Abstract

Autonomous robots have become more useful in research and industrial applications. For example, they can be found in hospitals, industries, military, and agriculture. They are capable of navigating an uncontrolled environment and performing many tasks without any help or guidance. Nowadays, because of their high influence in our economic and daily life, autonomous robots are a major focus of the research.

Environment perception is an important field in robotics. It enables the robot to collect information about its surrounding, analyze them, and make decisions based on that. Two of the main challenging tasks in this field are scene understanding and obstacle avoidance.

Obstacles are the objects encountered by the robot in its path to its target destination. Thus, Avoiding them is a critical step that helps the robot to perform its mission. There are various methods to tackle the obstacle detection problem, and we propose a 3D points cloud-based approach.

In this thesis, we propose an obstacle detection algorithm for a mobile robot. Basically, we will use an RGB-D dataset acquired by a Kinect v2.0 sensor. This sensor can work indoor and outdoor. Then we will apply computer vision-based techniques to detect the obstacles.

keywords: *perception, scene understanding, computer vision, autonomous robots, obstacle detection, 3D points cloud.*

Acknowledgements

This thesis has been conducted in the Moivre department of the université de Sherbrooke.

I would like to express my heartfelt thanks to my supervisor Professor **Djemel ZIOU** *for having welcomed me at* **THE MOIVRE GROUP** *in* **the université de Sherbrooke** *and also for the permanent encouragement, the valuable advice he provided me, and his enthusiasm, support, and guidance throughout the course of this research.*

I take this opportunity to express my gratitude to all of the MOIVRE Research Group members for their help and support.

I would like to thank my teacher, Professor **Riadh BEN ABDALLAH** *for his collaboration, suggestions and for the help he has given me.*

I would also like to thank Mrs. **Dominique CHOQUETTE** from the Quebec's Ministry of Agriculture, Fisheries and Food *for her collaboration.*

Finally, I cannot forget to thank my family for providing me the full support and motivation. This accomplishment would not happen without their support.

Contents

List of Figures

List of Tables

Abbreviations

MOIVRE	**MO**delisation en **I**magerie **V**ision et **RE**saux de neurones
PCA	**P**rincipal **C**omponents **A**nalysis
3D	**3 D**imensional
KNN	**K** **N**earest **N**eighbors
SL	**S**tructured **L**ight
TOF	**T**ime **O**f **F**light
SLC	**S**tandard **L**inear **C**ombination

I dedicate this work to my beloved family and colleagues, to express my gratitude for their endless support...

Introduction

With the appearance of new technologies like artificial intelligence and information engineering and with the exponential raise of digital sensors, more powerful robotics tools are introduced. These tools enable robots to accomplish their tasks efficiently without any external influence. They are capable of gaining information about their surroundings and avoiding obstacles in their ways. Today, these smart, autonomous, and self-learning machines are already everywhere: agriculture, hospitals, military, restaurants, etc.

One of the most important areas in robotics research is robot perception. It is related to various applications where sensory data, computer vision, and artificial intelligence are the main used techniques. As examples of such challenges, we can set obstacle avoidance, scene understanding, and activity recognition. Recently, expensive 3D lasers were used for the visual perception to perform these tasks, but not all of the research departments are able to provide such devices. Fortunately, in last decade, all the researchers throughout the world have had access to Microsoft Kinect, which is a cheap 3D perception equipment capable of performing such tasks [19].

The goal of this thesis is to propose and implement an obstacle detection system using RGB-D images.

The outline of this report is as follows:

- **General framework of the project:** This stage consists in implementing the working environment as well as bringing a global view on 3D obstacle detection. Then, we describe the problem statement and the project objectives.
- **Theoretical background:** Here we start by introducing the theoretical background needed for the obstacle detection algorithm used in this project.
- **Methodology:** In this stage, we detail the proposed solution to tackle the given problem.

- **Experimental setup results, and discussions:** This chapter presents the experimental setup and the results of our work along with some brief discussions and comments.
- **Conclusion and perspectives:** This is the synthesis to all our efforts with some perspectives.

Chapter 1

General Framework of the Project

1.1 Introduction

In this chapter, we will present the host research group the **MOdélisation en Imagerie, Vision et REseaux de neurones (MOIVRE) lab** and the general context of the project. Then, we will give an overview of the obstacle detection challenge. Later on, we will set the problem statement. Finally, we will detail the goals.

1.2 Presentation of the host organization

The Moivre research group, created in 1997, is led by Professor **Djemel Ziou**. This group is located in the computer science department of the université de Sherbrooke in Canada. Computer graphics, image and video processing, artificial vision, virtual reality, Machine learning, and deep learning are the main research areas in the lab. This allows the group to engage in large scale academic and industrial projects such as **Obstacles Detection for Mobile Robots**, which is my thesis project.

This work is carried out as part of an industrial project in collaboration with the Quebec's Ministry of Agriculture, Fisheries and Food. The original purpose of the project is to build a computer vision model that enables an agricultural robot to detect obstacles in a farming environment while navigating between the trees, eliminating weeds...

1.3 Problem statement

Obstacle detection for an autonomous robot is a challenging task. Finding a precise method to detect obstacles and non-safe areas for mobile robots is the key to accomplish their missions efficiently. There are various existing methods that solve the obstacle detection problem [20–23]. We can note, for example, the recent success of deep learning-based approaches. However, this technique requires a larger amount of data and powerful computer hardware.

With the arrival of 3D sensors, the concept of 3D imaging has been of much concern. Autonomous robots can use these 3D sensors to detect obstacles by creating a depth map of the target environment and use it with the RGB image to perform the task.
In this project, the aim is to build a computer vision model that can be applied in a various environments and enables an autonomous robot to detect obstacles while accomplishing its mission using RGB-D images provided by a 3D sensor.

Obstacles can be dynamic or static depending on the scenario and the application domain. For example, obstacles can be: trees, animals, walls, and humans. In this work, obstacle detection approach will be tackled using RGB-D data "mainly depth images". The data will be provided by an RGB-D sensor. So the choice of the sensor is another required task.
The Figure 1.1 represents the process discussed during this report.

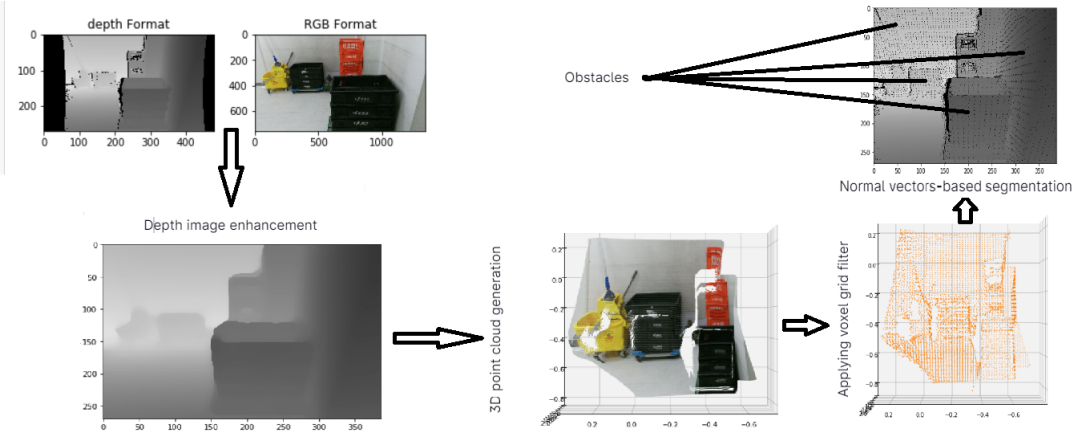

FIGURE 1.1: The process of obstacles detection.

1.4 Goals and objectives

The main goal of this project is to build a robust obstacle detection system for a mobile robot using RGB-D data provided by a suitable RGB-D sensor. For this purpose, we

aim to provide a convincing method and useful dataset. The specific project tasks are as follows:

- Choosing the right sensor.
- Having sets of depth maps and RGB-images of the environment of interest.
- Creating a 3D cloud of the target scene from the depth and the RGB images.
- Developing an accurate method of 3D obstacle detection.

1.5 Conclusion

In this first chapter, we introduced the general context of the work, after giving a presentation of the **Moivre** research group and the context of the project. Then, we have described the problem statement and we detailed our goals.

Chapter 2

Theoretical Background

2.1 Introduction

In this second chapter, we will present the important fundamentals that will help us to understand the work done during this report. First, we will set the foundations for the work done. Later, we will describe important operations used to tackle the obstacle detection problem.

2.2 Basic 3D elements

2.2.1 Point

A point in the 3D space is a zero-dimensional geometric element. It can be defined by a 3D vector, which connects the origin of the base and the point position.

In this report, a point P in a 2D space will be defined by a vector $\vec{p}$ of order 2:

$$P : \vec{p} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.1}$$

In this report, a point P in a 3D space will be defined by a vector $\vec{p}$ of order 3:

$$P : \vec{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.2}$$

where x, y, and z are the coordinates of the point P in the given base.

2.2.2 Straight line

A straight line is a one-dimensional geometric element. In a 3D space, a line L is defined by a position vector of one point on the line and the direction vector of the line, as illustrated in the Figure 2.1. In fact, through a point A(x_a,y_a,z_a), with a position vector $\vec{r_0}$=$x_a\vec{i}$+$y_a\vec{j}$+$z_a\vec{k}$, passes a line directed by a vector $\vec{s}$. So the line equation is given by:

$$\vec{r} = \vec{r_0} + t\vec{s},$$

with t $\in R$.

$$\vec{r}: \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} + t \begin{bmatrix} x_s \\ y_s \\ z_s \end{bmatrix} \tag{2.3}$$

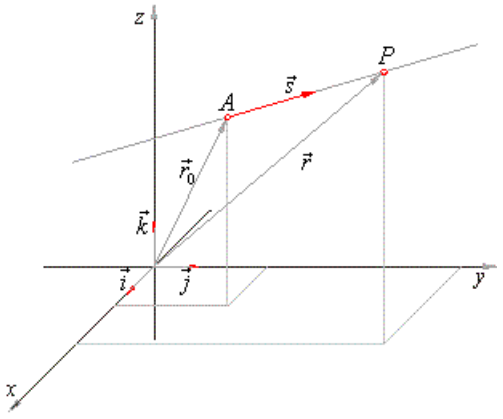

FIGURE 2.1: Equation of a line in a 3D space [1].

2.2.3 Plane

A 3D space contains many planes. A plane is a two-dimensional geometric element. In a 3D space, a plane P is defined by 3 points (non-colinear) or by a normal vector to the plane and a point. In the Figure 2.2 below, we have:

- $\overrightarrow{P_1P} = \vec{r} - \vec{r_1}$.
- $N(A, B, C)$ is a normal vector to the plane so: $(\vec{r} - \vec{r_1}) \cdot \vec{N} = 0 \iff (x - x_1) \times A + (y - y_1) \times B + (z - z_1) \times C = 0$.

The plane equation is given by:

$$A \times x + B \times y + C \times z + D = 0$$

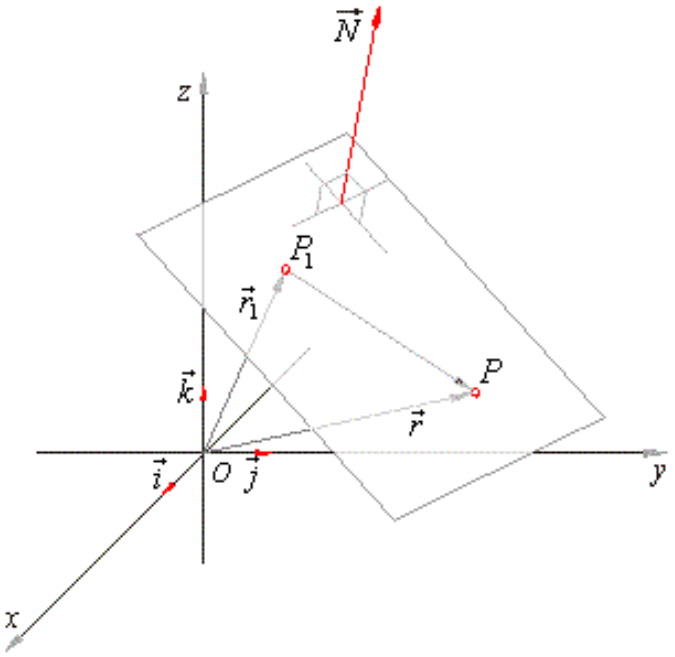

FIGURE 2.2: Equation of a plane in a 3D space [2].

2.3 Operations in a 3D space

In this stage, we will introduce some basic operations in a 3D space. These operations will be needed to understand the used approach later.

