

- Recall: represents positive class predictions made out of the positive examples in the dataset. It provides how much examples was predicted correctly out of all the positive classes.

$$Recall = \frac{TP}{TP + FN} \tag{4.9}$$

- F-measure: is a single score that balances both precision and recall.

$$Fmeasure = \frac{precision * recall}{precision + recall} \tag{4.10}$$

- Confusion matrix: gives a summary of all the correct predictions of each class and all the confusions between each class. It provides a detailed insight of how the model performs and which kind of error it makes. For instance, thanks to confusion matrix, we can say for a given class what are the classes that have confusion with it and how much. In addition, if two classes have high confusion between them, we can understand that the model find it difficult to distinguish between them.

- Receiver Operating Characteristic (ROC) curve: is graph that displays the performance of a model at all classification thresholds. It shows the sensitivity and specificity of the model using the true positive rate and the false positive rate. A typical ROC curve is a curve where false positives and true positives increase at the same time.

4.5.4 Datasets

In order to validate the ground detection algorithm on public dataset, we will use: NYU Depth dataset V2 [2]. Ground Detection in Indoor Scenes (GDIS) [15] dataset that will be also used to validate the ground detection algorithm and the geometric feature extraction.

GDIS dataset

Several datasets exist in the literature; however, there is no dataset that is especially devoted for ground detection from depth images. Some datasets include the *'ground'* class as one of the considered classes [2]. Some others consider the ground as part of the background [120].

Figure 4.15: (Top) Color and depth image, (bottom) ground colored with red color.

GDIS includes different depth and color images acquired by an RGB-D sensor (Microsoft Kinect V1) with various orientations and poses. The ground truth corresponding to the floor is indicated in both images (depth and color) (Figure 4.15). Each pixel is classified as ground or not the ground. The image labeling was done by running the DCGD algorithm on the depth images. Regarding the RGB images, we have performed an alignment and mapped the ground pixels to the RGB image. The dataset is still under construction as we aim to add more interior with more complex scenes.

NYU Depth dataset V2

NYU Depth dataset V2[2] (Figure 4.16) includes video sequences from a variety of indoor scenes recorded by both the RGB and Depth cameras from the Microsoft

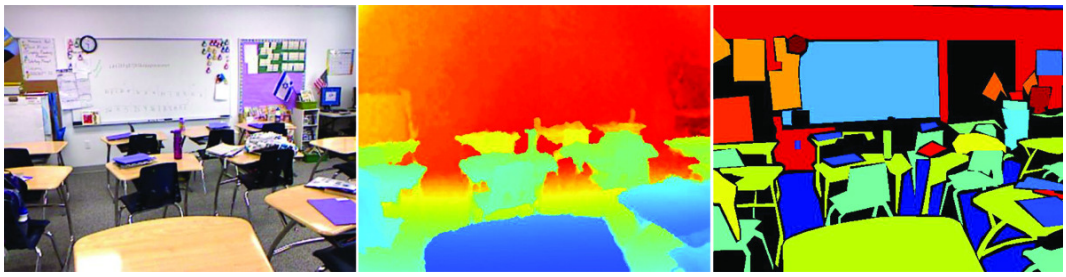

Figure 4.16: Dataset samples: NYU Depth dataset V2 [2]. From left to right: the RGB image, the preprocessed depth image and the image of labels.

Kinect. It consists of 1449 densely labeled pairs of aligned RGB and depth images. Each pixel in the image is labeled with a class and an instance number. The number of classes is 894.

4.5.5 Evaluation of DCGD algorithm

In order to evaluate the proposed algorithm, DCGD, we first explain some implementation details. Then, we analyze the defined parameters and compare the results in function of these latter. After that we evaluate it using GDIS and NYU depth dataset V2 respectively, and compare it with the state of the art.

Implementation details

The proposed algorithm can be divided into two main steps, namely: curves (G_i) construction (Figure 4.17 (top)) and ground detection. The first step is done in one shot by browsing the depth image only once. The complexity of this step having as an input a depth image with n pixel is $O(n)$. Furthermore, a down sampling by depth *step* can be performed, it reduces the complexity and improves the detection quality.

The second step includes subdividing of a curve into sub-curves and finding the ground from objects. The subdivision (Figure 4.17 (bottom)) can be performed

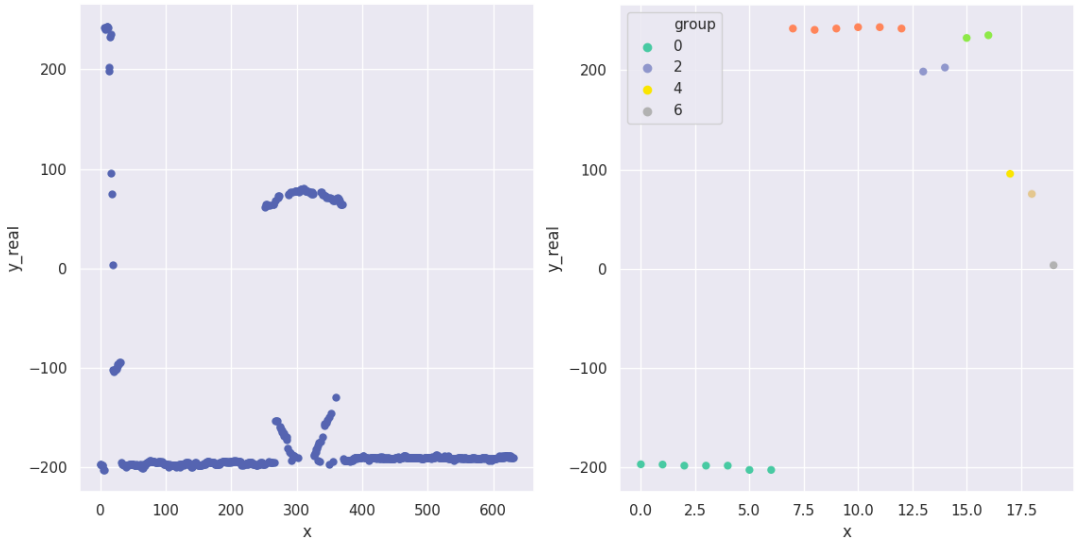

Figure 4.17: DCGD algorithm: curve reconstruction plot (left) and curve subdivision into sub-curves (right) (only the first seven groups are plotted.

according to certain criteria such as the permitted error in height between elements in same sub-curve h_err or the distance from the mean. Setting the value of these latter (height and down sampling step) depend on the sensor nature. This step is applied to all the obtained curves; so we need $2k$ iterations, including curve subdivision and floor finding from k curves. The worst case is either we have n cuts or we need n iterations to detect the ground. Thus, the proposed algorithm's complexity is $O(n)$.

Another step can be added to reduce noise: some sub-curves can be generated due to noise and affect the floor detection; thus, removing sub-curves having small size $size\_err$ can reduce noise as shown in Figure 4.18.

Parameter analysis

To evaluate the effectiveness of each discussed parameters, namely $step$, h_err and $size\_err$, we plotted the algorithm performance by varying each parameter as shown in Figure 4.19. Regarding h_err parameter, we notice an improvement in the values of the three metrics by increasing its value. However, the value of the precision starts

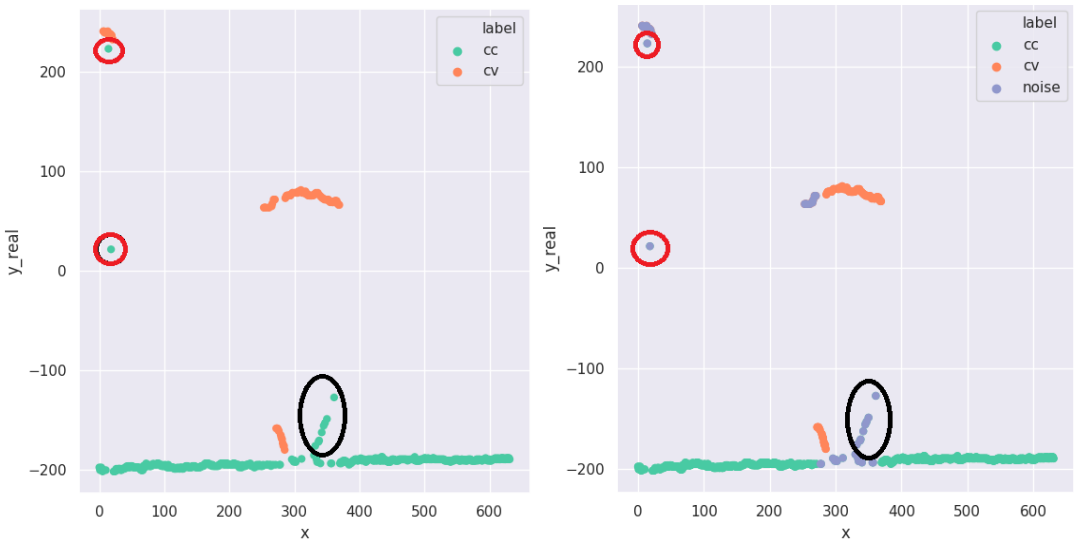

Figure 4.18: DCGD algorithm: without reducing noise (left) and with reducing noise (right). Note that by reducing noise, we prevent some false positive cases (circled in red). Note also, other noise area (circled in black) was appeared on the object borders.

to decrease from the value 27.5. Regarding the *step* parameter, the value of the recall improves with increasing the step, the value of the f-measure improves modestly. However, the precision starts to decrease from the value 4. As for *size_err*, we notice an improvement in the values of the three metrics by increasing its value. For best performance and real-time execution, we set *h_err* to 30, *step* to 4 and *size_err* to 15.

Algorithm evaluation on GDIS dataset

We applied our method for indoor scenes. The ground is located in real time with accuracy. Different scenarios have been tested with different orientations of the Kinect sensor. Figure 4.20 shows qualitative results for a sample of scenes. Note that some pixels of the ground are missing due to noise. As the color image is larger than the depth image, the left and right of the color image did not appear in depth image and thus not processed (see Figure 4.20).

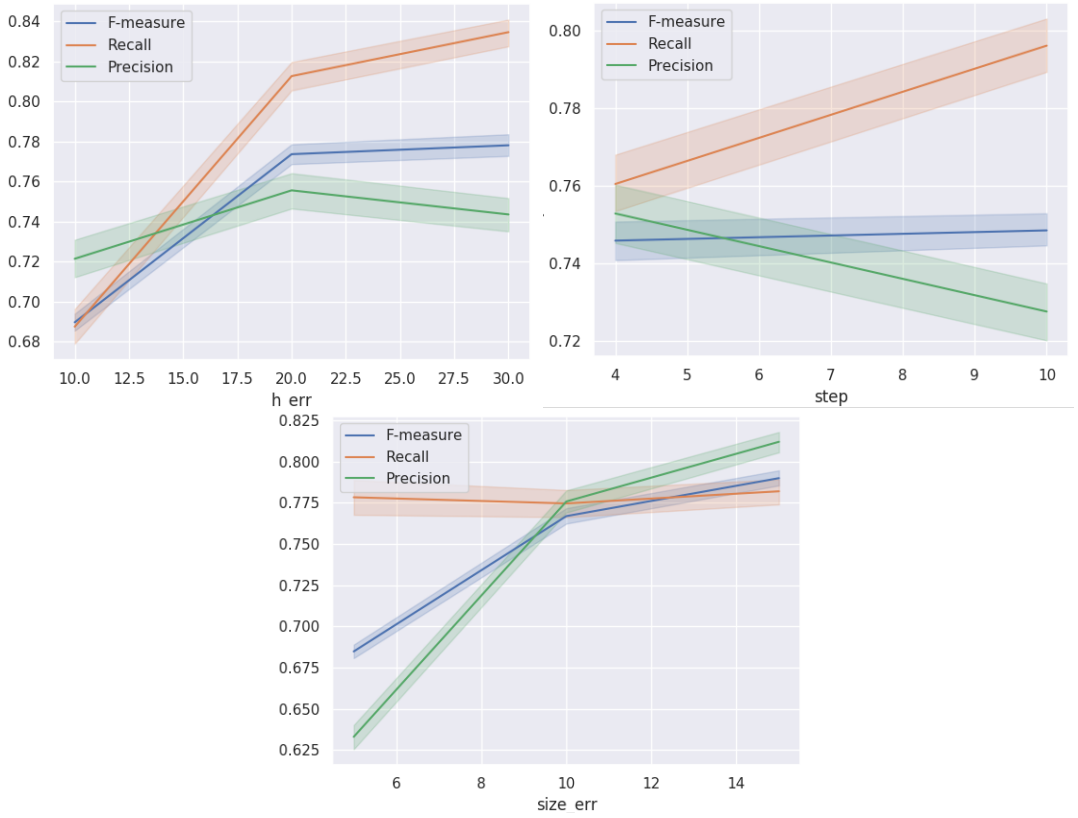

Figure 4.19: DCGD evaluation while varying *h_err*, *step* and *size_err*.

Figure 4.20: From left to right: Color image, depth image, ground colored with red color.

Figure 4.21 gives more details on computing the curve (G_i) for one value of depth by applying the algorithm 4. The drawn (G_i) contains a convex part corresponding to foot table, it is removed in the next step. Note that pixels defining the curve (G_i) aren't aligned because 3D points at the same depth z_i have different values of the depth. We then selected all pixels having the value $z_i \pm 10mm$. We note that the area under the chair is detected as ground, which corresponds to the truth. However, if we search to locate obstacles, the plane above this ground's area will be taken into account. In the next example shown in Figure 4.21, three depths are considered. All (G_i) have the same slope corresponding to the orientation of the Kinect sensor with respect to to the z-axis.

The measures, Precision, Recall and F-measure have been computed considering that the ground truth of the ground begins from the furthest pixel on the depth image as indicated in Figure 4.22. The mean of the computed measures are: $Precision = 0.98$, $Recall = 0.93$ and the $F - measure = 0.96$.

Algorithm evaluation on NYU Depth dataset V2

We evaluated *DCGD* Algorithm using NYU Depth dataset V2 [2]. The proposed algorithm scored a good performance nonetheless, it fails in some cases where depth images are noisy. Figure 4.23 shows different results obtained with high, medium and low score of accuracy. Our proposed algorithm scored the highest accuracy (91.84%) for ground detection compared to the proposed one in [121] (80.3%). Thus, the *DCGD* algorithm is more robust, especially when dealing with noisy data compared to [121].