2.3.1 The distance between two points

Given two points $p_1(x_1, y_1, z_1)$ and $p_2(x_2, y_2, z_2)$ in a 3D space, the distance between them is given by the the following formula:

$$d(p_1, p_2) = \sqrt{((x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2)}$$

2.3.2 Projection of a point onto a line

Given a point $p(x, y, z)$, the orthogonal projection of the point p onto a straight line that has as a direction vector $\vec{u}$(a,b,c) is given by:

$$V = \overrightarrow{op} \cdot \overrightarrow{u} = x \times a + y \times b + z \times c,$$

where o is the center of the 3D space and with a constraint: $\overrightarrow{u} \cdot \overrightarrow{u} = 1$.

2.3.3 Projection of a point onto a plane

- The projection of a point $p(x, y, z)$ onto the xy-plane is the point $p_1(x, y, 0)$.
- The projection of a point $p(x, y, z)$ onto the xz-plane is the point $p_2(x, 0, z)$.
- The projection of a point $p(x, y, z)$ onto the yz-plane is the point $p_3(0, y, z)$.

2.3.4 K-Nearest Neighboors of a point

K-Nearest Neighbors (KNN) is one of the most used algorithms in machine learning. For a given 3D cloud of points, KNN returns for each point its K nearest neighbors.

KNN returns the target neighbors of a given point after performing some distance measurement between them. In KNN algorithm, closer data are considered similar to each other.

Examples of distance measuring techniques
Let $P(x_1, x_2, x_3)$ and $Q(y_1, y_2, y_3)$ be two points on the 3D space. The distance between these two points can be defined by:

- Euclidean Distance:

$$\sum_{n=1}^{3}(x_i - y_i)^2$$

- Manhattan Distance:

$$\sum_{n=1}^{3}|(x_i - y_i)|$$

- Minkowski Distance:

$$\sum_{n=1}^{3}(|(x_i - y_i)|^p)^{1/p}$$

The Figure 2.3 illustrates the KNN principle in 2D space for the case K=3.

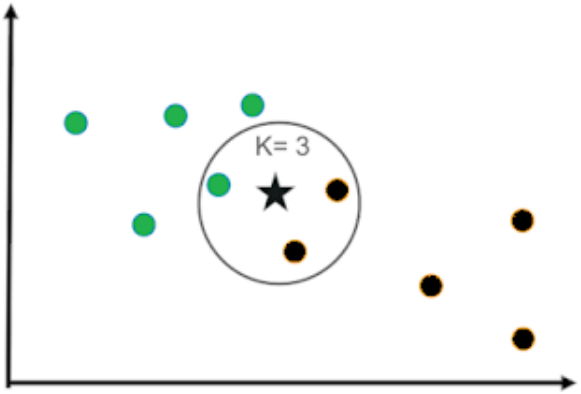

FIGURE 2.3: 3 nearest neighbors [3].

2.3.5 Summary statistics

Data matrix:

A data matrix M is a matrix that has m rows (observations or examples) and n columns (variables). To represent a 3D cloud n will be equal to 3.

$$\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ . & . & . \\ . & . & . \\ . & . & . \end{bmatrix}$$

Variance:

It measures the dispersion of a set of points around their mean. A variance of zero value means that all the points are identical. Giving a random variable X, the variance is calculated as below:

$$var(X) = E((X - E(X))^2)$$

$$E[X] = \overline{X} = \frac{1}{n}\sum_{0}^{n} x_i$$

$$var(X) = \frac{1}{n}\sum_{0}^{n}(x_i - \overline{X})^2$$

Covariance:

The covariance between two variables measures how these random variables vary together around their means. It is calculated as below:

$$Cov(X, Y) = E((X - E(X)(Y - E(Y))$$

$$Cov(X,Y) = \frac{1}{n}\sum_{0}^{n}(x_i - \overline{X})(y_i - \overline{Y})$$

Correlation:

The correlation between two random variables describes the degree to which the variables are related by moving together.

$$Corr(X,Y) = Cov(X,Y)/\sqrt{var(X)var(Y)}$$

The Figure 2.4 represents various correlations values.

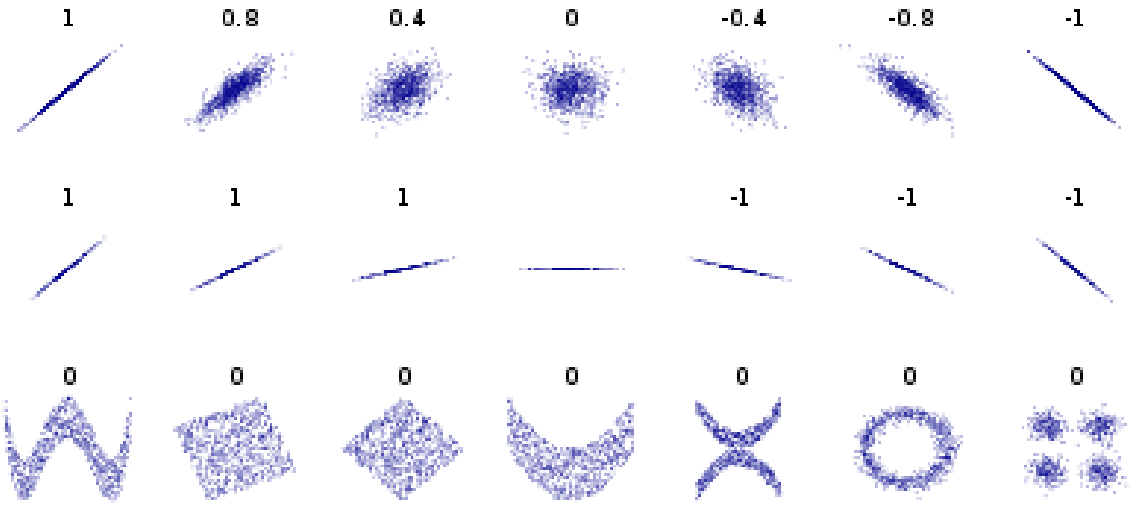

FIGURE 2.4: Examples of correlation values [4].

Eigenvalue and eigenvector decomposition of a matrix:

If an $N \times N$ matrix M is written this way:

$$Mu = \lambda u$$

If u is a non zero vector, it is called the eigenvector of M and λ is the eigenvalue of M corresponding to u.

2.3.6 Principal Component Analysis (PCA)

PCA aims to map the data into a new orthonormal base where the first component of the base has the maximum variance of the data, the second component has the second maximum variance, and so on. PCA is an orthogonal linear transformation.

The principle of PCA is:
Starting with N variables $X_1, ..., X_N$ and Finding a rotation of these variables $Y_1, ..., Y_N$ (principal components). Where $Y_1, ..., Y_N$ are uncorrelated and $var(Y_1) > var(Y_2)..... > var(Y_N)$

So the principal components are calculated this way:

- Giving a $m \times n$ data matrix $X = (x_1, ...x_p)^T$
- We say that $a^T X$ is a standard linear combination if $\sum_0^n a^2 = 1$
- Find the standard linear combination SLC_{a_1} so that $y_1 = a_1^T X$ has maximum variance.
- Find the SLC_{a_2} so that $y_2 = a_2^T X$ has a maximum variance, subject to the constraint that y_1 is uncorrelated to y_2 ($y_2 \perp y_1$)
- Find the SLC_{a_3} so that $y_3 = a_3^T X$ has a maximum variance, subject to the constraint that y_3 is uncorrelated to y_2 and y_1 ($y_3 \perp y_1, y_3 \perp y_2$)
- Repeat the same computation for the other components

Applying PCA in a 3D space

Let X be an $m \times 3$ matrix containing a set of points and (y_1, y_2, y_3) be the three principal components of this data. Following what we mentioned above, the first principle component for example can be found as follows:

$$y_1 = a^T X$$

$var(y_1) = var(a^T X) = (a^T X - \overline{a^T X})(a^T X - \overline{a^T X})^T$
$= (a^T X - a^T \overline{X})(a^T X - a^T \overline{X})^T$
$= (a^T X - a^T \overline{X})(X^T a - \overline{X}^T a)$
$= a^T X X^T a - a^T X \overline{X}^T a - a^T \overline{X} X^T a + a^T \overline{X} \overline{X}^T a$
$= a^T (X X^T - X \overline{X}^T - \overline{X} X^T + \overline{X} \overline{X}^T) a$
$= a^T (X - \overline{X})(X - \overline{X})^T a$
$= a^T cov(X) a$

$\max(var(y_1)) = \max_a (a^T cov(X) a)$
subject to $a^T a = 1$
To solve this optimization problem a Lagrange multiplier α is introduced.

$l(u, \alpha) = a^T cov(X)a - \alpha(a^T a - 1)$
Differentiating with respect to a we have:
$Dif(a^T cov(X)a) = 2cov(X)a$
$Dif(a^T a) = 2a$
So:
$Dif(l) = 0 \iff cov(X)a = \alpha a$
a is an eigenvector of the cov(X) that has an eigenvalue α, if we premultiply both sides by a^T we have:
$a^T cov(X)a = a^T \alpha a = \alpha$
$a^T cov(X)a = var(a^T X)$ is maximum if α is maximum.

So the first eigenvector of cov(X) is the one that has the maximum variance of the data, the second eigenvector has the second maximum variance, and the third eigenvector has the minimum variance. The third one is also orthogonal to the first and the second eigenvectors. In other words, it is orthogonal to the plane that is composed of the first two eigenvectors.

The Figure 2.5 illustrates the resulted principal components on a 3D space. The green vector has as a direction the x-axis where it captured the most variance, the blue one points in the direction of y-axis where it has the second variance, and the red one points on the direction of the least variance 'z-axis'.
The first two principal components form the gray plane and the red one is perpendicular to the plane.

2.3.7 Normal vector of a point in a 3D cloud

The normal vector of a point in a 3D cloud is the normal vector of the surface defined by the point and its nearest neighbors. For a given set of data vectors x_i, suppose that all points are stacked into the columns of an $n \times t$ matrix X, where each column corresponds to an n-dimensional observation and there are t observations. We assume that u is the normal vector of the surface defined by these observations (set of points).

The normal vector can be obtained as the third projection vector returned by PCA, as explained in section 2.3.6. Here we rederive it from scratch. By doing the orthogonal projection of the points that belong to this surface on the normal vector u, we obtain the axis with the minimum variance. So, we have:

$$v = u^T X$$

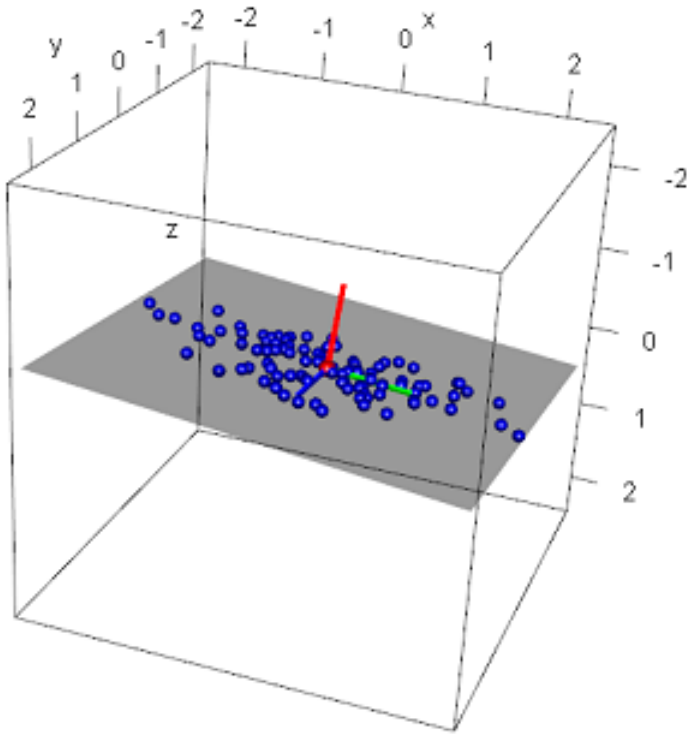

FIGURE 2.5: The normal vector of a surface [5].

$var(v) = var(u^T X) = u^T cov(X)u$
$\min_u(u^T cov(X)u)$
subject to $u^T u = 1$
To solve this optimization problem, we introduce a Lagrange multiplier α
$l(u, \alpha) = u^T cov(X)u - \alpha(u^T u - 1)$
Differentiating with respect to u we have:
$Dif(u^T cov(X)u) = 2cov(X)u$
$Dif(u^T u) = 2u$
So:
$Dif(l) = 0 \iff cov(X)u = \alpha u$
u is an eigenvector of the cov(x) that has an eigenvalue α, if we premultiply both sides by u^T we have: $u^T cov(X)u = u^T \alpha u = \alpha$
$u^T cov(X)u = var(u^T X)$ is minimum if α is minimum.

So the normal vector of the given surface is the eigenvector of cov(X) that has the least eigenvalue. With X is the data matrix of the points that define the target surface.

2.3.8 Perspective projection

- The perspective transformation gives the projection of one space to another. For example, the projection of a 3D scene into a 2D space.
- The perspective projection is mapping 3D points into 2D points.