The results are reported in table 4.1: despite the complexity of certain scenes in the dataset: the algorithm remains efficient (the minimum value of the f-mesure is 78.26 %). Figures 4.24 and 4.25 respectively illustrate the distribution of the values of different metrics as well as the ROC curve and the confusion matrix. The

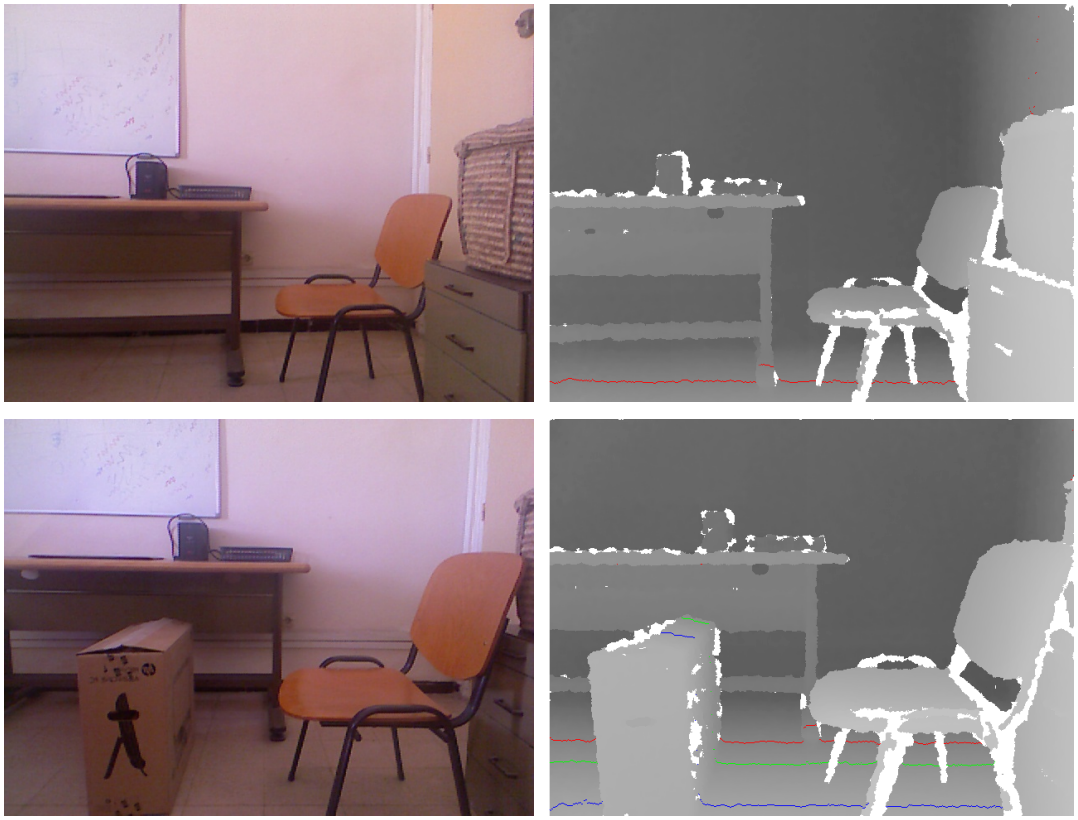

Figure 4.21: First row: The color and its associated depth image of indoor scene. The white pixels represents the noise (depth equal to zero). The computed curve (G_i) for depth $z_i = 2000m$ is drawn in red color. Note the presence of an obstacle (the table's foot) which produces a convex part in (G_i) is removed in the next step. The discontinuity of (G_i) is due to occlusion of the area located at the depth z_i by the feet of the chair.Second row: Located curves (G_i) for three depths z_i equal to $1500mm$, $1800mm$, $2000mm$ drawn respectively in blue, green and red color. For the third depth, the missing part of (G_i) is due to the occlusion of the corresponding area by the box.

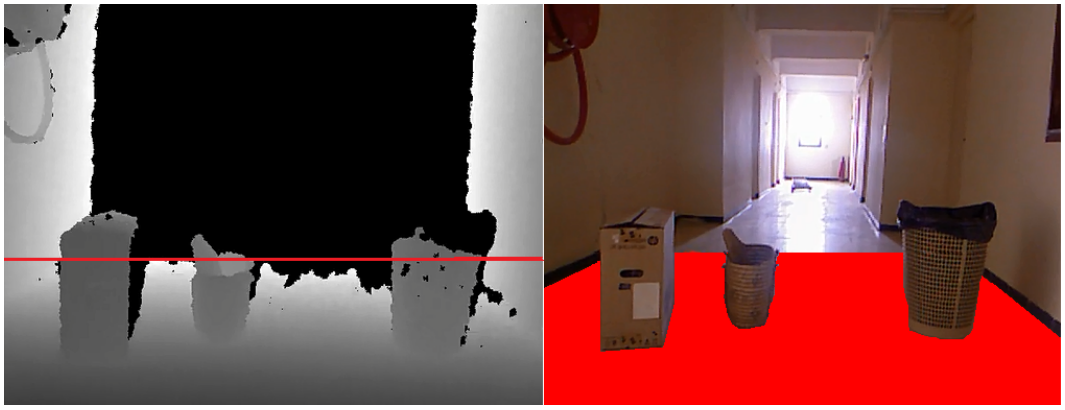

Figure 4.22: Determining ground truth data. (Left) the extremity of the ground truth localized as the horizontal line passing by the pixel with maximal depth. (Right) The drawn ground truth.

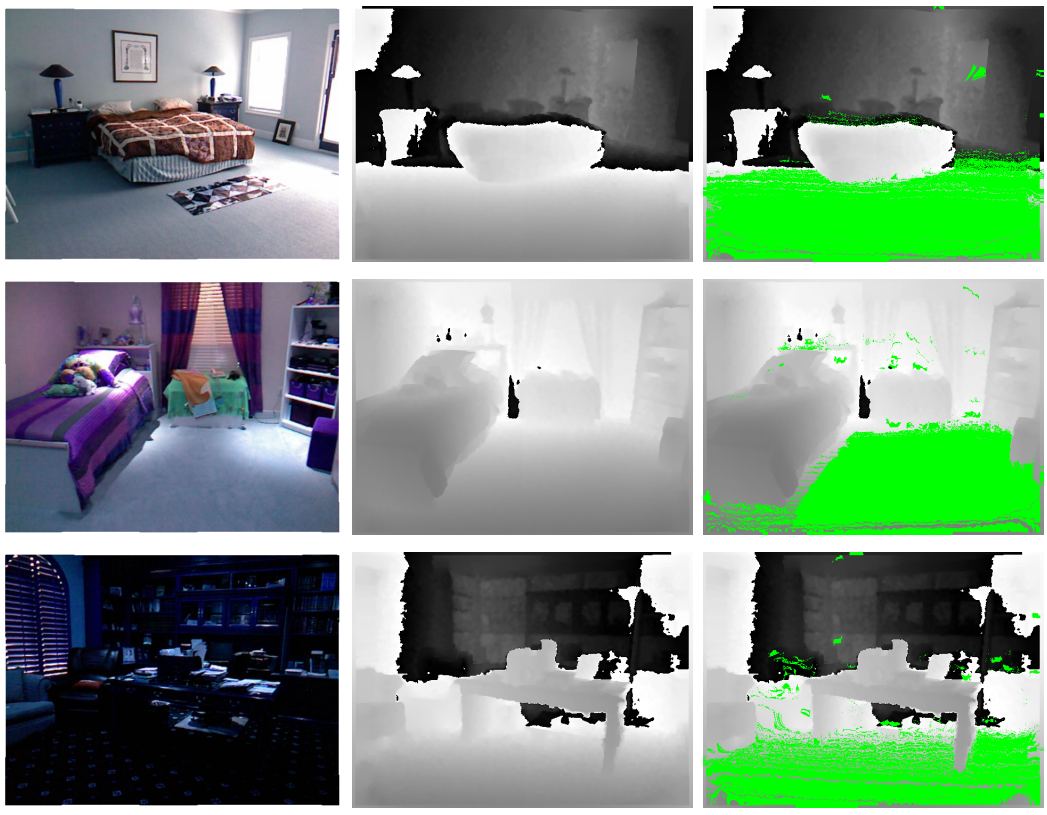

Figure 4.23: From left to right: sample color images from the NYU Depth V2 [2], depth image, ground pixels in green color. From top to bottom: Case of high, intermediate and of low score.

Table 4.1: Performance evaluation of our proposed algorithm using NYU Depth dataset V2 [2].

	Accuracy	Recall	Precision	F-measure
Min	85.23%	69.99%	68.30%	78.26%
Max	95.34%	95.42%	96.19%	91.35%
Mean	**91.84%**	84.21%	82.49%	83.05%
Ruiqi and Derek [121]	80.3%	-	-	-

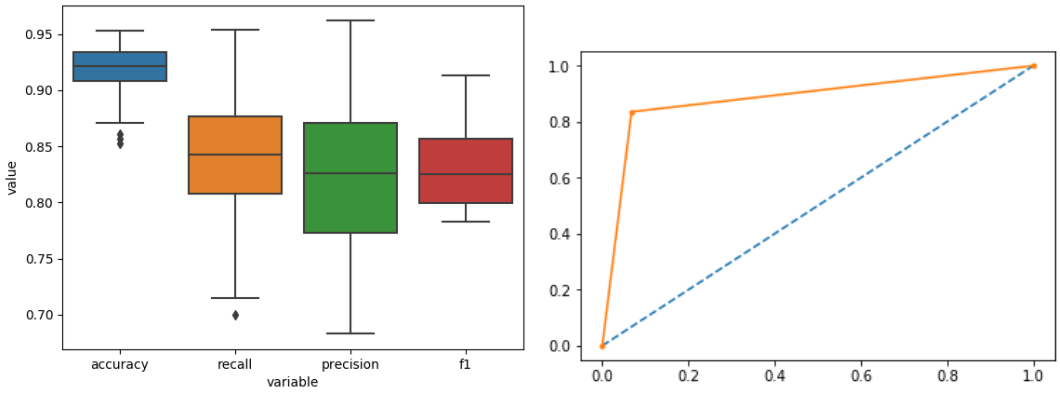

Figure 4.24: (Left) Values distribution for different metrics: The DCGD algorithm performs well for the majority of scenes (exceed 83%) in terms of the computed evaluation metrics. (Right) ROC curve: DCGD performs well in both sensitivity and specificity.

DCGD algorithm performs well for the majority of scenes (exceed 83%) in terms of the computed evaluation metrics (Figure 4.24 (left)) and in terms of sensitivity and specificity (Figure 4.24 (right)). In addition, the system confuses ground and non-ground pixels with a small value that does not exceed 1.6 (Figure 4.25).

4.5.6 Planes detection

For experiments purposes, we have taken two frames: with only one obstacle and with two obstacles parallel to floor (Figure 4.26). Note that by using basic RANSAC, some outliers persist: points that do not belong to obstacles, but they were considered as part of the detected planes as seen in Figure 4.26 surrounded by red circles. Fortunately, the applied statistical outlier removal filter, implemented by Point Cloud

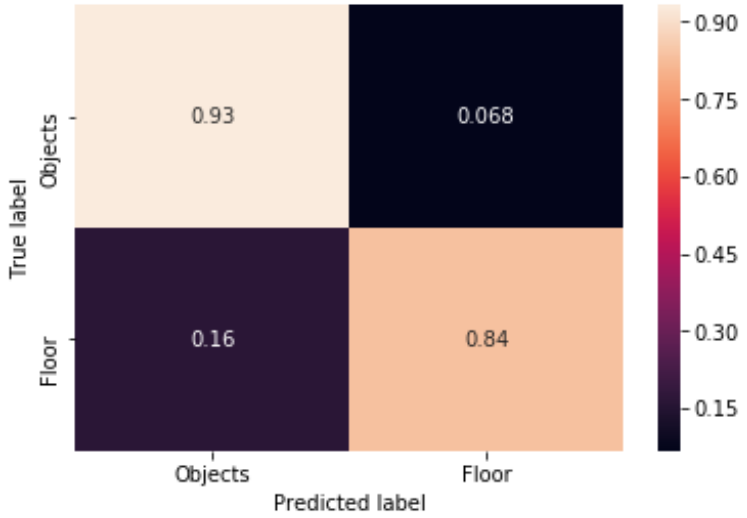

Figure 4.25: Confusion matrix: DCGD is very efficient in ground detection with confusion not exceeding 2.5%.

Table 4.2: Table of AE between respectively the computed radius and height, and the radius and height in real world in millimeters.

	Radius	Height	Radius AE	Height AE
Scene 1	298	197	6	38
Scene 2, Object 1	282	264	22	29
Scene 2, object 2	303	193	31	33
		MAE	45.988	35.306

Library (PCL) [122], reduced the noise and thus these latter do not affect significantly the plane characteristics computation such as the height and the radius. The table was not detected as plane parallel to the ground, it was detected in fact, as perpendicular plane since the table's perpendicular area is larger than the parallel one.

In order to compute how much the obtained characteristics are near to the real-world measurements, we have taken obstacles' measurements and then compared them with the computed characteristics as seen in table 4.2. The MAE does not exceed $46mm$ for both characteristics, this latter is generally insignificant regarding our proposed labeling.

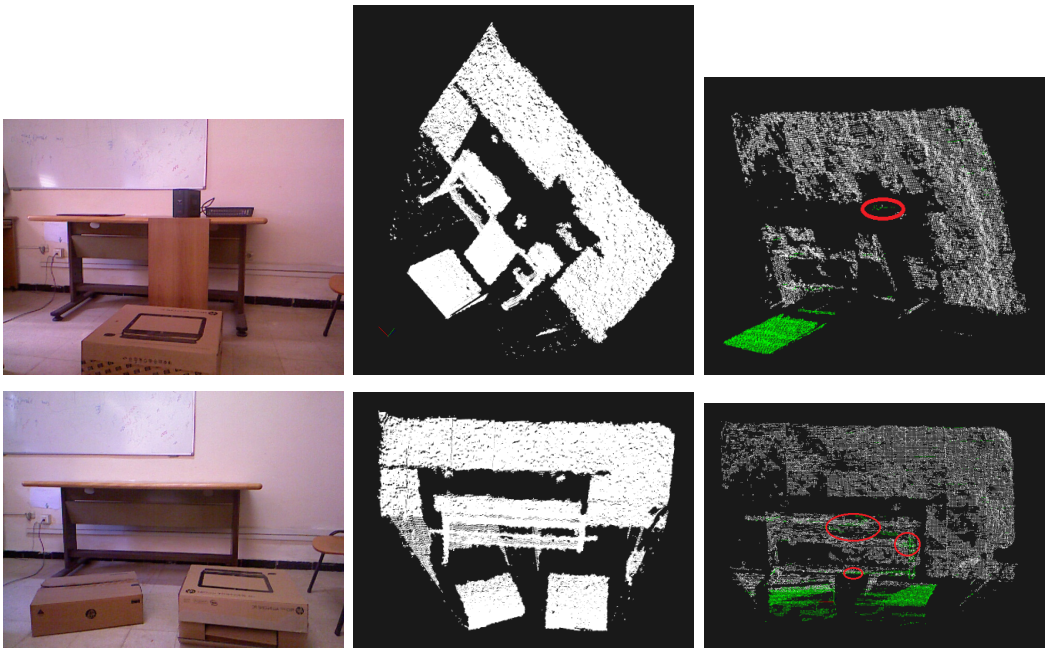

Figure 4.26: From left column to right: color image, occupied space point cloud, planes parallel to the ground in green color for one plane (first row) and two planes (second row) detection.

As for running time evaluation, using scenes from GDIS dataset, we evaluated each step: the ground detection, the point cloud segmentation, the geometric features computing and the point cloud classification steps. For each scene, we computed the running time of each step and then computed their mean. On average, the ground detection algorithm takes 1.143 seconds, the point cloud segmentation algorithm takes 2.030 seconds, the geometric features computing and the point cloud classification modules take 2.007×10^{-6} seconds for each segment. For the ground detection, the running time is related to the number of objects in the scene since the algorithm eliminates recursively the depth cuts representing the objects. It is related to the point cloud size for the segmentation step: the larger the point cloud, the greater the execution time and vice versa. The running time of computing the geometric properties and the classes is negligible.

4.6 Discussion

BASISR is a novel output device that is proposed in order to serve as an output device that generates 3D cues and representations. It has the shape of the view field of a single depth camera. It allows to have an impression of the surroundings in the user's hand. It can generate more complicated representations as we will show in the next chapter. However, the generated output is mapped manually on the device since we still working on the electronic layer that will make the output generation automated.

The DCGD algorithm is also developed to detect efficiently the ground from a depth image. It scored a high accuracy in both NYU Depth dataset V2, a public dataset and GDIS, our local dataset.

A system for scene labeling was proposed. It is based on plane detection that represent objects/obstacles. The detected planes are classified regarding their geometric features that are mapped as cylinders on the proposed 3D interface, BASISR (see Figure 4.27). This classification is informative and it is simple to understand and to interpret the 3D interface in order to interact with the scene. However, the classification and the labeling modules only provide the geometric features of a given object. In other terms, they do not take in consideration the objects' shapes and their nature. In addition, unlike the first system, this system did not map objects / obstacles like the wall, for example, since it was not detected as a horizontal plane.