The perspective transformation is presented in the Figure 2.6.

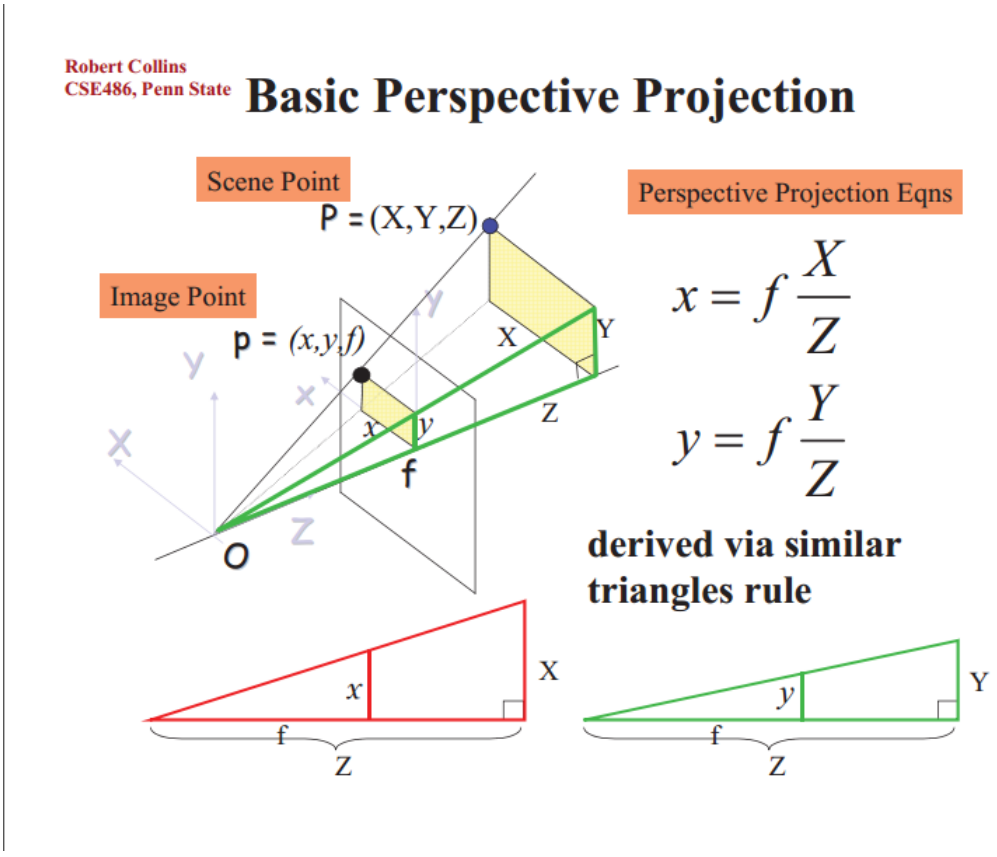

FIGURE 2.6: Perspective projection [6].

According to the figure above, we have a scene point P(X,Y,Z) and an image point p(x,y). We want to find (x,y) given (X,Y,Z) coordinates and the focal length f of a camera. We have:

- Two rectangle triangles: in the first one $x||X$ and in the second one $y||Y$.
- By applying Thales theorem we get:
  For the first triangle: f/Z=x/X $\iff$ x=f X/Z.
  For the second triangle: f/Z=y/Y $\iff$ y=f Y/Z.

2.4 Conclusion

Throughout this chapter, we explained the needed concepts related to 3D geometrics and descriptive statistics that we will use in the next chapters. The upcoming chapter is devoted to detailing the project specification and its general idea and presenting the sprints we have gone through.

Chapter 3

Methodology

3.1 Introduction

In this chapter, we detail the two most challenging parts of this work to achieve our final goal. As we said earlier, our main purpose is to build a robust obstacle detection method for a mobile robot by taking into account our available resources.
As a reminder, an obstacle is defined by the environment where the robot is navigating. It depends also on the robots' characteristics if it is giant or small. For example in a field a rock is not an obstacle for a big tractor. The purpose of this project is to build a computer vision system for a robot that enables it to avoid obstacles while accomplishing some tasks. As examples of obstacles, we can mention trees, animals, humans, walls, etc.

3.2 Motivation

Let's assume that the vehicle is moving on an (XY) plane. If it is on a safe area, it means "no obstacles". The Z component of the normal vectors along this surface will be very closer to one (unit vectors). It means that the normal vectors of safe areas will be approximately parallel to the Z-axis of the base (XYZ).

In contrast, if it is in a damaged area, it means "there are obstacles". The Z component of the normal vectors along this surface will be very lower than one (unit vectors). It means that the normal vectors of damaged areas will not be parallel to the Z-axis of the base (XYZ). This is shown in Figure 3.1.

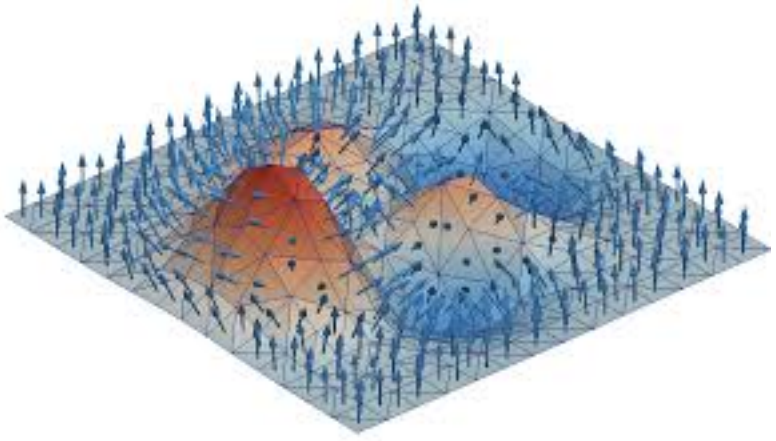

FIGURE 3.1: Normal vectors of a given huge surface [7].

Mathematically:
Assume that $\overrightarrow{N}$ is a normal vector of a given surface, with $\|\vec{N}\| = 1$. By projecting $\overrightarrow{N}$ in the original base (XYZ), we arrive to define mathematically an obstacle.

$$\vec{N} : \begin{bmatrix} nx \\ ny \\ nz \end{bmatrix} \tag{3.1}$$

$$\vec{X} : \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{3.2}$$

$$\vec{Y} : \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{3.3}$$

$$\vec{Z} : \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{3.4}$$

1.An obstacle:

$\vec{N} \cdot \vec{X} \neq 0$
$\vec{N} \cdot \vec{Y} \neq 0$

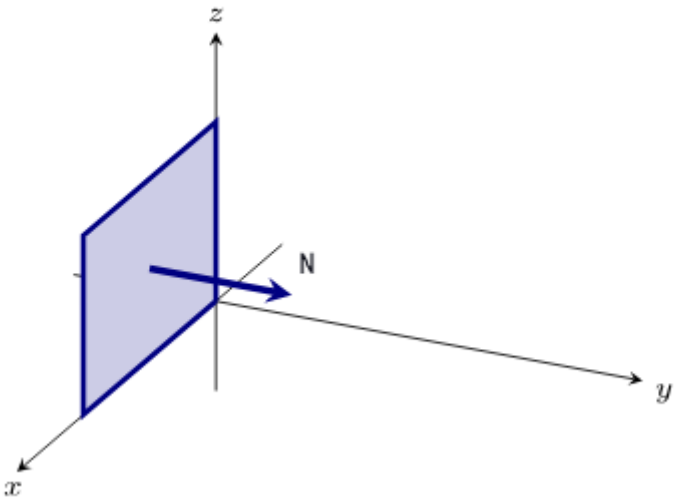

FIGURE 3.2: Normal vector of a given obstacle surface [8].

$\vec{N} \cdot \vec{Z} < t \iff nz < t$, with $t \in [0, 1]$ is a chosen threshold. An example of obstacle is shown in Figure 3.2.

2.Safe area:

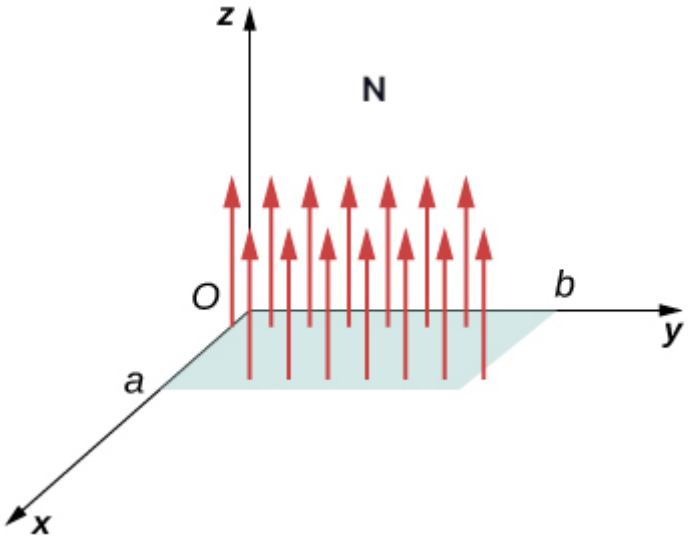

FIGURE 3.3: Normal vectors of a safe surface[9].

In this example $\vec{N} \cdot \vec{Z} = 1 \iff nz = 1$
In safe areas: $\vec{N} \cdot \vec{Z} \geq t \iff nz \geq t$, with $t \in [0, 1]$ is a chosen threshold. An example of a safe area is shown in Figure 3.3.

3.2.1 Methodology

To apply the previously discussed techniques, we need the 3D coordinates of each point in the presented scene. Then we can calculate the normal vectors and apply the decision rule-based obstacle detection technique. So two main questions need to be answered: How to acquire the 3D cloud of a scene? How to get the normal vectors of the 3D scene and use them to detect the obstacles in front of the robot?

3.2.1.1 Data acquisition

As we said earlier, the presented obstacle detection algorithm uses as input a 3D point cloud provided by a 3D sensor. The choice of the sensor is a critical and challenging task.
In this work, we will use a Kinect sensor because it is:

- A cheaper sensor compared to many others[24].
- Robust compared to a stereo vision system which depends on image contrast and needs powerful hardware to calculate the depth information.
- More precise compared to the Ultrasonic sensor which is generally used to detect obstacles at short ranges(ramps can be seen as obstacles, low height obstacles can not be detected).
- Commonly used in research.

Microsoft Kinect sensor system

In this subsection, we will initially present some concepts about the kinect device and its architecture [19].

Definition:

Microsoft Kinect was invented in 2010, it is used to provide an RGB-D image of a given scene. An RGB-D image is a combination of an RGB and a depth map.

RGB image: it is an image composed of three channels, red channel, green channel, and blue channel.

Depth map: the value of each pixel in the depth matrix is equal to the distance between each point of the scene and the lens of the sensor.

The architecture of the sensor:

It composed of the RGB camera, infrared(IR) emitter and IR depth camera, a tilt motor that enables us to change the vertical angle of the sensor, and a microphone array. With these components we will be able of capturing the color and the depth information of each pixel of the given scene.

- The microphone array: it is used for detecting the position of a voice source. The microphone array was not included in this work.
- Depth camera: this camera provides an image, where each pixel value is contained the depth information. the Kinect sensor uses an IR camera instead of a stereo camera system.

The Figure 3.4 illustrates the architecture of a Kinect sensor.

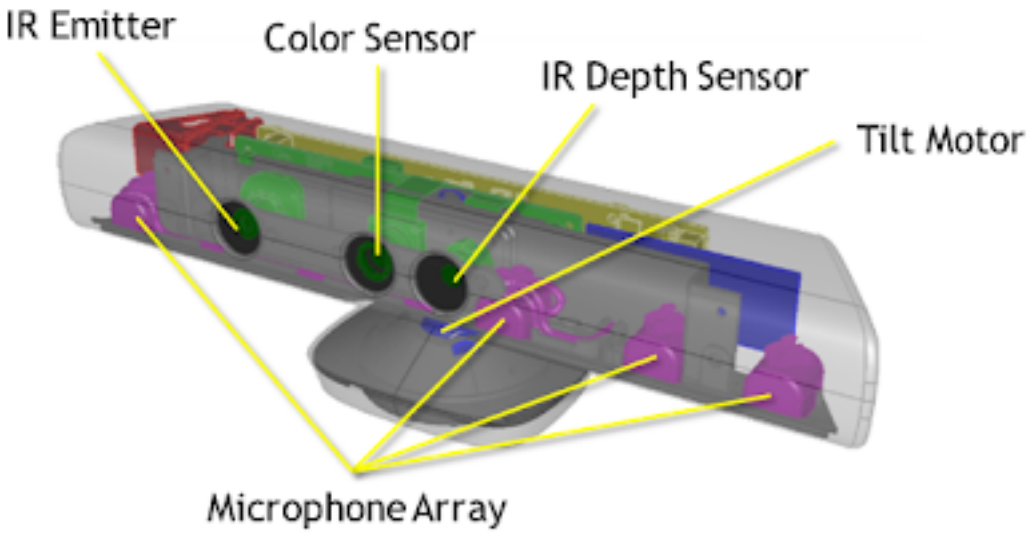

FIGURE 3.4: Kinect sensor [10].

Why the use of IR light?

The Kinect sensor uses the infrared light because it reduces the risks of the signal interference. Furthermore, it can be distinguished easily from the ambient light.

Uses of the Kinect sensor:

It can be used in several areas like games and robotics in both industry and research.

Versions of the Kinect sensor:

We have in total two versions of the Microsoft Kinect sensor depending on the principle of measuring the depth values:

VERSION 1: The first version is very old. It is based on the structured light range sensing. In fact, given a sequence of known patterns that are sequentially projected over an element from the scene, which will be deformed by the geometric of the object. The object is then observed from a camera from a different direction. By doing some analysis, the depth information can be given by[25]:
$d = (b \times f)/m(x, y)$
with:

- f= the focal length of the camera.
- b= the distance between the camera and the projector.
- m(x,y)= the disparity value between the original patterns and the deformed ones.

The Figure 3.5 shows tha Kinect sensor version 1.

FIGURE 3.5: Kinect sensor version 1 [11].

The Figure 3.6 shows the structured light sensing principle.

VERSION 2: It uses the time of flight principle which is based on the difference of time between the emission task of the IR signal and its return to the sensor after being reflected by the object. Given the speed information of the light, the depth value is given by: $d = s.t$
with:

- s= The speed of the light.
- t= The time between the emission and the reception of the light.

The Figure 3.7 shows an example of a Kinect sensor version 2.

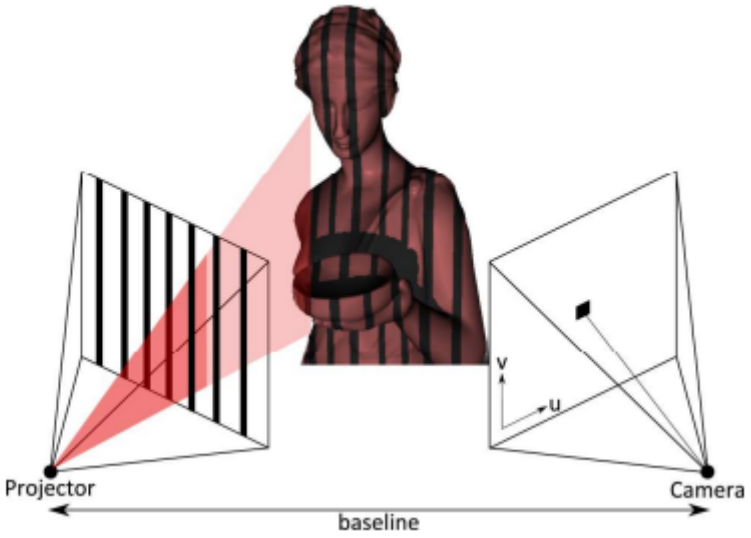

FIGURE 3.6: SL principle [12].

FIGURE 3.7: Kinect sensor version 2 [13]

The Figure 3.8 shows the Time of Flight sensing principle. The sensor sends a signal and waits until it receives it back. Using the difference in time, it measures the distance.

Comparison between Kinect v1 and Kinect v2:

The Table 3.1 below presents a comparison between Kinect v1 and Kinect v2.

We aim to tackle this problem using a kinect v2 RGB-D sensor. It has a robust data acquisition principle. It can also work well indoor and outdoor. So how to generate the 3D points cloud from an RGB-D image?

3D point cloud generation

Based on the perspective projection, which was discussed in the previous chapter "Section 2.3.8", we can get the 3D coordinates of each pixel in the depth map. The 3D coordinates give the true physical location of each pixel in the real world.

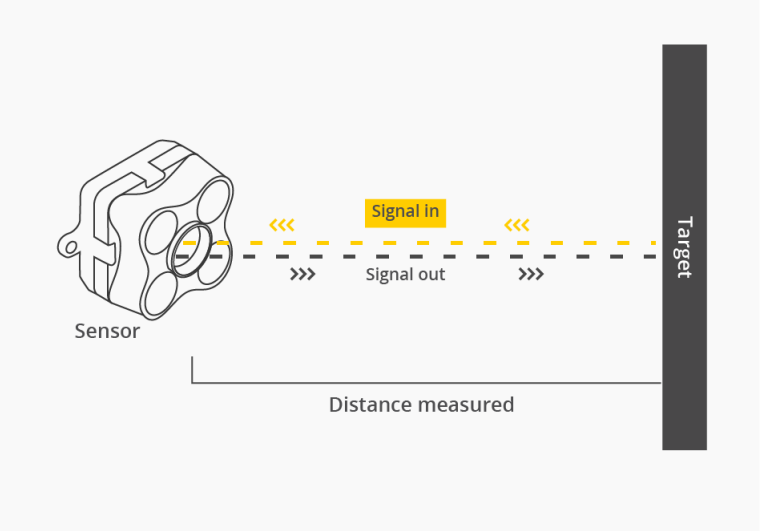

FIGURE 3.8: TOF principle [14].

Features	version2	version1
Depth sensing technology	Time of flight	Structured light
Wide-angle field of view lens	70° horizontally and 60° vertically	57° horizontally and 43° vertically
Minimum distance for obtaining depth readings	1.37 m	2.4 m
Working conditions	Indoor	Indoor and outdoor
Color camera resolution	1920*1080	640*480
Range of operation	0.8 m to 4 m	0.5 m to 4.5 m

TABLE 3.1: Kinect v1 VS kinect v2.

FIGURE 3.9: Microsoft Kinect v2 showing the orientation of the coordinate system [15].

To do this calculation, in addition to the depth values of each point, we need the intrinsic parameters of the depth camera of the Kinect sensor, which are the focal length of the camera and (centerX, centerY). These two centers define the position of the focal center in the referential (XYZ).

Given the orientation of the axes X, Y, and Z in the previous Figure 3.9, the coordinates x, y, and z of the point $A(x, y, z)$ corresponding to each pixel $p(i, j)$ in the depth image is given by:

$$x = ((j - centerX)p(i,j))/focallength$$

$$y = ((i - centerY)p(i,j))/focallength$$

$$z = p(i,j)$$

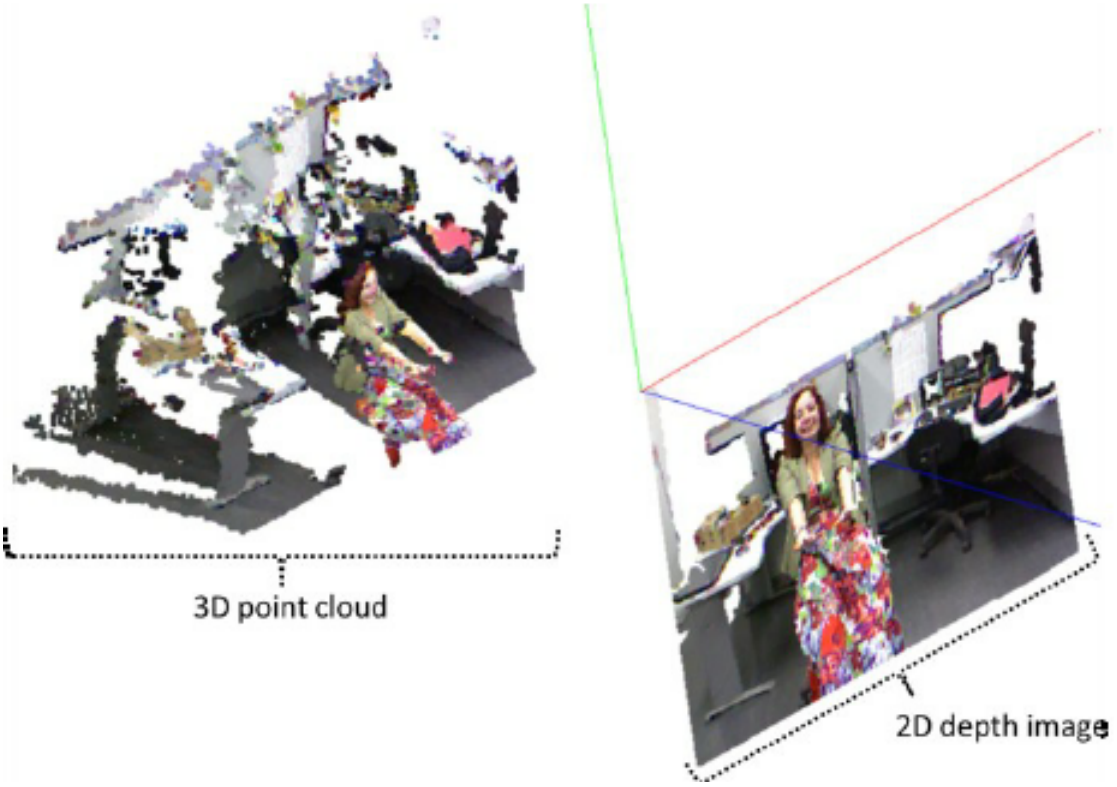

FIGURE 3.10: Depth image and 3D point cloud of the same scene [16].

3.2.1.2 Obstacele Detection

We follow a similar approach to the one described in[24]. This technique is called Normal-based plane segmentation. The segments will be obtained based on the normal vectors of the points in the 3D cloud. Each point will be defined by its coordinates in the 3D space (XYZ) and we also attribute to it the normal vector of the plane defined by its nearest neighbors.

As discussed in the previous chapter "Section 2.2.3", a plane can be defined at least by 3 points. So the considered point and its two nearest neighbors will be enough to generate such a plane. But to be more accurate and to avoid noisy points we can use more neighbors. So How to calculate the normal vector of each point in the cloud?

Normal vector of a point in the cloud:

According to the previous chapter "section 2.3.7", the normal vector of an elementary surface is defined as the eigenvector of the covariance matrix of the given point and its nearest neighbors that has the least eigenvalue.

Obstacle identification:

We got all the unit normal vectors of the 3D points cloud. Each normal vector is defined by 3 coordinates: The first is the projection of the normal vector on X (nx), the second is the projection of the normal vector on Y (ny), and the third is the projection of the normal vector on Z (nz).
So how to use them to detect the points that correspond to obstacles?

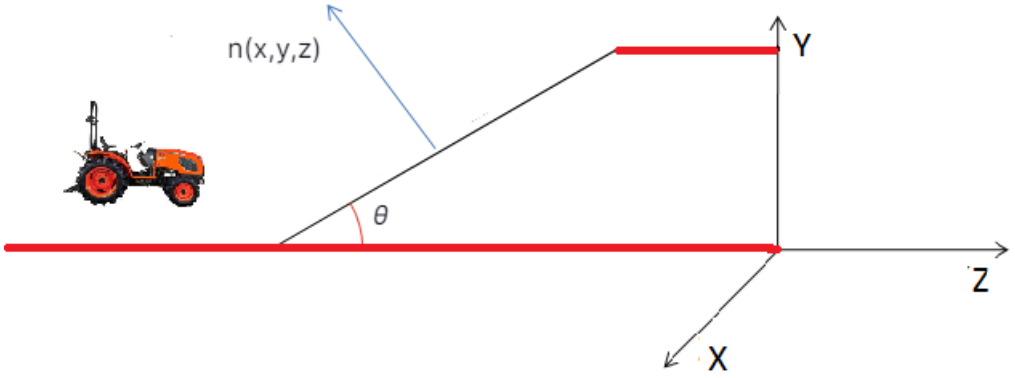

FIGURE 3.11: Principle of navigation of a robot.

When a robot is navigating, it can face ramps. Let assume that θ is the maximum ramp angle that the robot can climb. As we can see in the Figure 3.11 above the angle between the normal vector and Y-axis is obtained as follows:

- $\langle \vec{Z}, \vec{n} \rangle = \pi - (\theta + \pi/2) = \pi/2 - \theta.$
- $\langle \vec{Z}, \vec{Y} \rangle = \pi/2.$
- $\langle \vec{Z}, \vec{Y} \rangle = \langle \vec{Z}, \vec{n} \rangle + \langle \vec{n}, \vec{Y} \rangle \iff \langle \vec{n}, \vec{Y} \rangle = \pi/2 - (\pi/2 - \theta) = \theta.$

By projecting the normal vector (unit vector) on the vertical axis Y we get:
$ny = \vec{n} \cdot \vec{Y} = \|n\| \times \|Y\| \times \cos(\theta) = \cos(\theta)$

Later on obstacles will be identified as follows:

- All the points that have normal vectors with Y-component under $\cos(\theta)$ belong to the obstacles areas. Otherwise, they belong to safe areas.
- θ is a threshold depends on the robot and the sol characteristics. It will be chosen by trial and error. For example:
  - For smooth ground like in "labs, rooms, factories, θ should be small.
  - For non-smooth ground with holes and ramps like in a "farming environments", θ should be high.