4.7 Conclusion

In this chapter, we introduced our proposed 3D interface for human-scene interaction. This is our main contribution. In order to provide the visually impaired with a concise and an informative output that can be exploitable by touch, we designed

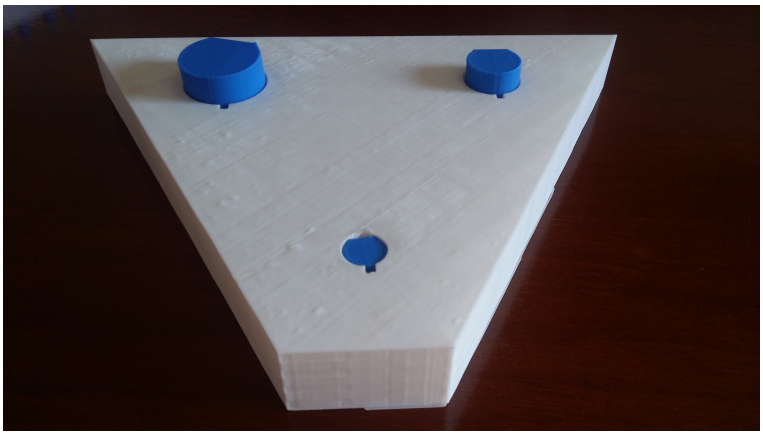

Figure 4.27: 3D printing of the proposed device. The device with three generated shapes coding the content of the perceived 3D scene using a depth camera.

two systems that can be used for navigation and scene understanding for human-scene interaction. However, due to their limitations when the user needs more details to perform more difficult tasks and/or to have more details about their surroundings, in the next chapter, we will propose a more informative scene labeling that also takes into consideration the object's shape and their nature. We will also show how to use BASISR for a better human-scene interaction.

Chapter 5

Scene Semantic Labeling for Human-Scene Interaction

5.1 Introduction

In the previous chapter, we introduced our designed 3D interface named BASISR. This latter was used in two different scene semantic labeling systems: a basic system that directly maps the occupied space on the device and an improved system that is based on plane detection. We also proposed the DCGD algorithm that will be also used for ground detection in this system.

In this chapter, we will present our system based on object classification. We propose two approaches: the first approach concerns object classification and the second approach concerns object labeling. The proposed labels are mapped on our 3D interface. They are easy to understand, learn and remember as we will show later. For this purpose, in Section 5.2, we will overview the system architecture. After that we will present its different modules, namely: point cloud classification and scene semantic labeling in Section 5.3, and Section 5.4 respectively. The system components are validated in Section 5.5 and the results are discussed in Section 5.6.

5.2 The system overview

The proposed system for human-scene interaction is based on point cloud processing and includes four main steps: the first step consists of filtering the input and reducing noise. This latter can sometimes be ignored when the processing techniques are robust to noise such as BDSCAN algorithm and deep neural networks. The second step consists of detecting the ground or the free space. After that, according to the system application, the free space or/and the occupied space is/are processed. In our case, the occupied space is extracted, segmented and then classified. The last step represents the output generation.

This system (Figure 1.2), that is for human-scene interaction, takes a single depth frame captured by a head-mounted depth camera as an input. After detecting the ground using the DCGD algorithm, the occupied space is extracted and segmented. Each computed segment is then fed to the feature extraction to perform object classification. The feature extraction module computes geometric features (such as the object's height) and the segment nature. After that, the provided class and features are used to generate semantic labels. Finally, these labels are mapped on the 3D interface to be exploited by the user. The system can be adapted to use an RGB camera as the input sensor; however, one more step that consists of estimating the depth image will be required.

5.3 Point cloud classification

Before the point cloud classification step, a segmentation step is required as it will be described in Section 5.3.1. After segmenting the occupied space, each segment is then classified. The classification is performed according to the nature of the objects

and their geometric properties as we will see in Section 5.3.2. The conception of this latter is then presented in Section 5.3.3.

5.3.1 Point cloud segmentation

In this system, we are interested in salient objects, thus coarse segmentation is needed. In case of point cloud coarse segmentation, the DBSCAN is more suitable as shown by Figure 5.1. Indeed, unlike K-means and Mean-shift that are centroid based algorithms (the clusters are shaped like circles around the centroids) and require the number of clusters, DBSCAN algorithm is based on density and does not require the number of clusters as input parameter.

To reduce the temporal complexity, we applied a down sampling on the input point cloud. A pass-through filter can be applied for noise removal; its parameters are fixed according to the characteristics of the depth camera. In our case, we only considered points with a depth between $800mm$ and $4000mm$. After the segmentation step, each segment is injected into the feature extraction and classification modules.

5.3.2 Point cloud classification: approach

Generally, object classification is achieved by considering a set of objects or by grouping them into different groups. In [64], they classified objects into three classes: fixed objects, rearrangeable objects and transit objects. The fixed objects' class includes objects that are not rearrangeable, they are permanent and/or hard to uninstall such as stairs and elevators. The rearrangeable objects class includes objects that can be moved more or less easily. They can be small, medium such as tables and chairs; or larger such as dressers and refrigerators. As for the transit objects class, it represents self-moving objects or objects that are created by schedules. However, this approach

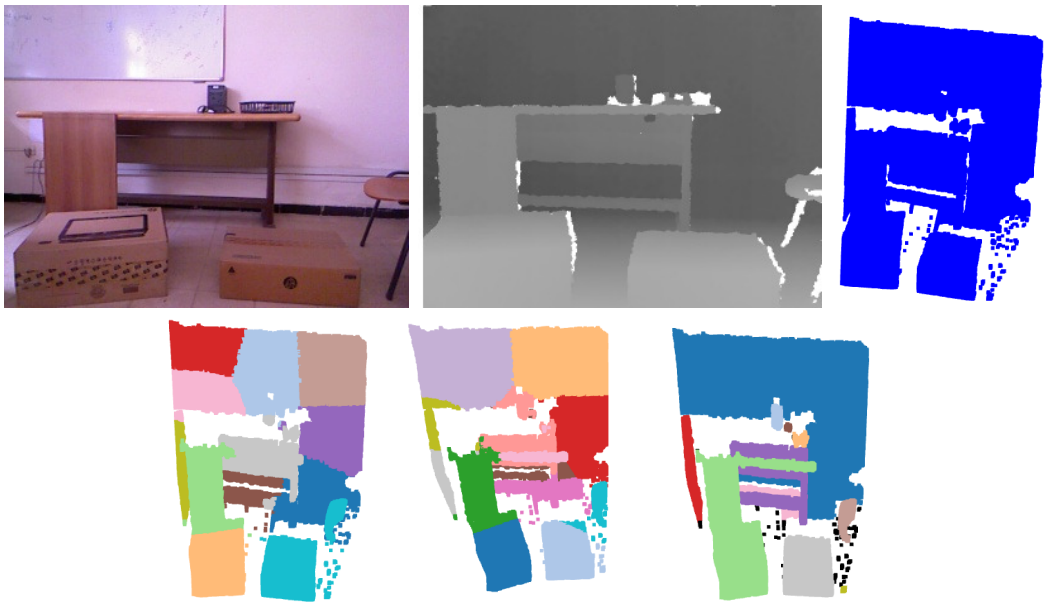

Figure 5.1: (Top) From left to right: color image, depth image and occupied space's point cloud. (Bottom) From left to right: K-means, Mean-shift and DBSCAN. The DBSCAN (the black points represent outliers).

is not suitable for semantic scene labeling for human-scene interaction, especially when the visually impaired and blind people are involved.

We propose a new approach for object classification, we design classes which are related to the semantic of the selected objects. For this purpose, in Section 5.3.2, we present our designed classes that relies on the selected objects. In Section 5.3.2, we show how to combine the object classification and the classification based on geometric features for an informative point cloud classification.

Object classification

Our aim is to provide an object classification that is more informative, especially in the context of assistive systems. In order to cover a significant number of classes in a simple way, we consider salient objects (large and medium objects) and regroup them into 7 semantic classes that include 16 object classes, namely chairs, beds, sofas, benches, stools, tables, desks, night-stands, dressers, wardrobes, shelves, bathtubs,

toilets, windows, doors and stairs. These classes do not only contain semantic meaning, but the objects belonging to the same class have also similar geometric shape while preserving some characteristics.

The first class represents objects that we sit on, it includes chairs, stools, beds, sofas and benches. The second class represents objects that we put something on, it includes tables, night-stands and desks. The third class represents objects that we put, hide or arrange something in, it includes dressers, wardrobes and shelves. As for the fourth, the fifth and sixth classes, they represent bathtubs and toilets; windows and doors respectively. The last class represents stairs and since the stairs can be dangerous, making the difference between the stairs leading to upstairs and stairs leading to downstairs is important.

This approach requires the object in its 2D or point cloud representation as input. In order to obtain the input's class, deep neural networks can be used as we will explain in Section 5.3.3.

Point cloud classification

We combined the two classification methods, i.e. the designed object classification and the classification using the geometric features to provide rich information. For each point cloud, we provide its object class, its height class, its occupied area class and its location. When the deep learning model predicts the object class with low probability, only the second classification is maintained. In this way, we are sure to provide an accurate description even if the deep learning model fails to predict the right class.

In addition, combining these classes allows the distinction between objects belonging to the same class. Chairs, beds, sofas and benches are objects that we sit on; however, they are different in their forms and occupied area: beds are generally a large object (3rd class regarding the occupied area), chairs have small surface (1st class regarding

the occupied area) and benches and sofas are medium objects (2nd class regarding the occupied area). Tables, night-stands and desks represent the second class; however, they differ in their forms and locations: night-stands are usually small whereas desks and tables are larger. In addition, unlike desks, tables are usually surrounded by chairs. Dressers and shelves are all objects that we arrange something in; however, they have different heights and occupied areas: dressers are represented by the 3rd class regarding the height, but shelves are usually medium objects.

Figure 5.2 and Figure 5.3 represent the layouts of two different scenes. In these latter, the objects are represented by their class, their geometric features and their positioning in the scene; however, using these features combined with their location, we can deduce their nature. In Figure 5.2, the larger chair is a bed, the medium chair is a sofa, the rest are chairs; the small tables that are next to the bed are night-stands, the table that is alongside the wall is a desk (only a chair is next to it), the rest are tables (surrounded by chairs) and the dresser is near the night stand. In Figure 5.3, the small objects that belong to the class 'chairs' are chairs; since the tables are at most surrounded by two chairs, they probably are desks and the dressers represent shelves.

Combining the two methods allow us to design only 7 labels to represent 16 classes. This latter will facilitate users' interaction with the scene by learning only 7 classes instead of 16 different classes. In addition, even if an object is not included in the 7 classes, it will be labeled using its geometric features.

5.3.3 Point cloud classification: conception

After defining our classes, we now explain how they are obtained. This module includes 3 different sub-modules. The first sub-module computes the point cloud class based on classifying its global features. The second sub-module computes the point cloud geometric features. As for the last sub-module, it combines the output

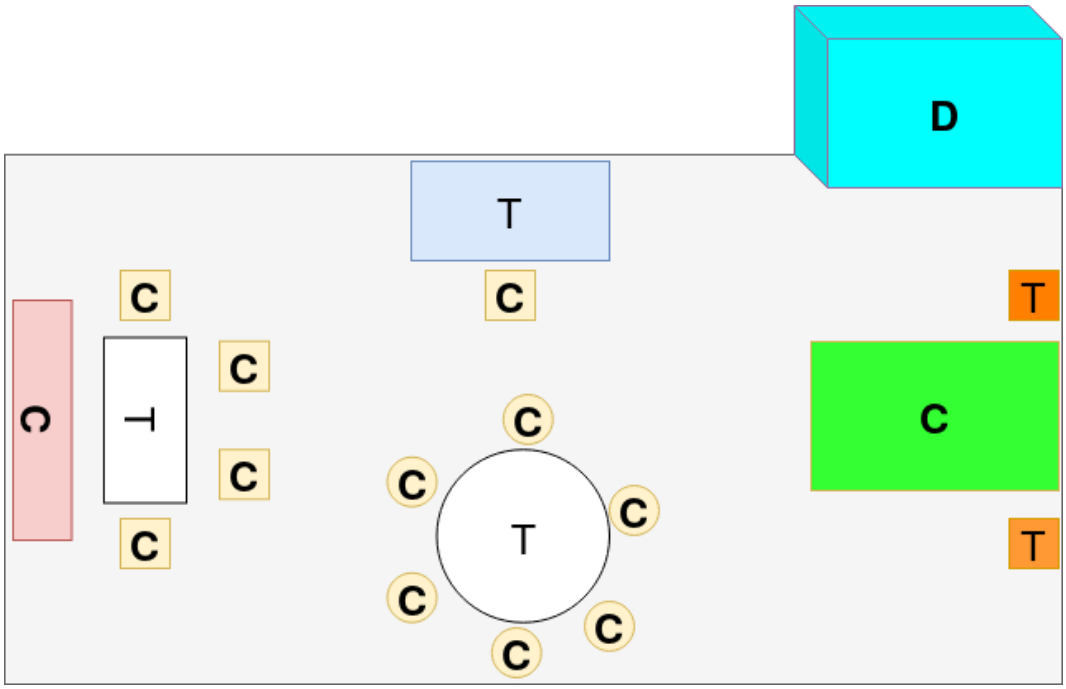

Figure 5.2: Example of a scene representing a room. The objects are represented by their class, their geometric features and their positioning in the scene (C, T and D to represent the class chairs, tables and dressers respectively; and blue, yellow, purple, green, red, orange, white and cyan represent the object desks, chairs, shelves, beds, sofa, night-stands, tables and dressers respectively).

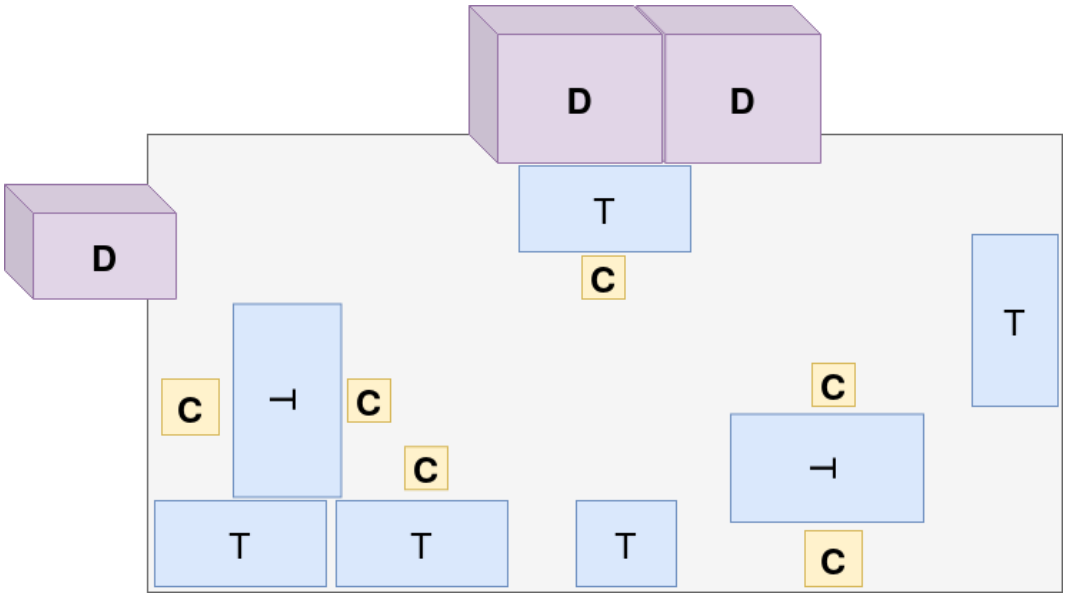

Figure 5.3: Example of a scene representing an office (my supervisor's office). The objects are represented by their class, their geometric features and their positioning in the scene (C, T and D to represent the class chairs, tables and dressers respectively; and blue, yellow and purple represent the object desks, chairs and shelves respectively).

of these sub-modules in order to provide the final classes and provide them for the semantic labeling module. For the first sub-module, we tested two categories of classification architectures: multi-view based models and raw point cloud based models.