3.2.2 Algorithm

The proposed algorithm, which was implemented during this work is shown below. We will go in depth through its steps in the next chapter:

Algorithm 1 Obstacles detection algorithm

```
Depth ← preprocess the raw depth image using median filter.
3D point cloud generation ← RGB + Depth (Perspective Projection)
PCloud ← 3D point cloud
PCloud1 ← PCloud ∩ admissible 3D region to robot
PCloud2 ← Filtered PCloud1 (3D voxel grid filtering)
Estimate normals of PCloud2 (using N neighbors and applying PCA technique)
PCobstacles ← ∅
N1 ← The number of neighbors used for the validation process
Neighbors ← The neighbors coordinates
θ ← Maximum angle of passable slopes
for all p_i ∈ PCloud2 do
    y_i ← Y component of the normal vector of p_i
    if |y_i| ≤ cos(θ) then
        n ← 0
        for all p_j ∈ Neighbors(p_i) do
            y_j ← Y component of the normal vector of p_j
            if |y_j| ≤ cos(θ) then
                n ← n + 1
        if n > N1/2 then
            PCobstacles ← PCobstacles ∪ p_i
Costmap ← the orthogonal projection of the PCobstacles on the (XZ) plane
return PCobstacales, Costmap
```

3.3 Conclusion

During this chapter, we presented an approach for detecting obstacles. This approach is efficient and suitable for real-time and its computational complexity is linear to the number of the cloud points.

Chapter 4

Experimental set-up, Results, and Discussions

4.1 Introduction

In this last chapter, we start by talking about the used dataset. Then we will go through the various parts of the algorithm. later we give and discuss the obtained results.

4.2 Datasets

In the experimental part, we used examples from the two following datasets:

- RGB-D images: DIML/CVLAB RGB+D Dataset [17].
- RGB-D videos: An Open Benchmark Corpus for mobile RGB-D Related Algorithms [18].

4.2.1 RGB-D images: DIML/CVLAB RGB+D Dataset

This dataset is composed of various RGB and depth images of indoor and outdoor scenes. The indoor scenes dataset was captured by a Kinect v2.0 sensor (TOF principle), while the outdoor dataset was obtained by stereo cameras. In this work, we are interested in the Kinect RGB-D dataset. The images resolutions are:

- Color image resolution: 1920 x 1080.

- Depth image resolution: 480 x 270.

Some of the images samples are shown in the Figure 4.1 below. In the depth images, the dark pixels are the points of the object that are near to the camera and the bright pixels are the farthest points from the camera.

FIGURE 4.1: Sample data from the DIML CVLAB RGB-D Dataset [17].

4.2.2 RGB-D video: An Open Benchmark Corpus for mobile RGB-D Related Algorithms

This dataset is composed of RGB and depth videos of various indoor scenes. It was captured by a Kinect v2.0 sensor. In this work, we used a corridor videos with the

following characteristics:

- Color frame resolution: 1920 x 1920.
- Depth frame resolution: 512 x 424.
- Number of frames per second: 30 frame per second.

The video samples are presented in the Figure 4.2 below.

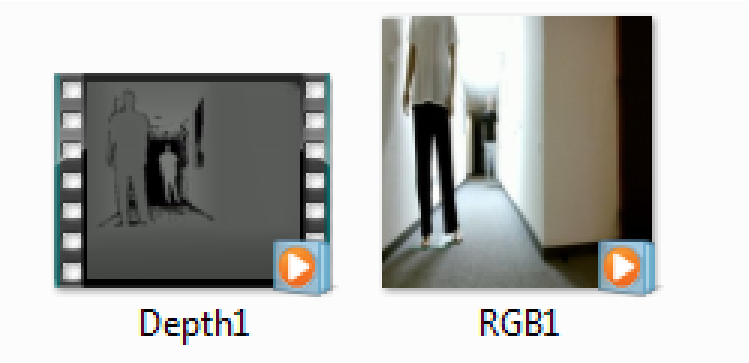

FIGURE 4.2: Sample data from the Open Benchmark Corpus for mobile RGB-D Related Algorithms Dataset [18].

4.3 Implementation

After getting the dataset, we will detail the implementation process step by step. Through that, we will deepen by knowledge in the various parts of the approach.

4.3.1 Depth enhancement

After discovering and visualizing the data, we notice that the depth images provided by the Kinect sensor present many blind spots, and that influences the result. We mean by blind spots places where the Kinect is unable to provide the depth information, the Figure 4.3 below shows an example. This phenomenon may be caused by the intersection of the light "eg. IR" and the object materiel, or because this object is very far away from the camera.

Original depth images

Here we introduce two examples from our dataset in the Figures 4.4 and 4.5.

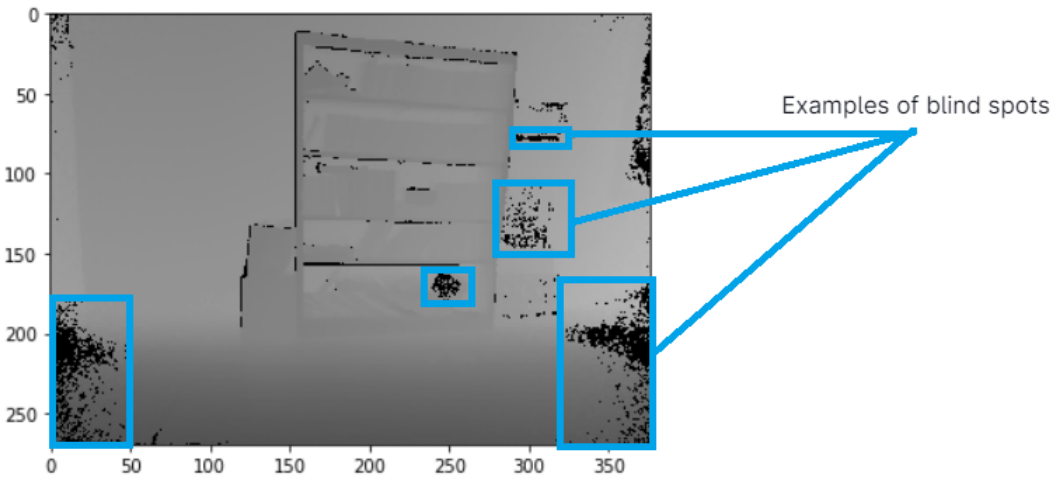

FIGURE 4.3: Examples of blind spots.

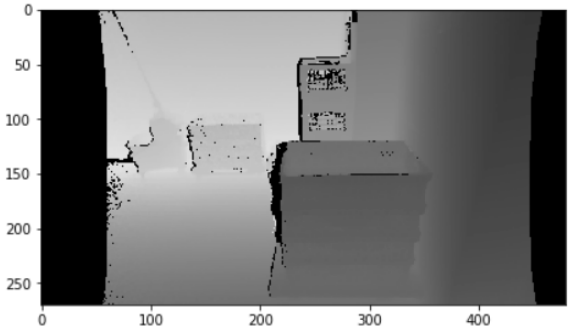

FIGURE 4.4: Original depth image 1.

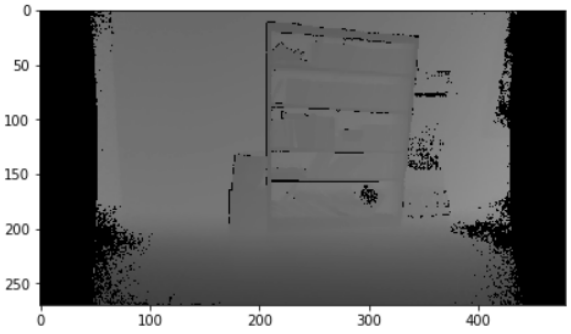

FIGURE 4.5: Original depth image 2.

As we can see the images present a pepper noise in the middle and we can also notice the problem of the borders which can be caused by camera calibration problems, or by low illumination at the image border (due to shadowing of the infrared emitter [26]).

Cropping step

For the borders which are not including the object of interest, we decided to apply a cropping technique. The result is shown in Figures 4.6 and 4.7 below.

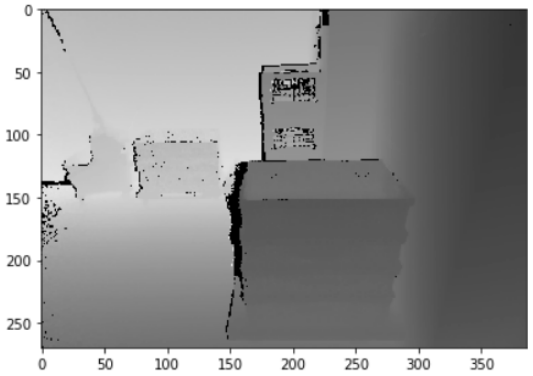

FIGURE 4.6: Cropped depth image 1

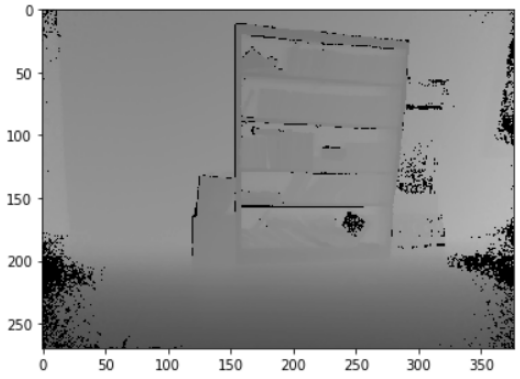

FIGURE 4.7: Cropped depth image 2.

Filtering step

For the pepper noise, we decided to apply a median filter with size K $\times K$. K is an odd integer and should be chosen wisely by trial. The value of K depends on the image. The result is shown in Figures 4.8 and 4.9 below.

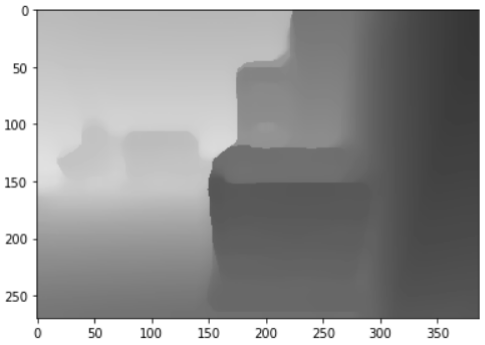

FIGURE 4.8: Filtered depth image 1, K=17

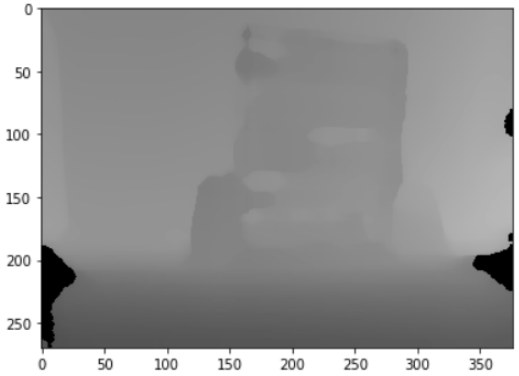

FIGURE 4.9: Filtered depth image 2, K=9

4.3.1.1 Second dataset

For this dataset, we will give a frame example 4.10.

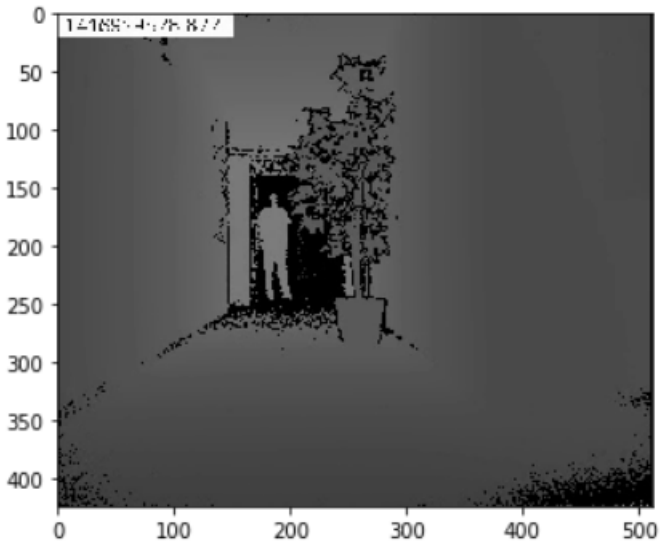

FIGURE 4.10: Original depth frame.

Resulted depth frame

The resulted image in 4.11 is the output of a median filter of size 7×7 applied to the input image.

4.3.2 Point cloud generation

In this stage, the obtained depth map will be converted into 3D point clouds. We will get this 3D representation based on the intrinsic parameters of the camera and the depth values. The technique that enables us to perform this transformation is the perspective projection explained in the second chapter 2.3.8.

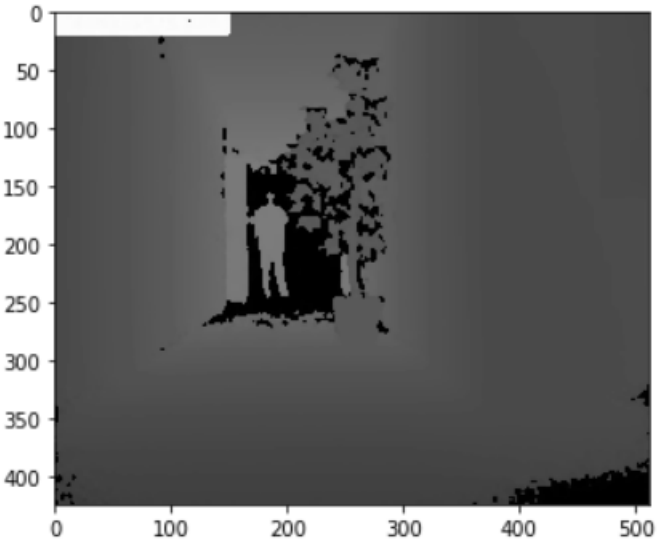

FIGURE 4.11: Filtered depth frame.

Given a pixel p(i,j) located in (i,j) from the depth image, its corresponding point p(x,y,z) coordinates in the referential (X,Y,Z) from the real 3D scene are:

$$z = p(i, j)$$

$$x = (j - centerx)z/f$$

$$y = (i - centery)z/f$$

With centerx, centery, and f are the intrinsic parameters of the Kinect. These three values were chosen based on the result of a Kinect calibration process from other works [27, 28] as follows in the Table 4.1.

Parameter	value (pixels)
Focal length "f"	580.0
Centerx	200.0
Centery	130.0

TABLE 4.1: Intrinsic parameters for the Kinect v2.0 sensor.

Examples of 3D point clouds representations

In the Figures 4.12, 4.13, and 4.14 we show the resulted cloud from a given RGB and depth information. These clouds are obtained using depth and RGB images and based on the perspective projection.

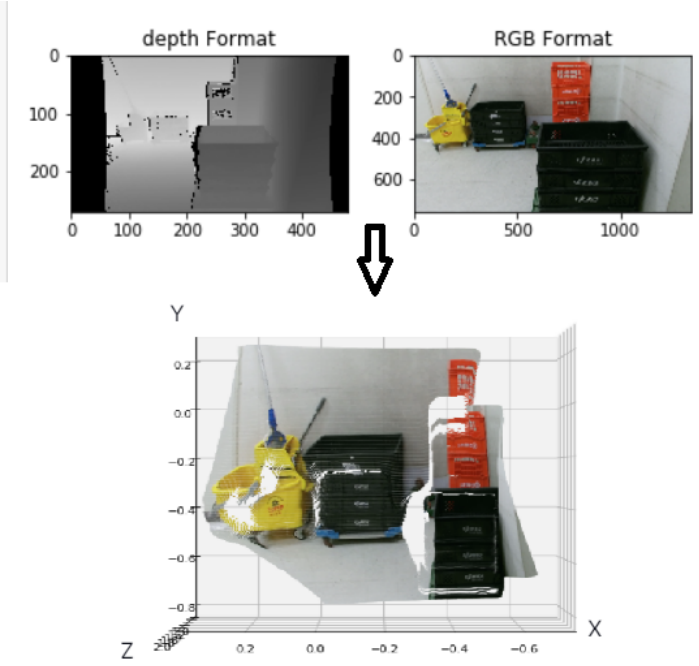

FIGURE 4.12: 3D cloud generated from RGB and depth images.

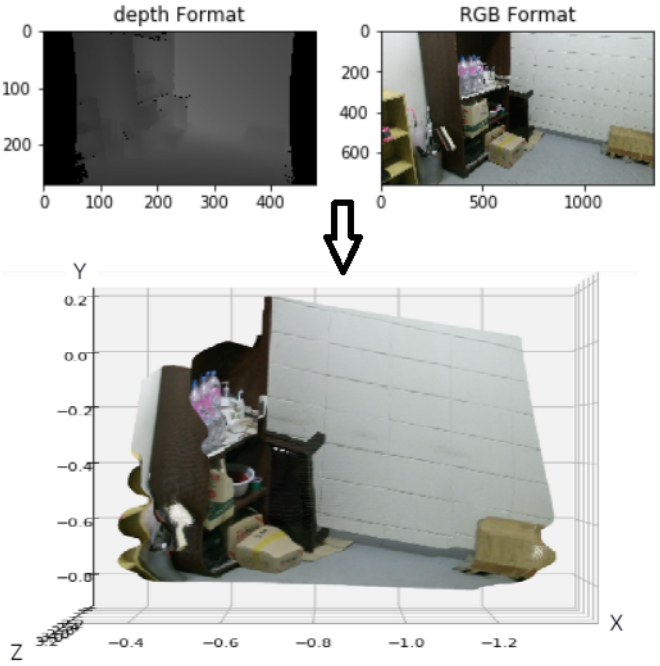

FIGURE 4.13: 3D cloud generated from RGB and depth images.

4.3.3 Determining the region of interest

After getting the 3D point clouds of the target scene, we will now eliminate the useless regions. The purpose of this task is to decrease the computational complexity of the program.

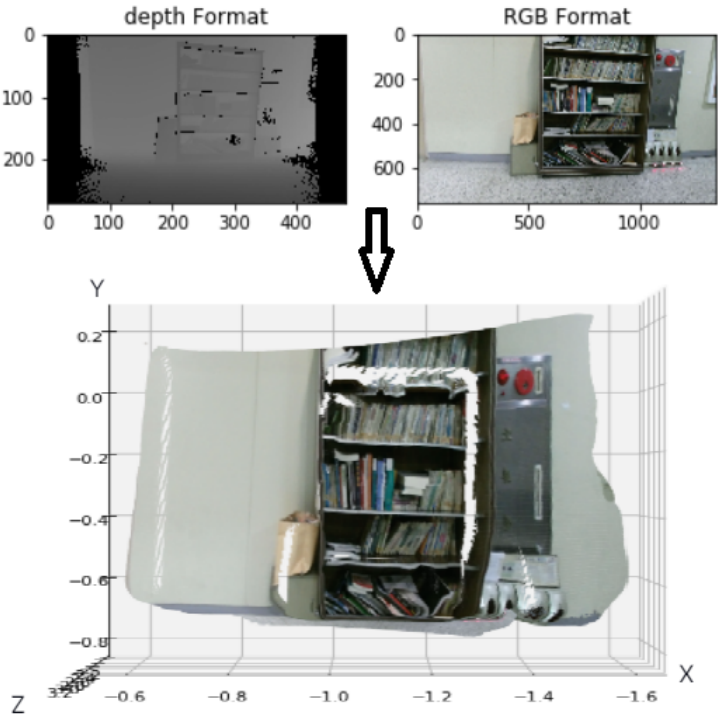

FIGURE 4.14: 3D cloud generated from RGB and depth images.

These useless regions are:

- The points that are below the ground. Just a $-\alpha$ meters among Y is allowed as a safety margin (for negative ramps). α depends on the ground characteristics. For instance, if we are on a farm α should take a high value. But if we are on a lab it should take a small value because no ramps there. In this work, α=0.1.
- The points that are above the sensor by β meters because all the obstacles are expected to be touching the ground. β depends on the position of the sensor on the robot, so it depends on the robot characteristics (robot length). In this work, β=0.8.
- The points that are far away from the robot. For example we can let the robot just see 2m as depth in front of it.

The Figure 4.15 illustrates the previous points.

4.3.4 Applying 3D voxel grid filter

After getting the target cloud, now we will go through another important step. To reduce the computational complexity of the algorithm and to remove outliers, we will

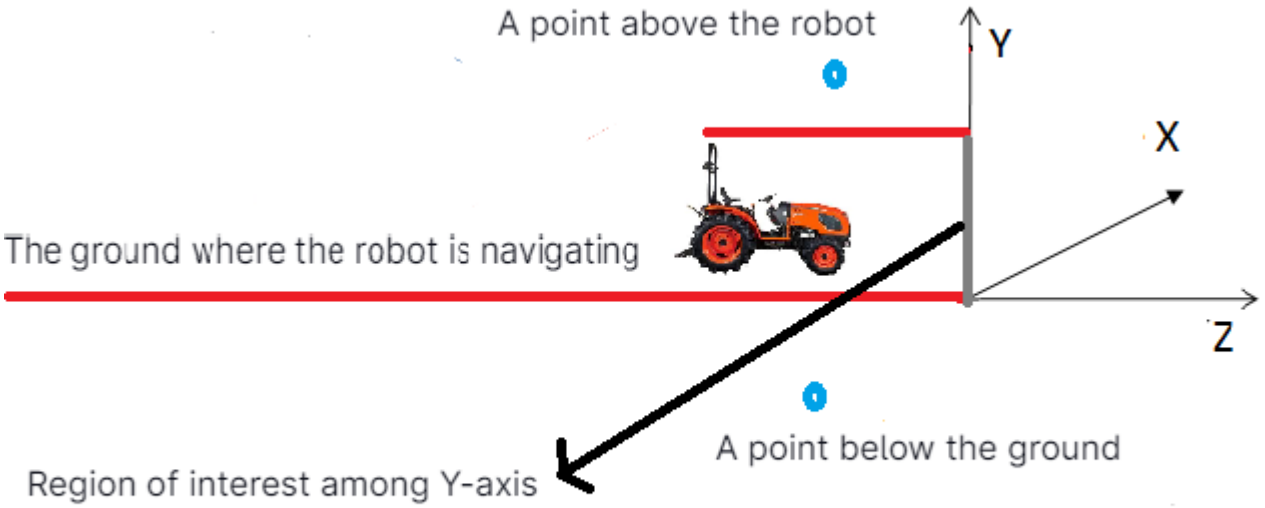

FIGURE 4.15: region of interest for the robot.

apply a filtering technique.
The used filter is named 3D voxel grid filtering. It gives a new representation of a 3D points cloud with conserving all the characteristics of the input data. This representation has the same format as the input "cloud" but with less number of points.
It transforms points cloud to voxels (cubic shape) by taking into account the voxelgrid's parameters "Vx,Vy,Vz".

This filter works this way:

1. Dividing the space of the cloud into a regular grid of cubes"voxels" with length equal to Vy, width equal to Vx, and depth equal to Vz. These parameters are the cube size along each dimension.
   The optimal found dimensions of the cube are (25mm,25mm,25mm).

2. The points that are in the same voxel will be represented by one point whose coordinates are:

$$x_c = 1/N \sum_{i=1}^{N} x_i$$

$$y_c = 1/N \sum_{i=1}^{N} y_i$$

$$z_c = 1/N \sum_{i=1}^{N} z_i$$

The Figure 4.16 illustrates the various previous steps.

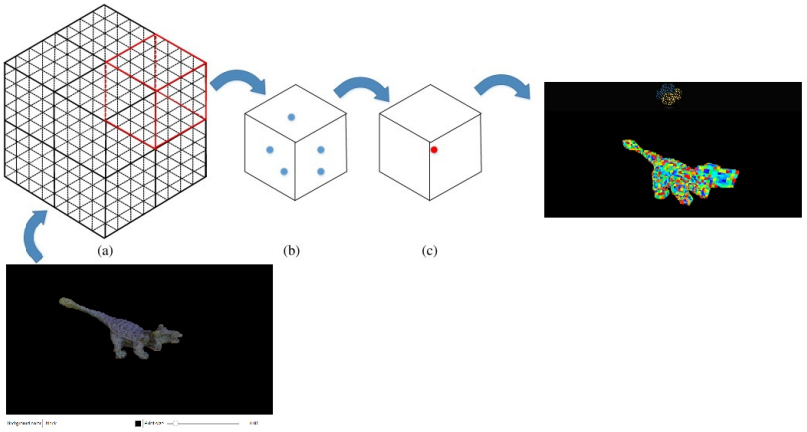

FIGURE 4.16: Voxel grid filter step by step with (a) is 1), (b) and (c) are 2).

The Figure 4.17 represents the output of a 3D voxel grid filter.