For the first category, in order to compute the input, we construct an organized point cloud from the depth image that contains only a single object after the segmentation step. An organized point cloud is a tensor $3 \times w \times h$ where 3 is the number of channels, w and h represents the width and the height of the bounding box after the scaling step. The first channel, the second channel and the third channel hold the X values, the Y values and the Z values respectively. The scaling step is essential since the input's shape of a given neural network is fixed and cannot be flexible for each entry. We used nearest neighbor interpolation in the scaling step. Feeding with the organized point cloud tensor will allow the neural network to learn the

relation between the adjacent points on the three axes using convolutions. Finally, we apply data normalization in $[-1, 1]$ interval. Since depth images are the input of our system, we investigated the behavior of different architectures that are designed for image classification and then, we selected the most accurate model. To do so, we trained the networks until they reach 100% as accuracy on training set or the training does not improve after some epochs. After that, we design a new architecture based on the selected neural network. The designed network has to be lightweight and accurate. Instead of starting training from scratch, we first took a pretrained model using data from the ImageNet challenge [123] and then, we opted for the transfer learning approach. The obtained results will be presented and discussed in Section 5.5.

As for the second category, to classify a given point cloud, we trained the Pointnet network [13]. The limitation of this model (as mentioned in Section 3.7.3) will not highly affect our system since we only consider salient objects (for the time being). Note that, in this work, we are not designing a novel architecture. However, using our approach of merging the objects' classes that are geometrically similar and have the same function increased the accuracy by 5% on the test set. The obtained results are reported, compared and discussed in the experiments Section 5.5.

Regarding the second sub-module, we followed the same step as in the first prototype except for the computation of the heights. Since we consider irregular segments and not planes, we computed the height as the height of the 90th percentile of the points' $y - coordinate$. We chose the 90th percentile instead of the 100th to avoid potential outliers. The third sub-module, combines the obtained classes and transmits them to the semantic labeling module.

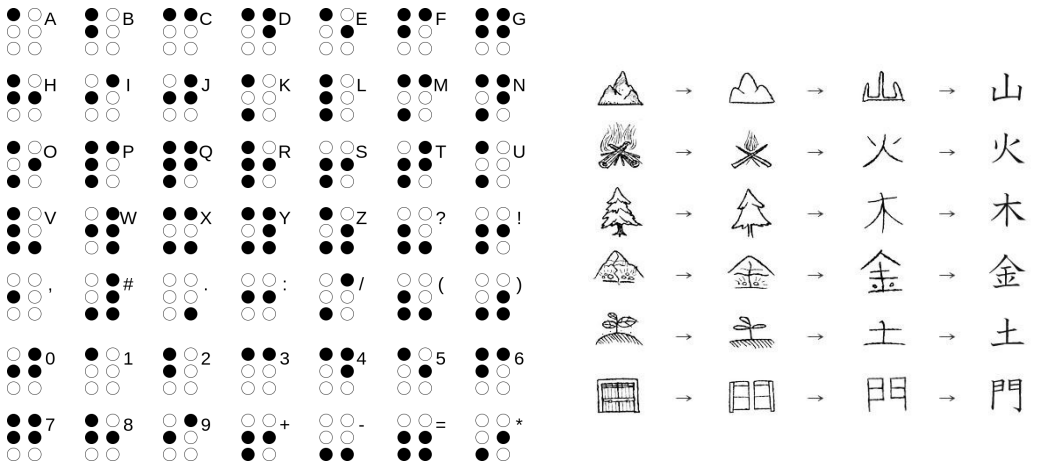

Figure 5.4: (Left) The Braille system. (Right) Kanji, the Japanese writing system.

5.4 Object semantic labeling

Regarding the point cloud classification that we designed, we propose an improved object semantic labeling which takes into consideration the nature of the objects. It is inspired by the **Braille system** and **Kanji** (the Japanese writing system) (Figure 5.4). An alphabet in the Braille system is represented by a cell that is provided with raised dots. Each cell contains at most six raised dots. **Kanji** is a Japanese writing system that is inspired by logographic Chinese characters. Some Kanji letters that represent some objects are driven from nature, i.e. these objects' shape in the real world; it's the case for 'mountain' as shown in Figure 5.4 (right) first line.

In order to draw our illustrative semantic labeling, we designed cells with at most 25 raised dots. The cell's shape can be revised depending on the precision of the 3D interface: if the 3D interface is rich in pins, we can design cells with more raised dots. The cells must be chosen such that the shape can be touched only by a single finger to avoid ambiguity.

On the other hand, the illustrative labeling is derived from the object's shape in the real world. For each semantic class, we chose the object that is the most close to the class' meaning: we chose a chair, a table, a dresser, a bathtub, a window and

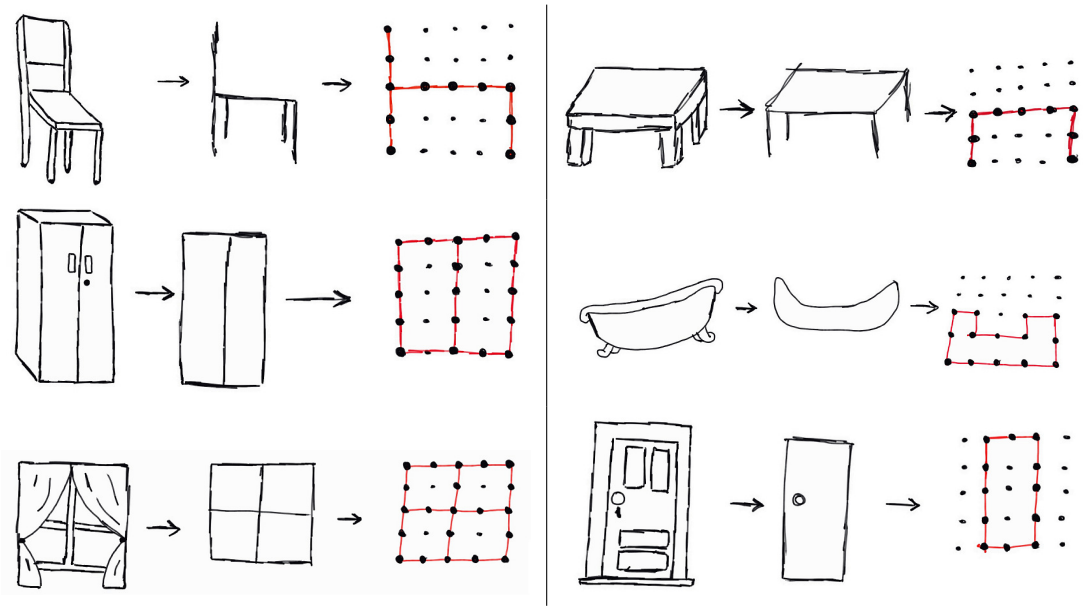

Figure 5.5: Our proposed semantic labeling from the first class to the last class, respectively (From top to down). We first drew the selected objects in the real world (Left) and then, we derived shapes recursively until we obtained the current semantic labeling (From left to right).

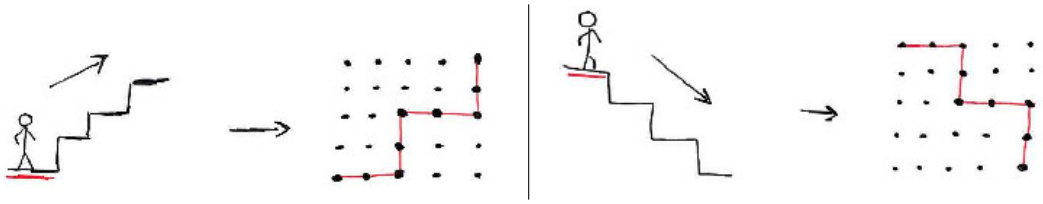

Figure 5.6: The proposed semantic labeling for stairs. Stairs leading to upstairs and leading to downstairs, their chosen labels, their positioning in the synthesis area.

a door to represent the proposed classes in Section 5.3.2 respectively as shown in Figure 5.5. To ensure the user's safety, we provide two different labels for the stairs leading to upstairs and stairs leading to downstairs as shown in Figure 5.6. In this way, by touch, the user will understand if he is about to climb the stairs or about to go downstairs.

The described labeling is used alongside with the geometric features semantic labeling to enrich the scene description. The reason we combined the two labels instead of creating an illustrative label for each object, is to reduce the number of possible

illustrative labels since it can be deducted from its height and its occupied area as explained in Section 5.3.2. Note that although the labels are inspired by Braille and Kanji systems, understanding and memorizing these labels do not require the user to be comfortable with them.

Once the geometric features and the classes are computed, we generate their semantic labeling on the BASISR device at the segments locations. We combine the proposed semantic labeling, but instead of mapping cylinders that represent the geometric features of each segment, we directly map the segments' convex-hulls. The advantage of this latter is that, the user will have a clear impression of the object's shape and pose contrary to cylinders that only represent the geometric features. If the semantic class of a given segment is predicted with high probability, we map its label at the center of the convex-hull. Otherwise, we only map the convex-hull of the segment.

Let $b_1(x, y), b_2(x, y), .., b_n(x, y)$ be the points defining the box encompassing the object. The mapped points $b'_1(u, v), b'_2(u, v), .., b'_n(u, v)$ are located on the area. All pins in the area defined by the points $b'_i, i = 1..n$ are set to level 2. The pins associated to the label of the object are set to their associated level 3, 4 or 5 at the barycenter as indicated by Figure 5.7. We assigned the grey color, the green color, the red yellow color and the red color to represent objects with level 2 (ground), 3, 4 and 5 respectively.

5.5 Validation

In this chapter, we will evaluate and test the deep neural network models that are used for point cloud classification (in Section 5.5.2) using public and local datasets that will be presented in Section 5.5.1. We will show also, how the output is generated and synthesized in Section 5.5.3. Our obtained results are reported, discussed and compared with the state-of-the-art systems.

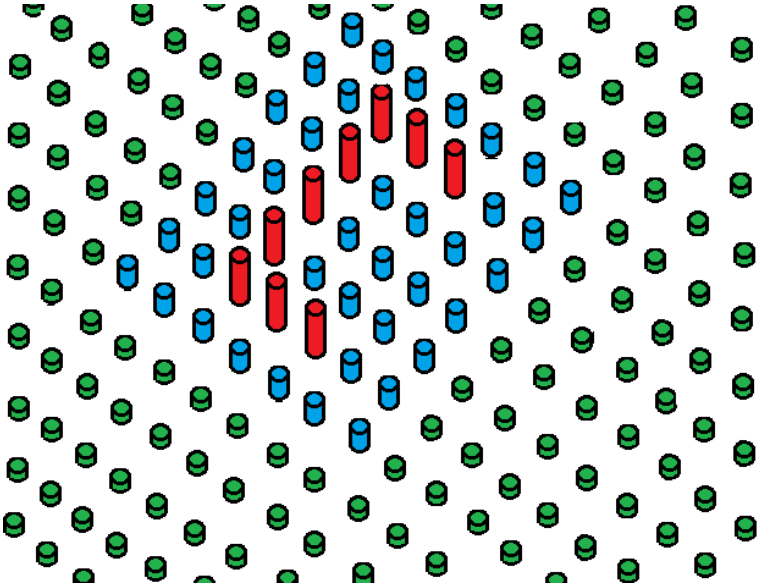

Figure 5.7: The convex hull and the positioning of the label (here for a table) on the area. Note that the pins of the label in red are at level 3 (corresponding to the height of the table in the scene). The pins of the convex area in blue are at level 2. The remaining pins in the synthesis area (in green) are set to level 1 corresponding to the ground.

In order to perform the training, we ran our models on a GPU provided by Google Colaboratory platform. After that, we executed our proposed system on a laptop having Intel Core $i5-7200U$ CPU ($2.50GHz \times 4$) and $4GB$ as processor and RAM respectively.

5.5.1 Datasets

In this section, we introduce the different datasets that will be used for validating and testing our models, namely: NYU Depth dataset V2 [2] that will be used to validate the ground detection algorithm, the dataset from the Reconstruction Meets Recognition Challenge (RMRC) challenge that will be used to train and validate the VGG-16 network, ModelNet40 dataset [14] that will be used to train and validate PointNet, and GDIS [15] that will be used to validate the ground detection algorithm, PointNet, the geometric feature extraction and the entire system.

Figure 5.8: Dataset samples: RMRC challenge (2014). From left to right: the RGB image, the preprocessed depth image and the image of labels.

Dataset from the RMRC challenge

To train the CNNs based models, similar to [112] and [14], we used the data from the RMRC challenge (Figure 5.8). Since the dataset for 3D classification is no longer available in RMRC challenge (2014), we construct our data from its Semantic Segmentation Challenge. The data was initially consisted of 1800 RGB-D images with dense classification (23 possible classes). For each image, we considered only regions that belong to our selected classes (i.e. chairs, beds, sofas, benches, tables, desks, dressers, night-stands, shelves, bathtubs, toilets, windows, doors and stairs). The classes *windows, doors* and *stairs* were not among the 23 proposed classes so, for the multi-view semantic classification, we only consider the remaining classes. After the selection process, we obtained 3494 new images each of which represents one object only (a single segment). Table 5.1 represents the number of instances for each of the seven classes. We used the terms *chairs, tables* and *dressers* to represent the first three classes respectively. We also considered 35% of the data as the validation set.

Table 5.1: The number of instances for each class.

Chairs	Tables	Dressers	Bath-tubs	Toilets
1595	934	786	80	99

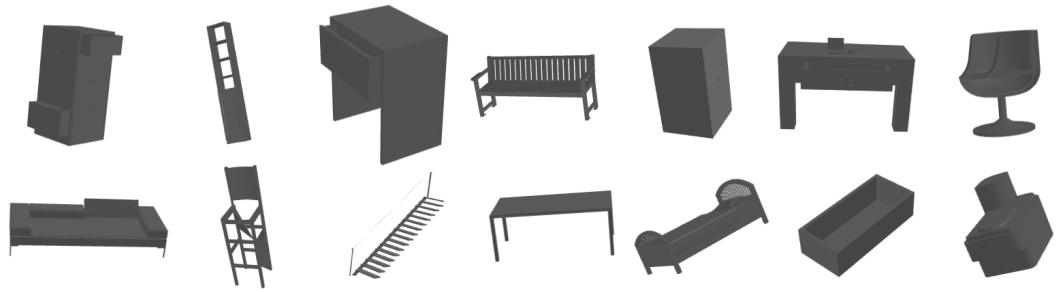

Figure 5.9: Dataset samples: Modelnet40[14].

ModelNet40

As to train the PointNet model, we extracted our 16 selected classes from ModelNet40 [14] (Figure 5.9), a public dataset which consists of 40 classes of CAD models. Note that we excluded doors and windows since they are represented by holes and in the real world and they can be confused with noise or walls when they are closed. ModelNet40 originally includes 12311 models among which 2468 are used for testing. Our constructed dataset includes 5560 models among which 988 are used for testing.

GDIS dataset

In addition, we used GDIS dataset [15] (Figure 5.10), our local dataset, for additional experiments. GDIS dataset was initially constructed for the ground detection task, but we can also consider some scenes for point cloud classification and the validation of the entire system (from the ground detection to the semantic labeling).