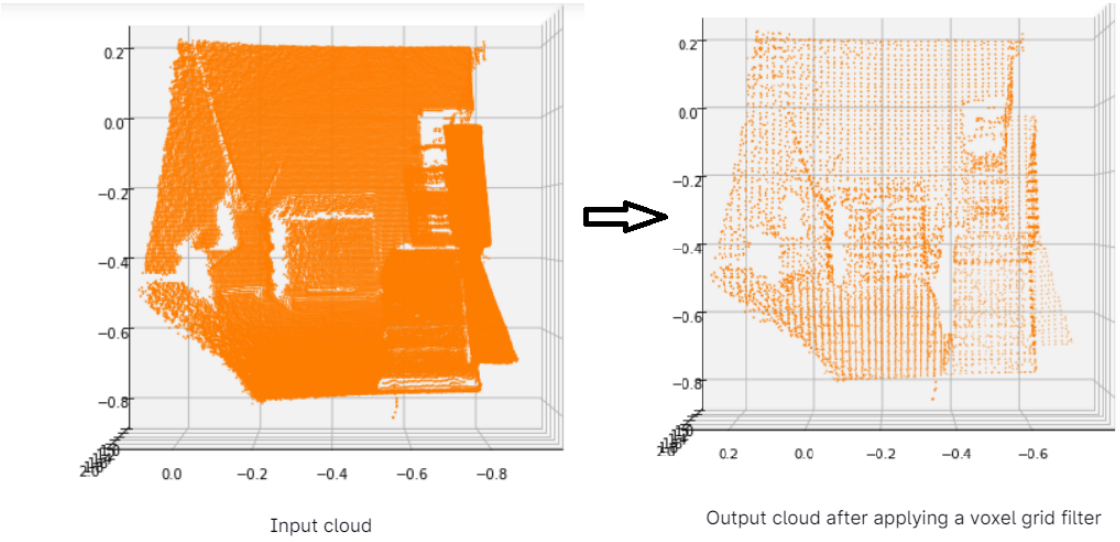

FIGURE 4.17: The output of a voxelgrid filter.

4.3.5 Normal-based plane segmentation

Determining the normal vectors of the points in the cloud

Once we down-sampled the cloud (to reduce the computational complexity of the algorithm), it is time to calculate the normal vectors of each point. As we discussed previously, we assume that the robot is navigating in a (XZ) plane:

- The normal vector of a point is the normal vector of the plane defined by its nearest neighbors.
- The normal vector is the eigenvector of the covariance matrix of the point's neighbors that has the least eigenvalue.
- For each point p_i, the covariance matrix C is:

$$C = 1/N(\sum_{n=1}^{N}(p_n - \overline{p})(p_n - \overline{p})^T)$$

with $\overline{p}$ is the mean of the neighbors of the point p_i and N is the number of the neighbors of the point p_i.

The Figure 4.18 below shows the output for different N values (N=3, N=10, and N=400).

- For N=3, we got many wrong points.
- For N=10, we got a better result compared to the first one.
- For N=400, we got wrong points again.

N=10, is the best choice.

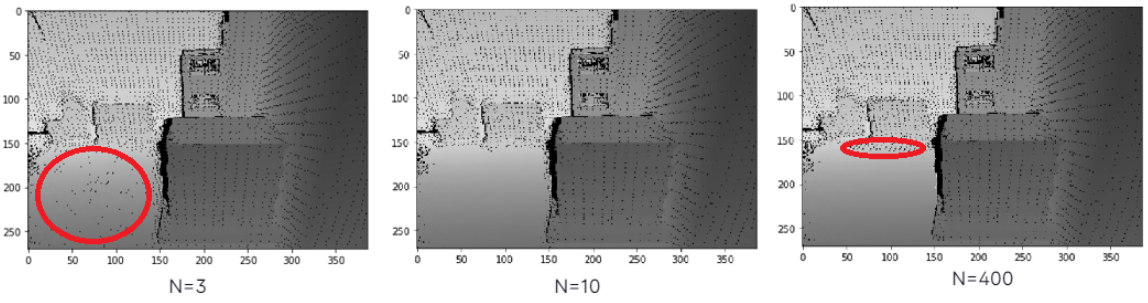

FIGURE 4.18: the depth output for different N. Obstacles are market by black dots.

Obstacles identification

Now we have all the normal vectors. Assume that θ is the maximum angle that the robot can climb.

- If the absolute value of the inner product between the normal vector and the vertical axis Y is over than $\cos(\theta)$: we have accessible pixels.
- If the absolute value of the inner product between the normal vector and the vertical axis Y is less than $\cos(\theta)$: we have obstacles.

The Figure 4.19 below shows the output for different angles θ to perform the previous thresholding technique (θ=π/12, θ=π/6, and θ=π/4).

- For θ=π/12, we got many extra wrong points.
- For θ=π/6, we got a better result and all the obstacles are detected correctly.
- If we increase θ many obstacles points will be missed, as illustrated in the figure below (θ=π/4).

θ=π/6, is the best choice.

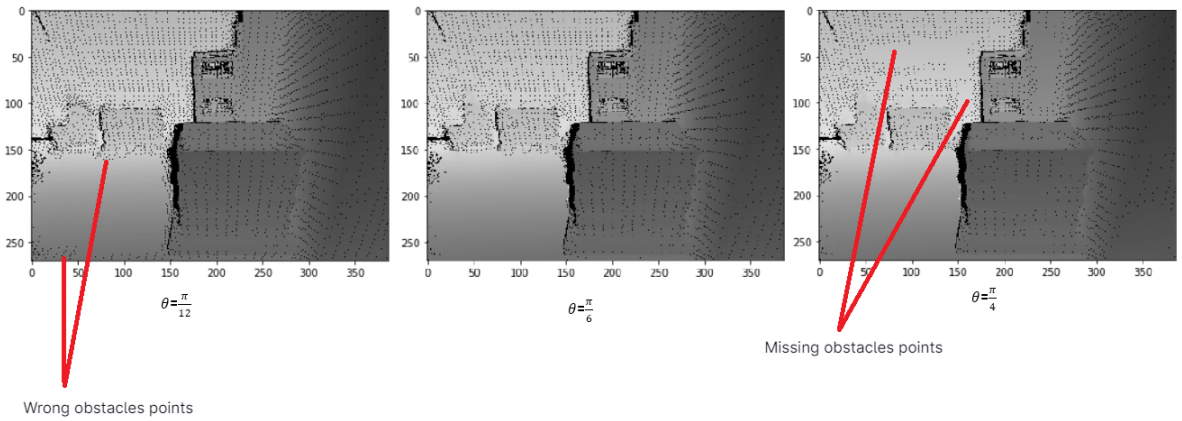

FIGURE 4.19: the depth output for different angles θ. Obstacles are market by black dots.

After getting the points that correspond to obstacles in the scene, we will go a step further to validate our result.
So we will make sure that the neighbors of a non-navigable point are also impassable. This step enables us to discard the wrong pixels.
Suppose that a point p_i is impassable and has N1 neighbors. If all its neighbors are impassable points too (the projection of their normal vectors into the Y-axis will be less than cos(θ)), so it will be still considered as an obstacle else it will be removed from the obstacle points.

The Figure 4.20 below shows the output after validating the obtained obstacles points.

4.3.6 Costmap generation

In this step, we will generate a costmap based on the output of the previous step. In fact, given the 3D points cloud of the obstacles in the scene, the costmap is a 2D grid

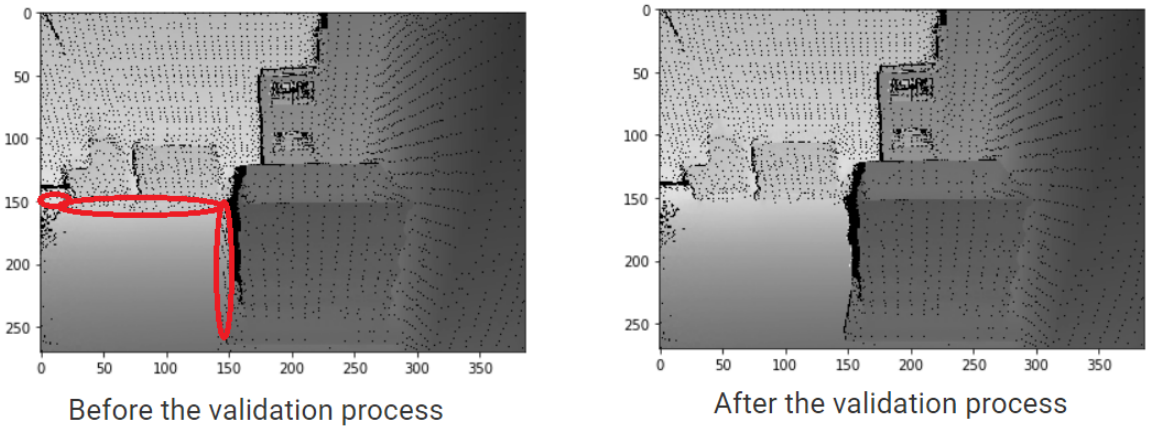

FIGURE 4.20: the output after the validation process. Obstacles are market by black dots.

representation where obstacles points are orthogonally projected on the plane (XZ). this map enables the robot to distinguish between navigable routes and non-navigable ones.

Mathematically, each point $p(x, y, z)$ in the cloud will be represented by $p(x, z)$ in the costmap. The Figure 4.21 below is an example of a costmap of the example in the figure 4.13.

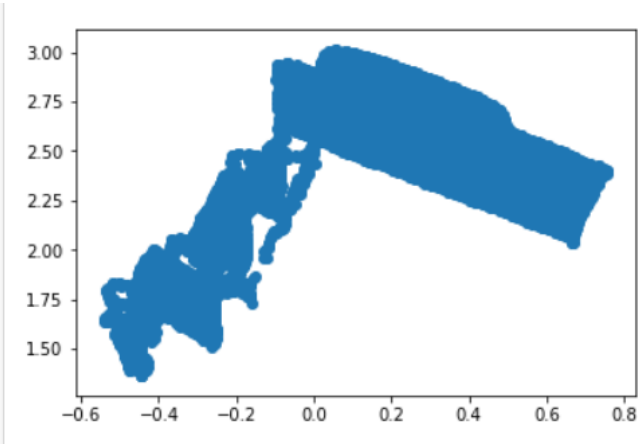

FIGURE 4.21: The blue area is a non-navigable area in the plane (XZ).

4.4 Results and discussions

4.4.1 Dataset 1

4.4.1.1 Example 1

In the Figure 4.22, we notice that:

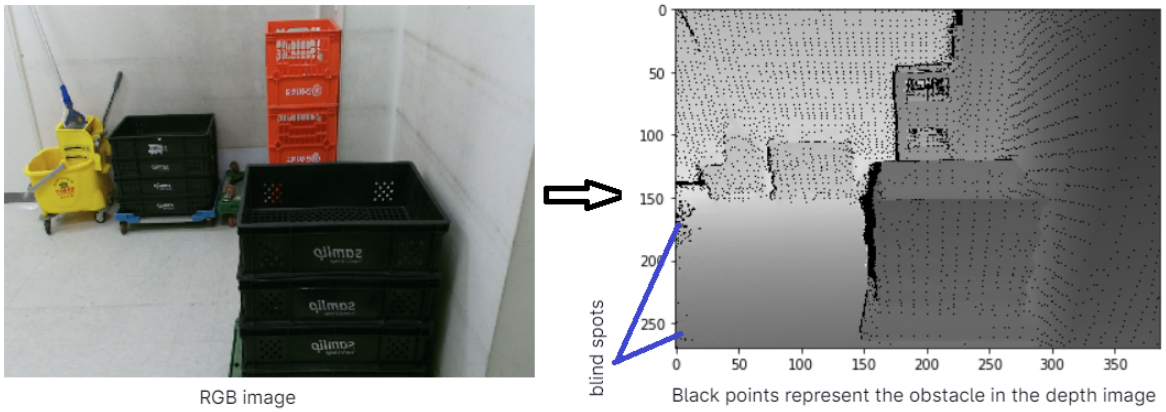

FIGURE 4.22: Obstacles are market by black dots in the depth image except the blind spots.

- Obstacles in the RGB image are walls, black boxes, the red box, and the yellow box.
- In the depth image, the black dots represent the obstacles.

All the obstacles in the image are detected. The black parts represent the blind spots in the depth image.

After visualizing the obstacles, now we provide its corresponding costmap, as shown in the Figure 4.23. This map represents the impassable areas of the plane where the robot is navigating. It enables the robot to distinguish between safe areas and obstacles.

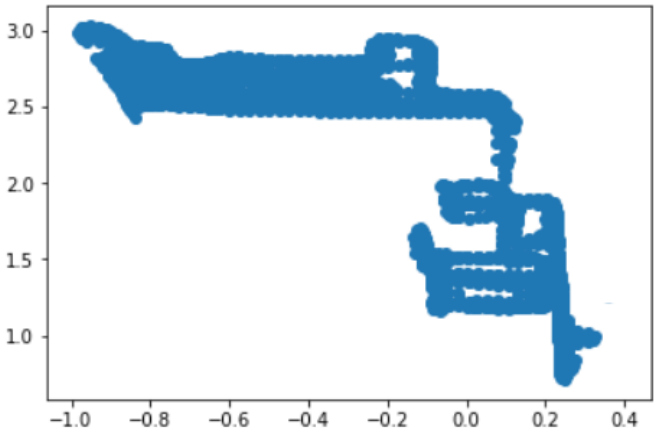

FIGURE 4.23: Costmap of the previous scene.

4.4.1.2 Example 2

In the Figure 4.24, we notice that all the obstacles points are detected correctly.

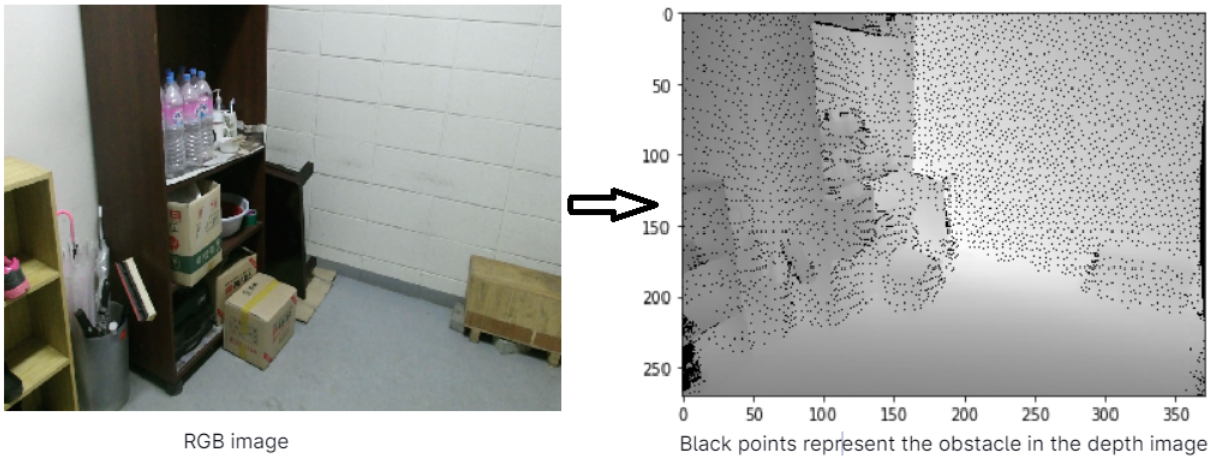

FIGURE 4.24: An image and its corresponding output. Obstacles are market by black dots in the depth image.

In the Figure 4.25, we visualize the corresponding costmap.

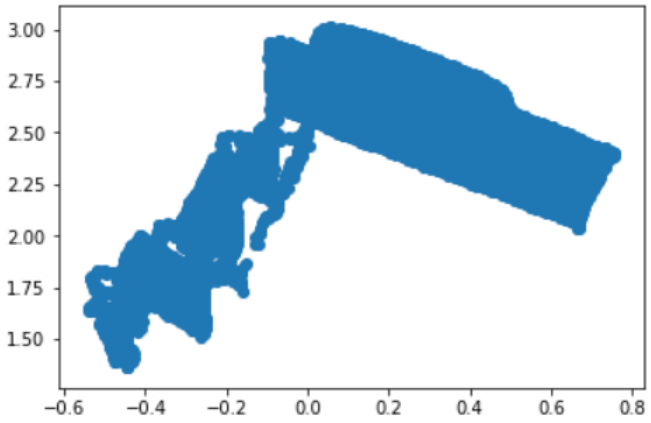

FIGURE 4.25: Costmap of the previous output.

In both examples, the parameters and their values are given in the Table 4.2. The parameters are:

- θ: is the maximum angle of passable slopes.
- N: is the number of neighbors used to determine the normal vector of each point in the 3D cloud.
- N1: is the number of the nearest neighbors of a point used to discard the outliers "validation process".
- TDBG: is the tolerant distance below the ground "in case there are negative ramps".

- SMAR: is the security marge above robot.

Parameter	θ	N1	N	TDBG	SMAR
Value	$\pi/6$	12	10	-0.1	0.8

TABLE 4.2: Optimal parameters for the first scenario.

4.4.2 Dataset 2

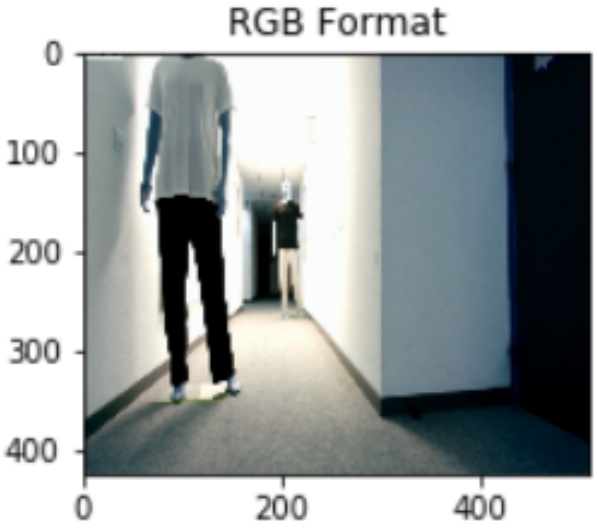

FIGURE 4.26: RGB frame

In the Figure 4.27, we notice that all the obstacles points "walls and the man" are detected correctly. In this frame, the robot sees only 1m as depth (Z $\leq$ 1m).

FIGURE 4.27: Black points represent obstacle on the image.

For all the video frames, the parameters and their corresponding values are given in the Table 4.3. The parameters are:

- θ: is the maximum angle of passable slopes.

- N: is the number of neighbors used to determine the normal vector of each point in the 3D cloud.
- N1: is the number of the nearest neighbors of the point used to discard the outliers "validation process".
- TDBG: is the tolerant distance below the ground "in case there are negative ramps".
- SMAR: is the security marge above robot.

Parameter	θ	N1	N	TDBG	SMAR
Value	$\pi/3$	7	5	-0.1	0.8

TABLE 4.3: Optimal parameters for the second scenario.

These parameters are optimized for each environment. We tried many values and chose the ones that gave the best result.

4.5 improvements

4.5.1 Data preparation

- Form a set of pixels which are obstacles: $S_t = (x, y, s), s < threshold1, s = cos(Z, Y)$ and t is the frame number.
- Form a set of pixels which are safe points: $S_t = (x, y, s), s > threshold1, s = cos(Z, Y)$, and t is the frame number.
- Use the Sobel filter as an edge detector algorithm and apply it to the depth image.

4.5.2 Logistic regression

The random variables p_t and g_t represent the slope and the gradient. Let$V_{t-1} = (x-1, y-1, t-1), (x-1, y, t-1), (x, y-1, t-1), (x, y, t-1), (x+1, y, t-1), (x, y+1, t-1), (x+1$ the N neighbors of pixels (x, y, t). With the neighbor pixel number k is associated a variable e_k; which is equal to 1 if the pixel is an obstacle and 0 otherwise. The values of the variables e_k have been calculated before. We define the dependent variable y_t

$$y_t = a_0 + a_1 s_t + a_2 c_t + b_1 e_1 + b_2 e_2 + \ldots + b_N e_n$$

with a_k and b_k real numbers, c and s are the gradient and the slope.

4.5.3 Learning

Considering the data S_t and C_t, we need to calculate the parameters a and b using the maximum likelihood.
The logistic regression algorithm will estimate the parameters of this binary model to classify pixels as obstacle points or non-obstacle points based on the information of the gradient, the slope, and the neighbors status.

4.5.4 Result

Accuracy classification score:
In binary classification, this function computes subset accuracy: the set of labels predicted for a sample must exactly match the corresponding set of labels in y_{true}. the resulted accuracy metric value with the constraint that the model is trained on 10 frames is equal to 0.83.

4.6 Conclusion

In this chapter, we have shown the experimental setup by going through the various parts of the algorithm. Then, we have reported the results achieved by our approach in different scenarios after optimizing the used parameters.

Conclusion and Future Perspectives

The goal of this thesis is to develop a computer vision algorithm for mobile robots. This system will enable the robot to avoid obstacles on its way. In this project, we concentrated on developing the method, analyzing its feasibility, and choosing a suitable sensor.

As we mentioned, we used the Microsoft Kinect v2.0 sensor. It has a low price, it can work indoor and outdoor, and it is more performant compared to many other sensors. For the implementation, we used python along with the library PyntCloud which enables us to deal with 3D clouds.

The system presented in this thesis takes as input depth and RGB images and gives as output the points in the images that represent the obstacles, then it provides a map of the scene which can be used for navigation.

This internship was an opportunity to learn new values like time management, team work, and improve my social skills and communication abilities. This internship was also an opportunity to deepen my technical skills, in **Python** and **3D computer vision**.

This work gave good results, but always there are various ways to improve things and this depends on our needs. This project can be extended by tackling the following axes:

- Building a ground truth and use it to assess the accuracy of the algorithm we implemented.
- Testing the proposed approach in various scenarios "different environment: night/-sun, farm/house".
- Carrying out the generalization of the algorithm to handle videos.
- Exploiting more the color information in the RGB data.

Bibliography

[1] Equation of a line in a 3d space. URL http://www.nabla.hr/PC-LinePlaneIn3DSp1.htm.

[2] Equation of a plane in a 3d space. URL http://www.nabla.hr/CG-LinesPlanesIn3DA3.htm.

[3] 3 nearest neighbors. URL https://kraj3.com.np/blog/2019/06/basic-concepts-of-knn-algorithm/.

[4] Examples of correlation values. URL https://en.wikipedia.org/wiki/Correlation_and_dependence.

[5] The normal vector of a surface. URL https://www.algosome.com/articles/pca-three-dimensions-point-cloud.html.

[6] Perspective projection. URL https://www.slideshare.net/zukun/lecture12-8077305.

[7] Normal vectors of a given huge surface. URL https://www.mathworks.com/matlabcentral/fileexchange/23063-compute-normal-vectors-of-2-5d-triangulation.

[8] Normal vectors of a given obstacle surface. URL https://ximera.osu.edu/mooculus/calculus3/shapeOfThingsToCome/digInSurfaceIntegrals.

[9] Normal vectors of a flat surface. URL https://opentextbc.ca/universityphysicsv2openstax/chapter/electric-flux/.

[10] Kinect sensor. URL https://kinect-i.blogspot.com/2012/05/how-to-install-and-use-openni-microsoft.html.

[11] Kinect sensor vrsion1. URL https://hidale.com/shop/dp-kinect/.

[12] Sl principle. URL https://arxiv.org/pdf/1505.05459.pdf.

[13] Kinect sensor version2. URL https://www.technologyrecord.com/Article/microsoft-stops-producing-kinect-for-windows-v2-sensors-45395.

[14] Tof principle. URL https://www.terabee.com/time-of-flight-principle/?fbclid=IwAR1365R2AJH1SWsSxSRyzCGadiyMsUdGUh3f8nhr5qltyMquetuIdpWi7mA.

[15] Microsoft kinect v2 showing the orientation of the coordinate system. URL https://homes.cs.washington.edu/~edzhang/tutorials/kinect2/kinect3.html.

[16] Depth image and 3d point cloud of the same scene. URL https://www.researchgate.net/figure/Correspondence-between-depth-map-and-3D-point-cloudfig1267990793.

[17] Sample data from the diml cvlab rgb-d datase. URL https://dimlrgbd.github.io/downloads/technicalreport.pdf.

[18] Sample data from the open benchmark corpus for mobile rgb-d re-lated algorithms datase. URL (http://mobilergbd.inrialpes.fr//RobotView.

[19] Zhengyou Zhang. Microsoft kinect sensor and its effect. *IEEE multimedia*, 19(2): 4–10, 2012.

[20] Raphael Labayrade, Didier Aubert, and J-P Tarel. Real time obstacle detection in stereovision on non flat road geometry through" v-disparity" representation. In *Intelligent Vehicle Symposium, 2002. IEEE*, volume 2, pages 646–651. IEEE, 2002.

[21] Tim G McGee, Raja Sengupta, and Karl Hedrick. Obstacle detection for small autonomous aircraft using sky segmentation. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 4679–4684. IEEE, 2005.

[22] Baozhi Jia, Weiguo Feng, and Ming Zhu. Obstacle detection in single images with deep neural networks. *Signal, Image and Video Processing*, 10(6):1033–1040, 2016.

[23] Noa Garnett, Shai Silberstein, Shaul Oron, Ethan Fetaya, Uri Verner, Ariel Ayash, Vlad Goldner, Rafi Cohen, Kobi Horn, and Dan Levi. Real-time category-based and general obstacle detection for autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 198–205, 2017.

[24] Javier Hernandez-Aceituno, Rafael Arnay, Jonay Toledo, and Leopoldo Acosta. Using kinect on an autonomous vehicle for outdoors obstacle detection. *IEEE Sensors Journal*, 16(10):3603–3610, 2016.

[25] Hamed Sarbolandi, Damien Lefloch, and Andreas Kolb. Kinect range sensing: Structured-light versus time-of-flight kinect. *Computer vision and image understanding*, 139:1–20, 2015.

[26] 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.

[27] Srikanth Varanasi and Vinay Kanth Devu. 3d object reconstruction using xbox kinect v2. 0, 2016.

[28] Niclas Zeller. Obstacle detection using microsoft kinect.