5.5.2 Point cloud classification

In this section, we evaluate the point cloud classification, the third module to execute in the system pipeline. In this section, we validate the Multi-view based model and

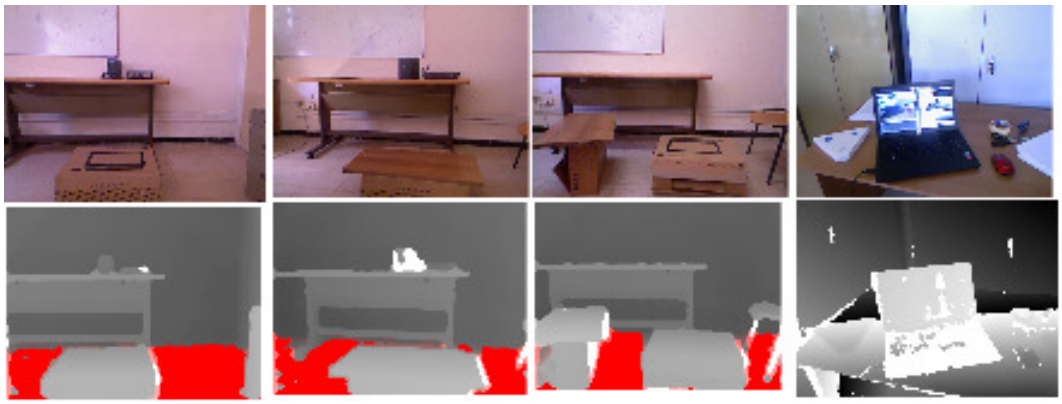

Figure 5.10: Dataset samples: GDIS [15]. The red color represents the detected ground.

the point cloud based model. Finally, we compare the obtained results with the state of the art.

Multi-view based classification

Several networks were proposed for object classification among which we cite: AlexNet [124], VGG16 [117], GoogleLeNet [19], ResNet [125] and MobileNet [126]. We took the pretrained model of each of the cited networks and we replaced the last dense layer with a dense layer with 5 perceptrons (we have 5 classes). We stop the training if the accuracy does not progress after executing at least 30 epochs or if the model reaches 100% as training accuracy.

Figure 5.11, Figure 5.12 and Figure 5.13 display the loss function and the accuracy on the training and the validation set respectively. Note that in this step, we did not apply any data augmentation. We wanted to study the architectures' performance without additional preprocessing.

The Figures 5.11, 5.12, 5.13, show an overfitting for all the models. In addition, the models' performance is not stable during training except for AlexNet model. This is due to the models' parameters, the nature of data, its complexity and the dataset

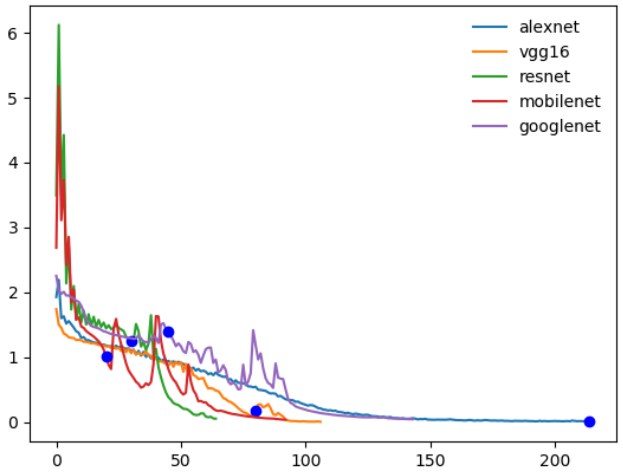

Figure 5.11: Comparison between the loss function of each tested model on training dataset. The blue marker represents the epoch where the model reached the highest accuracy on validation set.

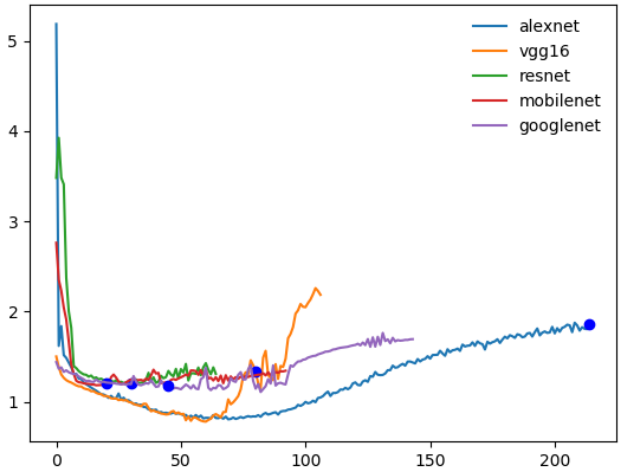

Figure 5.12: Comparison between the loss function of each tested model on training dataset. The blue marker represents the epoch where the model reached the highest accuracy on validation set.

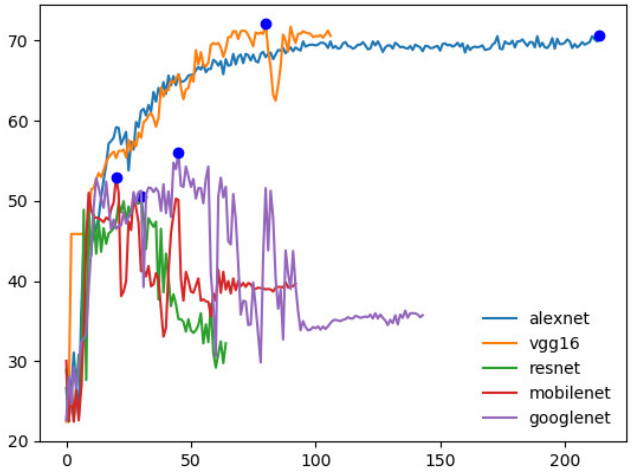

Figure 5.13: Comparison between the accuracy of each tested model on validation dataset. The blue marker represents the epoch where the model reached the highest accuracy on validation set.

size. The highest accuracy (the blue marker) on validation set is achieved at the same time as the highest accuracy on training set for only AlexNet model which showed smooth improvement during training; however, it was achieved while overfitting. The VGG-16 model achieved the highest accuracy (72.082%) on validation set.

Based on the obtained results, we built our model based on VGG-16 by reducing the number of hidden layers (see Figure 5.14). We continued training after applying the presented data preprocessing. The model achieved an accuracy equal to 72,81% on the validation test (Figure 5.15) and 94.85% on our GDIS dataset [15].

Figure 5.16 and Table 5.2 show the confusion matrix and the model's precision, recall and F-measure on each class respectively. The accuracy and the F-measure exceed 60% and 59% respectively for most classes. Although the model's accuracy on the class *'chair'* is the highest, its F-measure is the lowest. This is due to the confusion between this class and the remaining classes (Figure 5.16, column *'Chair'*). On the other hand, the class *'bath-tub'* has low accuracy, but a high F-measure. This latter is due to the fact that the model mistakes the class *'bath-tub'* for the other classes

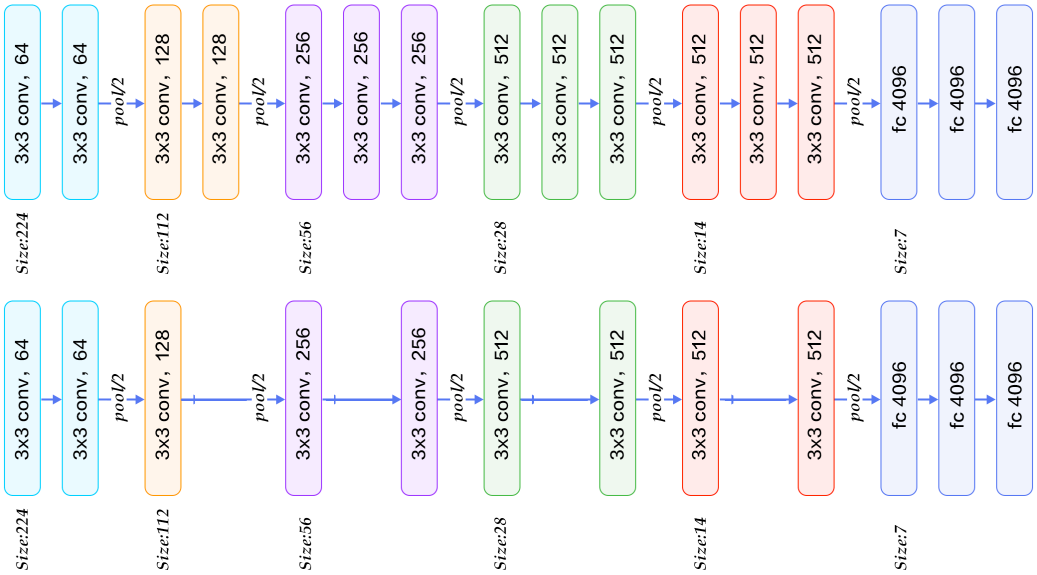

Figure 5.14: (Top) the VGG-16 architecture. (Bottom) The selected architecture: we removed the 4th, 6th, 9th and 12th layer.

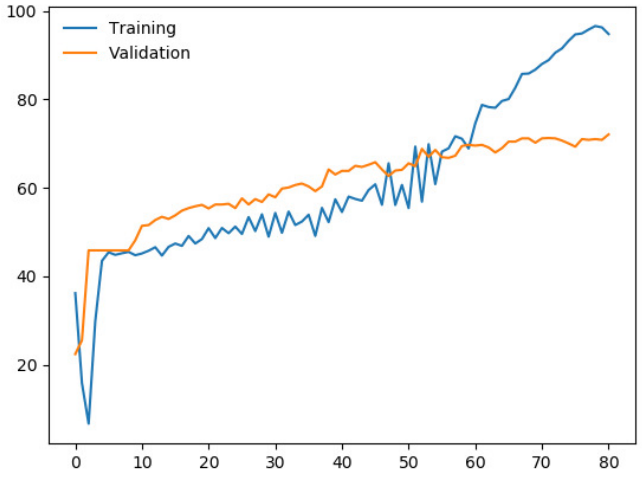

Figure 5.15: Accuracy evolution by epochs on training and test sets. Although the model couldn't generalize well, the accuracy improves on both sets.

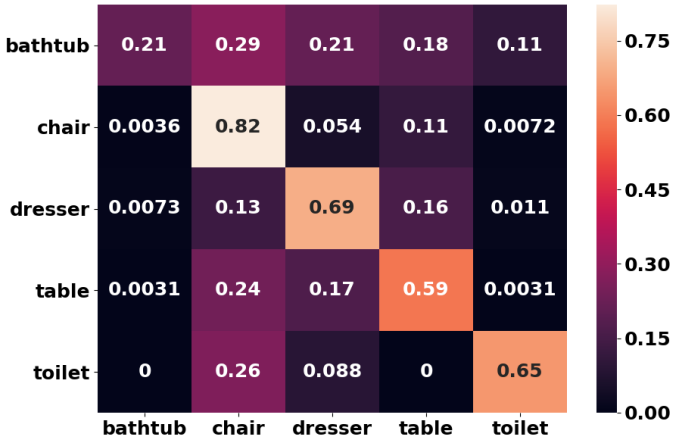

Figure 5.16: The confusion matrix of our selected model. The confusion matrix shows that there is some high confusion between objects having almost the same geometry: 26% of the toilet instances were confused with chairs and there is no confusion with tables and bathtubs.

Table 5.2: The validation metrics of the CNN network on the test set.

	Bath-tubs	Chairs	Dressers	Tables	Toilets
Precision	0.545	0.778	0.669	0.630	0.667
Recall	0.214	0.823	0.691	0.586	0.647
F-measure	0.308	0.8	0.68	0.607	0.657

(Figure 5.16, line *'Bath-tub'*), but it rarely mistakes the other classes for the class *'bath-tub'* (Figure 4.25, column *'Bath-tub'*). To visualize these latter, we display some samples from well classified instances and misclassified instances in Figure 5.17 and 5.18 respectively.

Point cloud based classification

Since PointNet is 3D raw data based, the CAD models are sampled into point clouds of 2048 points and normalized into a unit sphere. Regarding the experimental settings, we applied the same configurations as in [13]. As for data preprocessing, we, on the fly, applied random rotations around the yaw axis and jittered the points

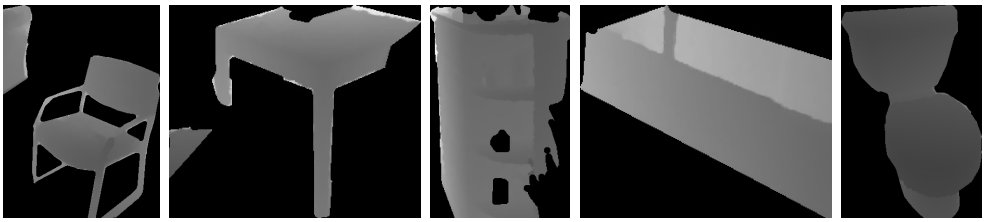

Figure 5.17: Sample of well classified instances. From left: Chair, Table, Dresser, Bath-tub and Toilet.

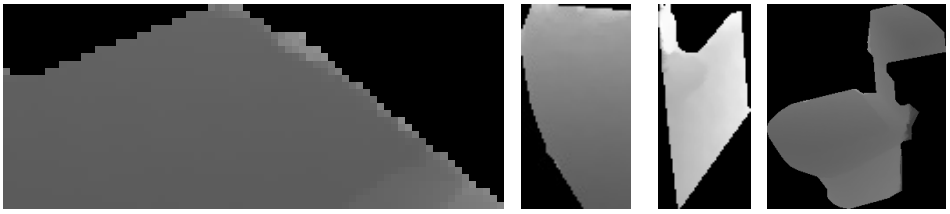

Figure 5.18: Sample of misclassified instances, the predicted class (class): Dresser (Chair), Chair (Table), Table (Bath-tub), Chair (Toilet)

position as described in [13]. PointNet achieved the state of the art with 89.2% accuracy in their original paper.

In this work, we trained our model on a sub-set of 14 (after excluding doors and windows) classes from ModelNet40. The model started to converge starting from the 50th epoch and reached 91.67% accuracy on the test set after 262 epochs (Figure 5.19 and Figure 5.20).

However, as shown by the confusion matrix (see Figure 5.21), there is a high confusion between the class 'wardrobe' and the classes 'bookshelf' and 'dresser' (about 20% and 10%, respectively), between 'table' and 'desk' (about 16%), 'stool' and 'chair' (about 15%). There is also a small confusion between the 'sofa' and the classes 'bench' and 'chair'. These confusions are due to the geometric resemblance between them.

This leads us to consider combining classes for the classification as explained earlier in Section 5.3.2. As the previous model, this model started to converge after the 50th epoch. The accuracy has been improved by 5% to reach 96.35% on the test set

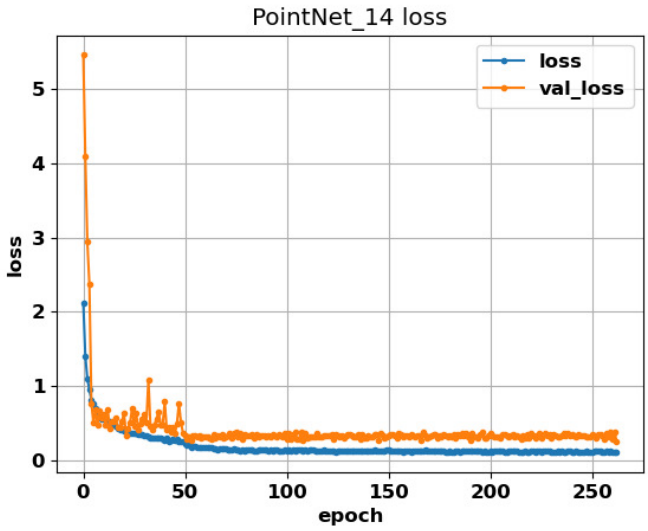

Figure 5.19: PointNet loss function: the model learned to generalize. Training PointNet with 14 classes; the model starts to converge from epochs 50

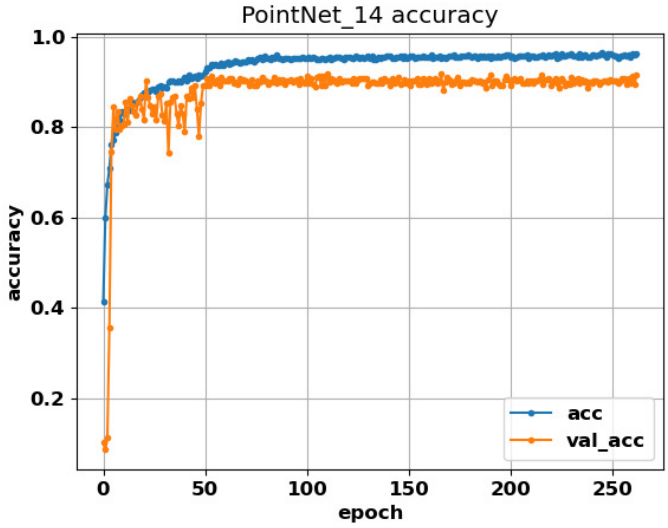

Figure 5.20: PointNet accuracy. Training PointNet with 14 classes; the model starts to converge from epochs 50 and reach 96.19% and 91.67% on train-set and test-set respectively.

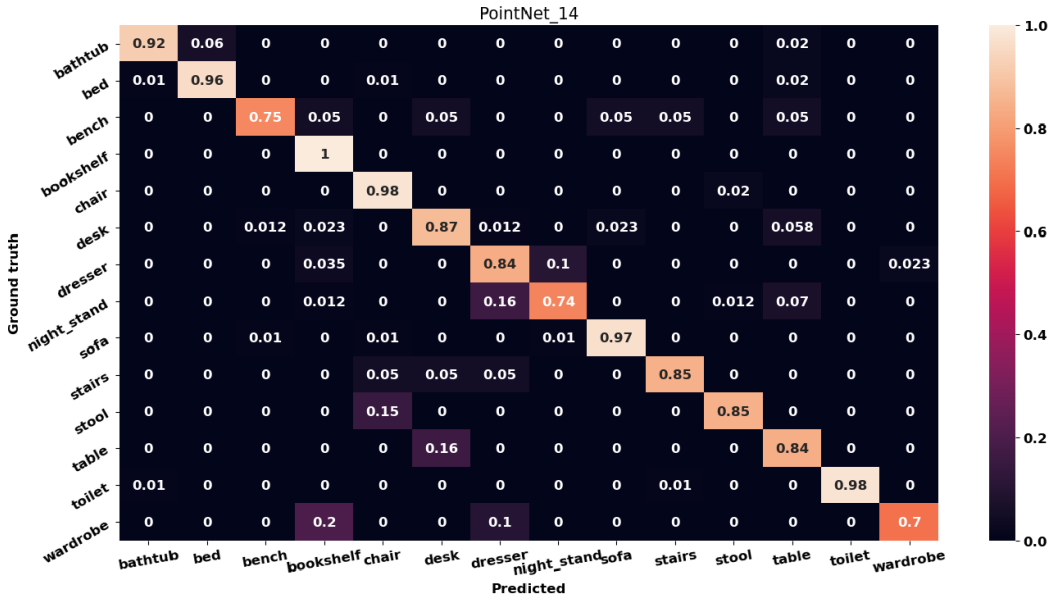

Figure 5.21: Confusion matrix: there is high confusion between the class 'wardrobe' and the classes 'bookshelf' and 'dresser' (about 20% and 10%, respectively), between the classes 'table' and 'desk' (about 16%), the classes 'stool' and 'chairs' (about 15%); and small confusion between the class 'sofa' and the classes 'bench' and 'chair'.

after only 226 epochs (Figure 5.22 and Figure 5.23). Thus, the described confusions have been eliminated (see Figure 5.24).

The confusion matrix (see Figure 5.24 (Left)) shows that the confusion between the different classes does not exceed 7% unless for stairs that 15% of them were confused as chairs. To visualize this latter, we plot the three worst classification for each class (Figure 5.25): the instances have similar geometry with the predicted class with a certain probability. However, the confusion remains small and the system predicts accurately with high recall and precision (see Figure 5.24 (Right)).

The model was trained using 3D CAD models. On real-world data (from GDIS dataset), the model predicts the right class when the input is noisy and slightly incomplete (see Figure 5.26); however, it fails otherwise. As shown in Figure 5.27 (column 2), the system has predicted incorrectly the class of the table that was poorly segmented (we only used background removal to simulate the cropped data).

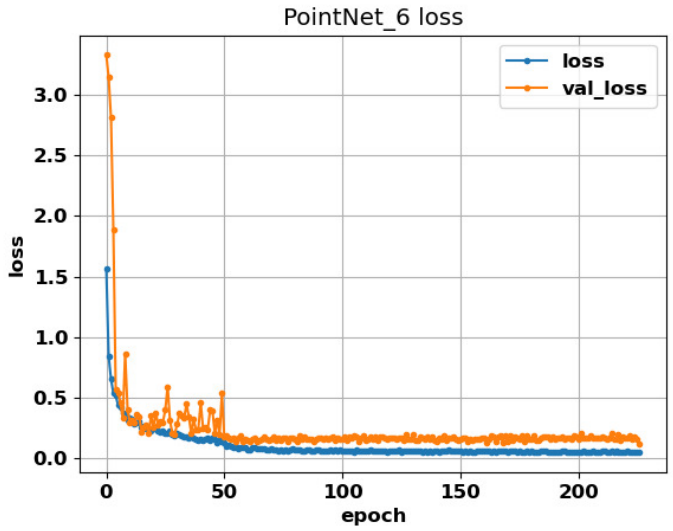

Figure 5.22: PointNet loss function. Training PointNet with 6 classes; the model starts to converge from epochs 50.

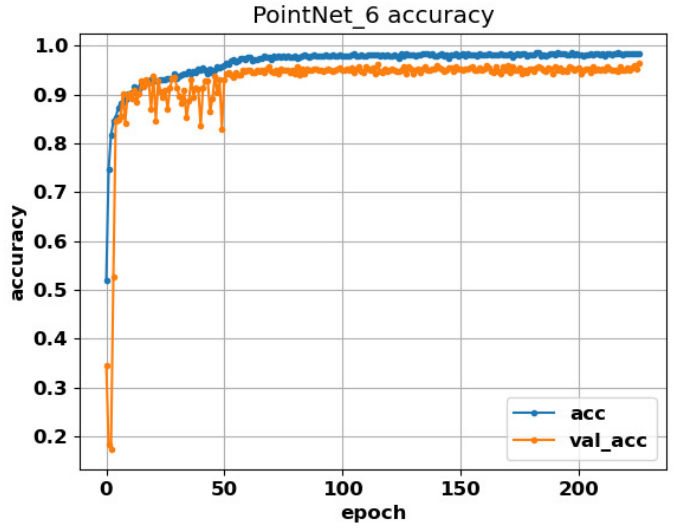

Figure 5.23: (Left) PointNet loss function. (Right) PointNet accuracy. Training PointNet with 6 classes; the model starts to converge from epochs 50 and reach 98.26% and 96.35% on train-set and test-set respectively.

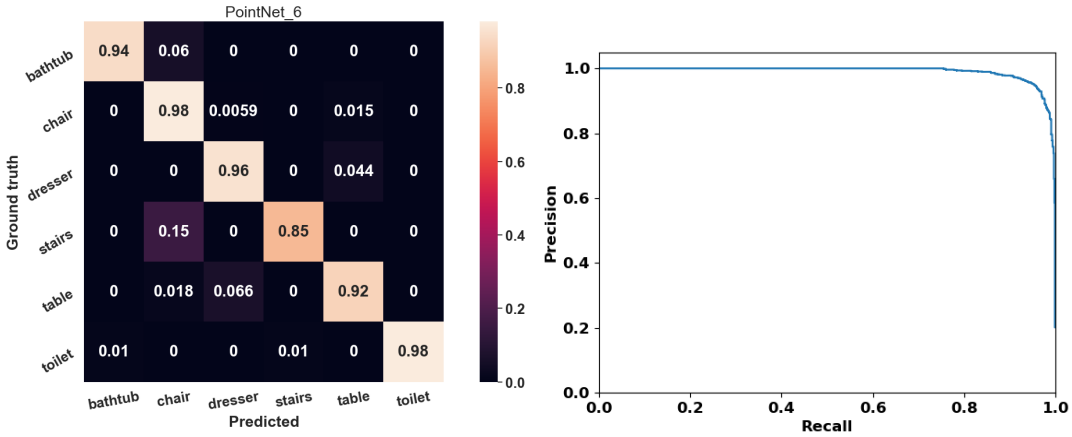

Figure 5.24: (Left) Confusion matrix: the confusion between objects does not exceed 6% unless for stairs that 15% of them were confused as chairs. Most of the objects are not confused with other objects (0% confusion). (Right) Recall-precision curve: the system predicts accurately with high recall and high precision).

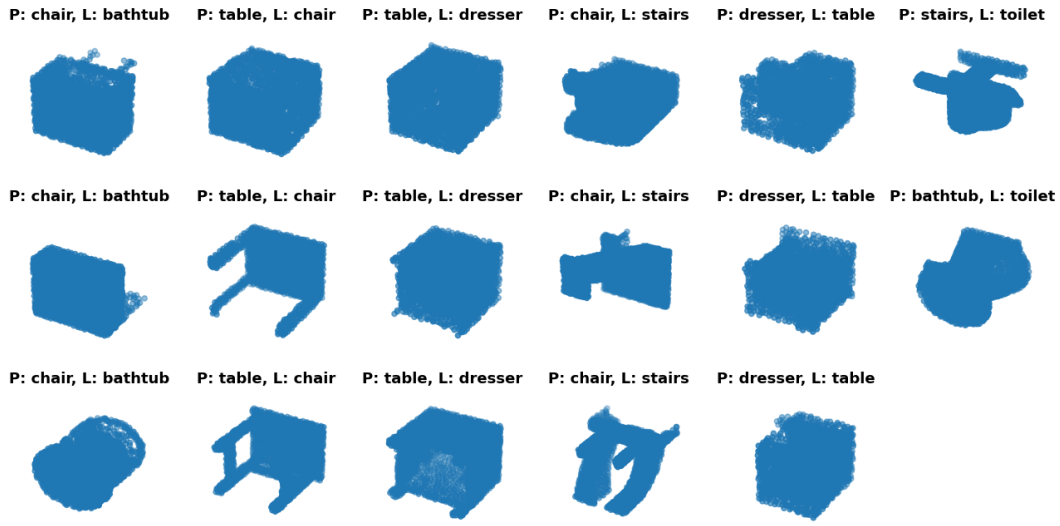

Figure 5.25: Worst 3 classification results. P and L stand for Predicted and Label respectively. There is a geometric resemblance of the input instance with the objects of the predicted class.

Figure 5.26: From left to right: the captured scene (chair, chair, chair), its associated depth image (the system input), the resulting segment. The system has predicted: chair, chair, chair.

To overcome this problem, we only consider the predicted class with high probability (higher than 0.95).

Comparison with state-of-the-art systems

Compared with the results of object classification based aid systems presented previously (Table 5.3), in the system [7], authors didn't mention the accuracy of their proposed depth-based classification algorithm. As for the system [6], the authors

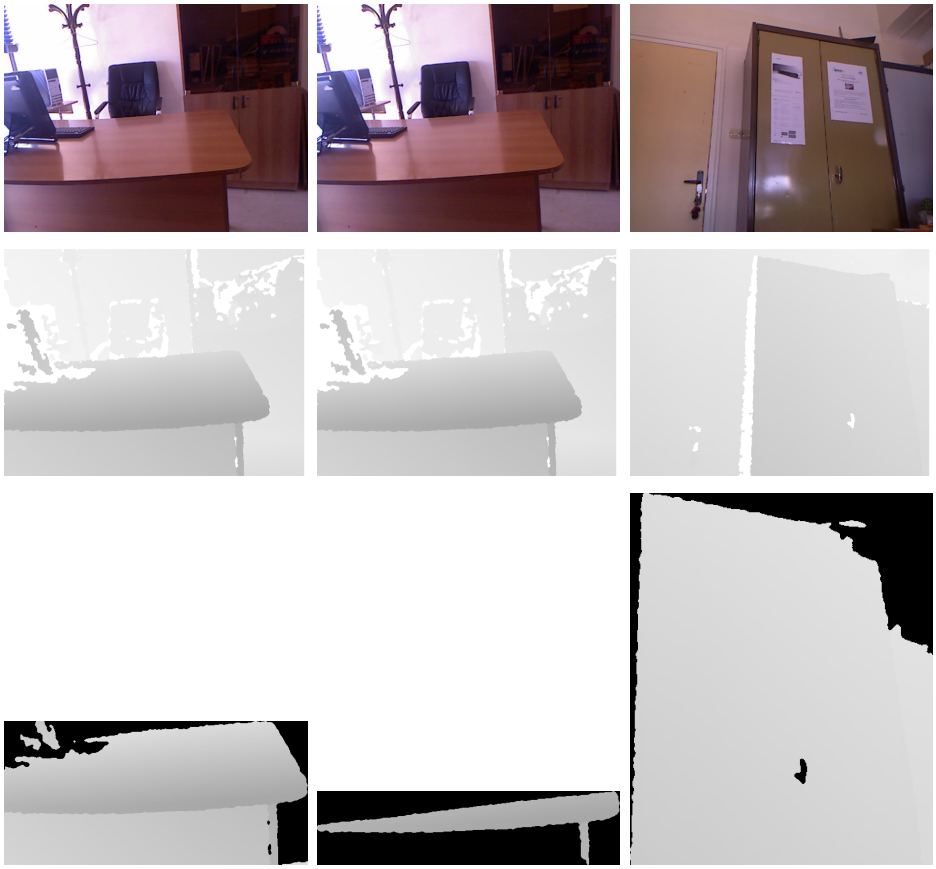

Figure 5.27: From top to down: the captured scene (table, table, dresser), its associated depth image (the system input), the resulting segment. The system has predicted: table, chair and dresser.

adapted FuseNet for RGB-D semantic segmentation that has 76.27% as accuracy. Compared to the third system [20], our model surpasses its accuracy and with a larger set of classes.

Table 5.3: Comparison with the classification module of the presented state-of-the-art systems.

	Input	Nb Classes	Classifier	Accuracy (%)
Wang et al.[7]	Depth image	5	Depth-based classifier [7]	-
Lin et al.[6]	RGB-D	70	FuseNet[18]	-
Wang et al.[20]	RGB-D	4	Cascaded Decision Tree	71.45
Ours	Depth map	6	VGG-16 based	72,81
Ours	Point cloud	14	PointNet[13]	91.67
Ours	Point cloud	6	PointNet[13]	**96.35**

5.5.3 Scene semantic labeling

For more experiments using the entire system with PointNet on real-world data, we show and discuss the output of each module for 4 images taken from a video sequence from our GDIS dataset [15] (Figure 5.28). After the ground detection (Figure 5.28 third row), we applied the DBSCAN clustering algorithm to break the occupied space into coarse segments (Figure 5.28 fourth row). After that, we compute for each segment its geometric features and its class.

For the segmentation module, we took the measurements of some objects in the real world and then compare it with the obtained geometric features. This is done by computing the MAE between the objects' height and the computed one. The obtained results showed a slight mean difference that does not exceed $30mm$. This small difference of $3cm$ will not affect the classification process except at the bounds of the defined intervals.

The Pointnet network has classified well the chair except for the last frame (Figure

5.28 last row, last column), it was classified as a table with a low probability (0.53), so the class was not mapped, only the convex hull was mapped. The model failed to predict the right class because the chair's point cloud was segmented into sub segments and this is due to the noisy nature of the Microsoft Kinect V1. If the class is not predicted, we set all the pins that represent the segment to the segment level, rather than mapping the class to that level.

As for running time evaluation, using scenes from GDIS dataset, we evaluated each step: point cloud segmentation, geometric features computing and point cloud classification. For each scene, we computed the running time of each step and then computed their mean. On average, the point cloud segmentation takes 1.700 seconds, geometric features computing takes 1.907×10^{-6} seconds for each segment and the point cloud classification step takes 0.807 seconds for all segments. For the ground detection, since we used the same algorithm, the running time is the same, on average, as in the previous system. It is related to the point cloud size and the model arguments for the segmentation step. The running time of geometric properties is negligible. As for the classification running time, it is strongly related to the complexity of the model and slightly related to the number of segments.

5.6 Discussion

The proposed system combines the deep learning features and the geometric features to provide for the user a scene description for a better human-scene interaction: for each segment, its label, location, shape, occupied space and its height are computed and mapped on the 3D interface.

The system detects 7 semantic classes that cover 16 different object classes. For each semantic class, we proposed a semantic label which is mapped to the center of the occupied space. Providing and only mapping coarse information is the first step for

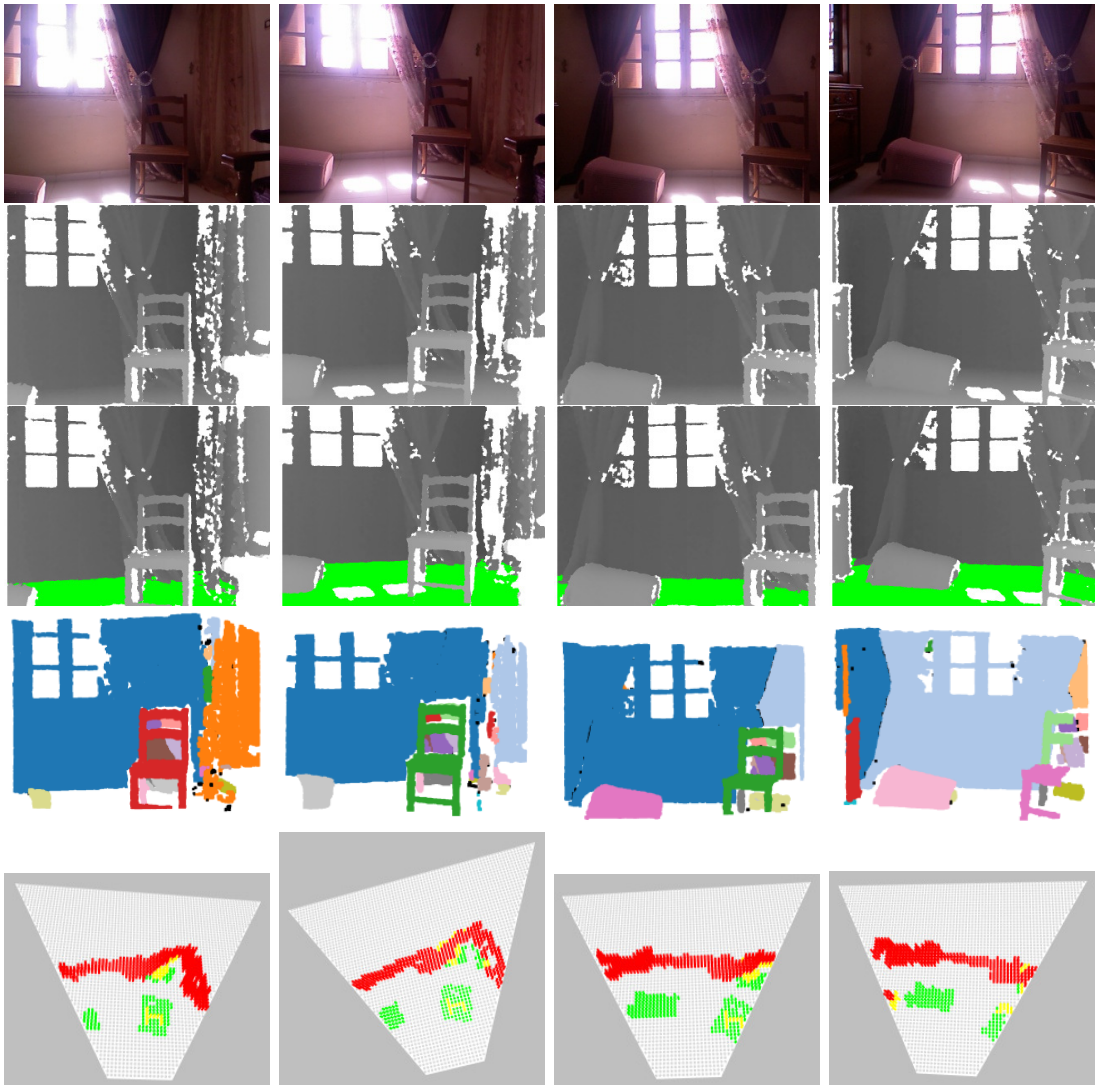

Figure 5.28: (From Top) The RGB image, the depth image, the output of the ground detection algorithm, the segmentation using DBSCAN and the semantic labeling mapped on the 3D interface. For more visibility, the color green, yellow and red represent levels 1, 2 and 3 respectively.

the scene description task. It provides a global scene description of the captured scene and how the objects are arranged in.

The proposed system fails to predict the semantic class of some point clouds due to the nature of real-world data that is incomplete and cropped. Different cropped objects can be similar to some cropped objects belonging to different classes. As current solution, the model ignores the predicted class if this latter has been predicted with a small probability.

5.7 Conclusion

In this chapter, we presented our latest system that employs a 3D interface for human-scene interaction. The system detects the ground as a first step using DCGD algorithm. It is more informative than the previous systems presented in chapter 4, since it takes into consideration the objects nature. It is able to detect a considerable number of classes while providing only 8 semantic labels that can be easily memorized. We validated our methods using two kinds of datasets: public and local datasets. This allowed us to compare our system with other systems and test it in real-world environment. Our methods showed efficient results. Some limitations have been detected during the test process as we reported in the discussion section. To this end, we will conclude our thesis with the next chapter, the conclusion and the future works.

Chapter 6

Conclusion

In this thesis, we proposed a scene semantic labeling for a better human-scene interaction. The semantic labeling that is inspired by the Braille and Kanji systems, is mapped on our proposed 3D interface. For this purpose, the designed system describes the captured scene architecture and provides geometric features and the nature of the detected objects mapped as the touch-based semantic labels. By sensing the generated interface, the user can understand his surroundings and thus interact with it whether by actions or contacts. The object classification approach is based on deep learning networks implemented to classify the 3D shape of the captured objects. Although we are using a state-of-the-art neural network, with our explained approach the accuracy has been improved by 5%.

The designed system acquires a depth image as input. Then the ground is detected using the DCGD algorithm and the occupied space is extracted for further segmentation using DBSCAN, an efficient clustering algorithm. Then, each segment is classified according to its nature and its geometric characteristics. The 3D labels of the estimated classes are finally mapped on our proposed device. However, the system only provides information about salient objects; other objects are not considered.

Furthermore, the system has shown promising quantitative results; however, in addi-

tion to the limitations discussed in the discussion section of the experiments chapter, an extensive clinical study should be performed for a qualitative validation.

6.1 Future works

This thesis has opened several doors to exploit, research in several fields: ground detection, point cloud segmentation, classification of 3D shapes and scene semantic labeling for human-scene interaction.

Regarding the DCGD algorithm, we aim to improve its performance so that it takes into account the holes. The holes are dangerous obstacles, especially for the visually impaired and blind people. Hence, taking them into consideration is an important step. In addition, the algorithm must differentiate between the ground and the lowest object when the ground does not appear.

As for the segmentation, a second step can be added in the future for fine segmentation. In this step, objects that are arranged on a given coarse segment can be detected using coarse to fine segmentation. These latter, can be mapped in 3D interface only by commands from the user to avoid ambiguity while exploring the scene. In other terms, mapping coarse and fine information at the same time is time consuming and ambiguous, such as mapping the table and the objects that are on it. We can imagine this scenario: the user is searching for a cup. After providing the global description, the user can locate the table on the 3D interface. After that, by long clicking on the table's label, we can at this time map on the whole surface of the area only the table and what's on it. At this step, by touching the new area's configuration, the user can have an idea about what is on the table and can search for his desired item.

Regarding the incomplete objects, a potential solution is that the proposed model will be extended to work for a succession of frames and thus, the cropped objects

can be completed by the alignment process and thus the prediction can be corrected. In addition, this latter can be used to improve the DCGD algorithm performance by comparing the previous ground's cuts with the current ones.

We also aim to provide a detailed scene semantic labeling, in a clear way, to include not only salient objects like tables, but also other small objects. This can be done by hierarchical classification, by considering the classification of the salient objects. For example, to classify objects that are on the table the system will consider that the object is on the table.

Bibliography

[1] Rolf Adams and Leanne Bischof. Seeded region growing. *IEEE Transactions on pattern analysis and machine intelligence*, 16(6):641–647, 1994.

[2] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[3] Ian Grosvenor and Natasha Macnab. 'seeing through touch': the material world of visually impaired children. *Educar em Revista*, (49):39–57, 2013.

[4] Lachlan Horne, Jose Alvarez, Chris McCarthy, Mathieu Salzmann, and Nick Barnes. Semantic labeling for prosthetic vision. *Computer Vision and Image Understanding*, 149:113–125, 2016.

[5] J. Rivera-Rubio, K. Arulkumaran, H. Rishi, I. Alexiou, and Bharath A.A. An assistive haptic interface for appearance-based indoor navigation. *Computer Vision and Image Understanding*, 149:126–145, 2016.

[6] Y. Lin, K. Wang, W. Yi, and S. Lian. Deep learning based wearable assistive system for visually impaired people. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2019.

[7] H. Wang, R. K. Katzschmann, S. Teng, B. Araki, L. Giarré, and D. Rus. Enabling independent navigation for visually impaired people through a wearable vision-based feedback system. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 6533–6540. IEEE, 2017.

[8] K. Konolige and P. Mihelich. Technical description of Kinect calibration. `http://wiki.ros.org/kinect_calibration/technical`, 2012.

[9] OpenCv team. Camera Calibration. `https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html`, 2018.

[10] 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.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[12] Rohit Thakur. Step by step VGG16 implementation in Keras for beginners. `https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c`, 2019.

[13] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[14] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[15] C. Zatout and S. Larabi. Dataset of ground detection for indoor scenes (GDIS dataset). `http://perso.usthb.dz/slarabi/DGIS.html`, 2019.

[16] A. Perez-Yus, D. Gutierrez-Gomez, G. Lopez-Nicolas, and J.J. Guerrero. Stairs

detection with odometry-aided traversal from a wearable rgb-d camera. *Computer Vision and Image Understanding*, 154:192–205, 2017.

[17] James M Coughlan and Alan L Yuille. Manhattan world: Compass direction from a single image by bayesian inference. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 941–947. IEEE, 1999.

[18] Caner Hazirbas, Lingni Ma, Csaba Domokos, and Daniel Cremers. Fusenet: Incorporating depth into semantic segmentation via fusion-based cnn architecture. In *Asian conference on computer vision*, pages 213–228. Springer, 2016.

[19] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[20] Zhe Wang, Hong Liu, Xiangdong Wang, and Yueliang Qian. Segment and label indoor scene based on rgb-d for the visually impaired. In *International Conference on Multimedia Modeling*, pages 449–460. Springer, 2014.

[21] Jinqiang Bai, Zhaoxiang Liu, Yimin Lin, Ye Li, Shiguo Lian, and Dijun Liu. Wearable travel aid for environment perception and navigation of visually impaired people. *Electronics*, 8(6):697, 2019.

[22] R. Trabelsi, I. Jabri, F. Melgani, F. Smach, N. Conci, and A. Bouallegue. Indoor object recognition in rgbd images with complex-valued neural networks for visually-impaired people. *Neurocomputing*, 330:94–103, 2019.

[23] R. Vaza, P.O. Fernandesb, and A.C.R. Veiga. Designing an interactive exhibitor for assisting blind and visually impaired visitors in tactile exploration of original museum pieces. *Procedia Computer Science*, 138:561–570, 2018.

[24] M. Chessa, N. Noceti, F. Odone, F. Solari, J. Sosa-García, and L. Zini. An in-

tegrated artificial vision framework for assisting visually impaired users. *Computer Vision and Image Understanding*, 149:209–228, 2016.

[25] J. P. Gomes, J. P. Sousa, C. R. Cunha, and E. P. Morais. An indoor navigation architecture using variable data sources for blind and visually impaired persons. In *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–5. IEEE, 2018.

[26] M. Nasralla, I. Rehman, D. Sobnath, and S. Paiva. Computer vision and deep learning-enabled uavs: proposed use cases for visually impaired people in a smart city. In *International Conference on Computer Analysis of Images and Patterns*, pages 91–99. Springer, 2019.

[27] R Gnana Praveen and Roy P Paily. Blind navigation assistance for visually impaired based on local depth hypothesis from a single image. *Procedia Engineering*, 64:351–360, 2013.

[28] Nadia Kanwal, Erkan Bostanci, Keith Currie, and Adrian F Clark. A navigation system for the visually impaired: a fusion of vision and depth sensor. *Applied bionics and biomechanics*, 2015, 2015.

[29] V. Nair, M. Budhai, G. Olmschenk, W. H. Seiple, and Z. Zhu. Assist: personalized indoor navigation via multimodal sensors and high-level semantic information. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018.

[30] S. Mascetti, D. Ahmetovic, A. Gerino, C. Bernareggi, M. Busso, and A. Rizzi. Robust traffic lights detection on mobile devices for pedestrians with visual impairment. *Computer Vision and Image Understanding*, 148:123–135, 2016.

[31] P. Gharani and H. A. Karimi. Context-aware obstacle detection for navigation by visually impaired. *Image and Vision Computing*, 64:103–115, 2017.

[32] Y.H. Lee and G. Medioni. Rgb-d camera based wearable navigation system for

the visually impaired. *Computer Vision and Image Understanding*, 149:3–20, 2016.

[33] D. Nandini and K.R. Seeja. A novel path planning algorithm for visually impaired people. *Journal of King Saud University – Computer and Information Sciences*, 31:385–391, 2019.

[34] João Guerreiro, Dragan Ahmetovic, Kris M Kitani, and Chieko Asakawa. Virtual navigation for blind people: Building sequential representations of the real-world. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility*, pages 280–289. ACM, 2017.

[35] Dragan Ahmetovic, Cole Gleason, Chengxiong Ruan, Kris Kitani, Hironobu Takagi, and Chieko Asakawa. Navcog: a navigational cognitive assistant for the blind. In *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 90–99, 2016.

[36] Z. Bauer, A. Dominguez, E. Cruz, F. Gomez-Donoso, S. Orts-Escolano, and M. Cazorla. Enhancing perception for the visually impaired with deep learning techniques and low-cost wearable sensors. *Pattern Recognition Letters*, 2019.

[37] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.

[38] Kircali Dogan and Tek Boray. Ground plane detection using an rgb-d sensor. In *Information Sciences and Systems*, pages 69–77. Springer, 2014.

[39] Li Bing, Zhang Xiaochen, Pablo Muñoz J., Xiao Jizhong, Rong Xuejian, and Tian Yingli. Assisting blind people to avoid obstacles: A wearable obstacle stereo feedback system based on 3d detection. In *IEEE Conference on Robotics and Biomimetics*, pages 2307–2311. IEEE, 2015.

[40] L. Puig A. Aladren, G. Lopez-Nicolas and J. J. Guerrero. Navigation assist-

ance for the visually impaired using rgb-d sensor with range expansion. *IEEE Systems Journal, vol. 10, no. 3*, pages 922–932, 2016.

[41] Ching-Tang Hsieh Hsieh-Chang Huang and Cheng-Hsiang Yeh. An indoor obstacle detection system using depth information and region growth. *Sensors*, 15:27117–27141.

[42] J.J. Guerrero A. Perez Yus, G. Lopez Nicolas. Detection and modelling of staircases using a wearable depth sensor. In *ECCV 2014: Computer Vision - ECCV 2014 Workshops*, pages 449–463. Springer.

[43] V. Hoang, T. Nguyen, and T. Le. Obstacle detection and warning system for visually impaired people based on electrode matrix and mobile kinect. *Vietnam J Comput Sci (4)*, pages 71–83, 2017.

[44] M. Cormier, K. Moffatt, R. Cohen, and R. Mann. Purely vision-based segmentation of web pages for assistive technology. *Computer Vision and Image Understanding*, 148:46–66, 2016.

[45] Aoran Xiao, Ruizhi Chen, Deren Li, Yujin Chen, and Dewen Wu. An indoor positioning system based on static objects in large indoor scenes by using smartphone cameras. *Sensors*, 18(7):2229, 2018.

[46] Yujin Chen, Ruizhi Chen, Mengyun Liu, Aoran Xiao, Dewen Wu, and Shuheng Zhao. Indoor visual positioning aided by cnn-based image retrieval: Training-free, 3d modeling-free. *Sensors*, 18(8):2692, 2018.

[47] Marek Kraft, Michał Nowicki, Adam Schmidt, Michał Fularz, and Piotr Skrzypczyński. Toward evaluation of visual navigation algorithms on rgb-d data from the first-and second-generation kinect. *Machine Vision and Applications*, 28(1-2):61–74, 2017.

[48] R. Zeineldin and N. El-Fishawy. Fast and accurate ground plane detection for

the visually impaired from 3d organized point clouds. *2016 SAI Computing Conference (SAI)*, pages 373–379, 07 2016.

[49] Péter Fankhauser, Michael Bloesch, Diego Rodriguez, Ralf Kaestner, Marco Hutter, and Roland Siegwart. Kinect v2 for mobile robot navigation: Evaluation and modeling. In *2015 International Conference on Advanced Robotics (ICAR)*, pages 388–394. IEEE, 2015.

[50] Walter CSS Simões and VF De Lucena. Blind user wearable audio assistance for indoor navigation based on visual markers and ultrasonic obstacle detection. In *2016 IEEE International Conference on Consumer Electronics (ICCE)*, pages 60–63. IEEE, 2016.

[51] R. Abobeah, M. E. Hussein, M. Abdelwahab, and A. Shoukry. Wearable rgb camera-based navigation system for the visually impaired. In *VISIGRAPP (5: VISAPP)*, pages 555–562, 2018.

[52] Paraskevas Diamantatos and Ergina Kavallieratou. Android based electronic travel aid system for blind people. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 585–592. Springer, 2014.

[53] P. Costa, H. Fernandes, P. Martins, J. Barroso, and L. J. Hadjileontiadis. Obstacle detection using stereo imaging to assist the navigation of visually impaired people. *Procedia Computer Science*, 14:83–93, 2012.

[54] Feixiang Lu, Bin Zhou, Yu Zhang, and Qinping Zhao. Real-time 3d scene reconstruction with dynamically moving object using a single depth camera. *The Visual Computer*, 34(6-8):753–763, 2018.

[55] K. Thakoor, N. Mante, C. Zhang, C. Siagian, J. Weiland, L. Itti, and G. Medioni. A system for assisting the visually impaired in localization and grasp of desired objects. In *European Conference on Computer Vision*, pages 643–657. Springer, 2014.

[56] Chayma Zatout, Slimane Larabi, Ilyes Mendili, and Soedji Ablam Edoh Barnabé. Ego-semantic labeling of scene from depth image for visually impaired and blind people. In *2019 IEEE/CVF International Conference on Computer Vision Workshops, ICCV Workshops 2019, Seoul, Korea (South), October 27-28, 2019*, pages 4376–4384. IEEE, 2019.

[57] Matteo Poggi and Stefano Mattoccia. A wearable mobility aid for the visually impaired based on embedded 3d vision and deep learning. In *2016 IEEE Symposium on Computers and Communication (ISCC)*, pages 208–213. IEEE, 2016.

[58] Young Hoon Lee and Gerard Medioni. Rgb-d camera based navigation for the visually impaired. In *Proceedings of the RSS*, volume 2, 2011.

[59] A. Bhowmick, S. Prakash, R. Bhagat, V. Prasad, and S. M. Hazarika. Intellinavi: Navigation for blind based on kinect and machine learning. In *International Workshop on Multi-disciplinary Trends in Artificial Intelligence*, pages 172–183. Springer, 2014.

[60] Gaurav Bhorkar. A survey of augmented reality navigation. *arXiv preprint arXiv:1708.05006*, 2017.

[61] O. Halabi, M. Al-Ansari, Y. Halwani, F. Al-Mesaifri, and R. Al-Shaabi. Navigation aid for blind people using depth information and augmented reality technology. *Proceedings of the NICOGRAPH International*, pages 120–125, 2012.

[62] Brian FG Katz, Slim Kammoun, Gaëtan Parseihian, Olivier Gutierrez, Adrien Brilhault, Malika Auvray, Philippe Truillet, Michel Denis, Simon Thorpe, and Christophe Jouffrais. Navig: augmented reality guidance system for the visually impaired. *Virtual Reality*, 16(4):253–269, 2012.

[63] J. Bai, S. Lian, Z. Liu, K. Wang, and D. Liu. Smart guiding glasses for visu-

ally impaired people in indoor environment. *IEEE Transactions on Consumer Electronics*, 63(3):258–266, 2017.

[64] W. Jeamwatthanachai, M. Wald, and G. Wills. Map data representation for indoor navigation by blind people. *International Journal of Chaotic Computing*, 4(1):70–78, 2017.

[65] Jongmoo Choi. *Range Sensors: Ultrasonic Sensors, Kinect, and LiDAR*, pages 2521–2538. Springer Netherlands, 2019.

[66] Song Zhang and Peisen S Huang. Novel method for structured light system calibration. *Optical Engineering*, 45(8):083601, 2006.

[67] Xing Zhou. A study of microsoft kinect calibration. *Dept. of Computer Science, George Mason University, Fairfax*, 2012.

[68] Po-Han Huang, Kevin Matzen, Johannes Kopf, Narendra Ahuja, and Jia-Bin Huang. Deepmvs: Learning multi-view stereopsis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2821–2830, 2018.

[69] Huizhong Zhou, Benjamin Ummenhofer, and Thomas Brox. Deeptam: Deep tracking and mapping. In *Proceedings of the European conference on computer vision (ECCV)*, pages 822–838, 2018.

[70] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. In *2016 Fourth international conference on 3D vision (3DV)*, pages 239–248. IEEE, 2016.

[71] Zhixiang Hao, Yu Li, Shaodi You, and Feng Lu. Detail preserving depth estimation from a single image using attention guided networks. In *2018 International Conference on 3D Vision (3DV)*, pages 304–313. IEEE, 2018.

[72] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and

Dacheng Tao. Deep ordinal regression network for monocular depth estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2002–2011, 2018.

[73] Ibraheem Alhashim and Peter Wonka. High quality monocular depth estimation via transfer learning. *arXiv preprint arXiv:1812.11941*, 2018.

[74] Xian-Feng Han, Jesse S Jin, Ming-Jie Wang, Wei Jiang, Lei Gao, and Liping Xiao. A review of algorithms for filtering the 3d point cloud. *Signal Processing: Image Communication*, 57:103–112, 2017.

[75] Carlos Moreno and Ming Li. A comparative study of filtering methods for point clouds in real-time video streaming. In *Proceedings of the World Congress on Engineering and Computer Science*, volume 1, pages 388–393, 2016.

[76] Xian-Feng Hana, Jesse S Jin, Juan Xie, Ming-Jie Wang, and Wei Jiang. A comprehensive review of 3d point cloud descriptors. *arXiv preprint arXiv:1802.02297*, 2, 2018.

[77] Luis A Alexandre. 3d descriptors for object and category recognition: a comparative evaluation. In *Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vilamoura, Portugal*, volume 1, page 7. Citeseer, 2012.

[78] Carlos Manuel Mateo, Pablo Gil, and Fernando Torres. A performance evaluation of surface normals-based descriptors for recognition of objects using cad-models. In *2014 11th international conference on informatics in control, automation and robotics (ICINCO)*, volume 2, pages 428–435. IEEE, 2014.

[79] Yulan Guo, Mohammed Bennamoun, Ferdous Sohel, Min Lu, Jianwei Wan, and Ngai Ming Kwok. A comprehensive performance evaluation of 3d local feature descriptors. *International Journal of Computer Vision*, 116(1):66–89, 2016.

[80] Andrew Edie Johnson and Martial Hebert. Surface matching for object recognition in complex three-dimensional scenes. *Image and Vision Computing*, 16(9-10):635–651, 1998.

[81] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Persistent point feature histograms for 3d point clouds. In *Proc 10th Int Conf Intel Autonomous Syst (IAS-10), Baden-Baden, Germany*, pages 119–128, 2008.

[82] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE international conference on robotics and automation*, pages 3212–3217. IEEE, 2009.

[83] Bertram Drost, Markus Ulrich, Nassir Navab, and Slobodan Ilic. Model globally, match locally: Efficient and robust 3d object recognition. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 998–1005. Ieee, 2010.

[84] Zoltan-Csaba Marton, Dejan Pangercic, Nico Blodow, and Michael Beetz. Combined 2d–3d categorization and classification for multimodal perception systems. *The International Journal of Robotics Research*, 30(11):1378–1402, 2011.

[85] Marius Muja, Radu Bogdan Rusu, Gary Bradski, and David G Lowe. Rein-a fast, robust, scalable recognition infrastructure. In *2011 IEEE international conference on robotics and automation*, pages 2939–2946. IEEE, 2011.

[86] Anh Nguyen and Bac Le. 3d point cloud segmentation: A survey. In *2013 6th IEEE conference on robotics, automation and mechatronics (RAM)*, pages 225–230. IEEE, 2013.

[87] Eleonora Grilli, Fabio Menna, and Fabio Remondino. A review of point clouds

segmentation and classification algorithms. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 42:339, 2017.

[88] Paul J. Besl and Ramesh C Jain. Segmentation through variable-order surface fitting. *IEEE Transactions on pattern analysis and machine intelligence*, 10(2):167–192, 1988.

[89] Minjie Fan and Thomas CM Lee. Variants of seeded region growing. *IET image processing*, 9(6):478–485, 2015.

[90] Fayez Tarsha-Kurdi, Tania Landes, and Pierre Grussenmeyer. Hough-transform and extended ransac algorithms for automatic detection of 3d building roof planes from lidar data. In *ISPRS Workshop on Laser Scanning 2007 and SilviLaser 2007*, volume 36, pages 407–412, 2007.

[91] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[92] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient ransac for point-cloud shape detection. In *Computer graphics forum*, volume 26, pages 214–226. Wiley Online Library, 2007.

[93] Ramy Ashraf Zeineldin and Nawal Ahmed El-Fishawy. A survey of ransac enhancements for plane detection in 3d point clouds. *Menoufia Journal of Electronic Engineering Research*, 26(2):519–537, 2017.

[94] RA Kuçak, E Özdemir, and S Erol. The segmentation of point clouds with k-means and ann (artifical neural network). *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 42:595, 2017.

[95] Sampath Aparajithan and Shan Jie. Clustering based planar roof extraction from lidar data. In *ASPRS 2006 Annual Conference*, 2006.

[96] Thomas Melzer. Non-parametric segmentation of als point clouds using mean shift. *Journal of Applied Geodesy Jag*, 1(3):159–170, 2007.

[97] Josep Miquel Biosca and José Luis Lerma. Unsupervised robust planar segmentation of terrestrial laser scanner point clouds based on fuzzy clustering methods. *ISPRS Journal of Photogrammetry and Remote Sensing*, 63(1):84–98, 2008.

[98] T Czerniawski, B Sankaran, M Nahangi, C Haas, and F Leite. 6d dbscan-based segmentation of building point clouds for planar object classification. *Automation in Construction*, 88:44–58, 2018.

[99] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1887–1893. IEEE, 2018.

[100] Chunxiao Wang, Min Ji, Jian Wang, Wei Wen, Ting Li, and Yong Sun. An improved dbscan method for lidar data segmentation with automatic eps estimation. *Sensors*, 19(1):172, 2019.

[101] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[102] Andrea Frome, Daniel Huber, Ravi Kolluri, Thomas Bülow, and Jitendra Malik. Recognizing objects in range data using regional point descriptors. In *European conference on computer vision*, pages 224–237. Springer, 2004.

[103] Aleksey Golovinskiy, Vladimir G Kim, and Thomas Funkhouser. Shape-based recognition of 3d point clouds in urban environments. In *2009 IEEE 12th International Conference on Computer Vision*, pages 2154–2161. IEEE, 2009.

[104] Alex Teichman, Jesse Levinson, and Sebastian Thrun. Towards 3d object recog-

nition via classification of arbitrary object tracks. In *2011 IEEE International Conference on Robotics and Automation*, pages 4034–4041. IEEE, 2011.

[105] Jens Behley, Volker Steinhage, and Armin B Cremers. Performance of histogram descriptors for the classification of 3d laser range data in urban environments. In *2012 IEEE International Conference on Robotics and Automation*, pages 4391–4398. IEEE, 2012.

[106] Richard Socher, Brody Huval, Bharath Bath, Christopher D Manning, and Andrew Ng. Convolutional-recursive deep learning for 3d object classification. *Advances in neural information processing systems*, 25:656–664, 2012.

[107] Nico Höft, Hannes Schulz, and Sven Behnke. Fast semantic segmentation of rgb-d scenes with gpu-accelerated deep neural networks. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 80–85. Springer, 2014.

[108] Ian Lenz, Honglak Lee, and Ashutosh Saxena. Deep learning for detecting robotic grasps. *The International Journal of Robotics Research*, 34(4-5):705–724, 2015.

[109] Luís A Alexandre. 3d object recognition using convolutional neural networks with transfer learning between input channels. In *Intelligent Autonomous Systems 13*, pages 889–898. Springer, 2016.

[110] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015.

[111] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[112] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural

network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015.

[113] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017.

[114] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8895–8904, 2019.

[115] Jiaxin Li, Ben M Chen, and Gim Hee Lee. So-net: Self-organizing network for point cloud analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9397–9406, 2018.

[116] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[117] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[118] Sameera Ramasinghe, Salman Khan, Nick Barnes, and Stephen Gould. Spectral-gans for high-resolution 3d point-cloud generation. *arXiv preprint arXiv:1912.01800*, 2019.

[119] Stefan Milz, Martin Simon, Kai Fischer, Maximillian Pöpperl, and Horst-Michael Gross. Points2pix: 3d point-cloud to image translation using conditional gans. In *German Conference on Pattern Recognition*, pages 387–400. Springer, 2019.

[120] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[121] Ruiqi Guo and Derek Hoiem. Support surface prediction in indoor scenes. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2144–2151, 2013.

[122] R. B. Rusu and S. Cousins. 3d is here: Point cloud library (pcl). In *2011 IEEE International Conference on Robotics and Automation*, pages 1–4. IEEE, May 2011.

[123] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. IEEE, 2009.

[124] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

[125] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[126] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.