Ivo André Serafim Ferreira

Depth Camera Based Image Processing for Indoor Mobile Robot Localization and Navigation

Thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in Electrical and Computer Engineering

September, 2016

UNIVERSIDADE DE COIMBRA

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Depth Camera Based Image Processing for Indoor Mobile Robot Localization and Navigation

Ivo André Serafim Ferreira

Coimbra, September of 2016

Depth Camera Based Image Processing for Indoor Mobile Robot Localization and Navigation

Supervisor:

Prof. Dr. Urbano José Carreira Nunes

Co-Supervisor:

Prof. Dr. Ana Cristina Barata Pires Lopes

Jury:

Prof. Dr. Jorge Manuel Moreira de Campos Pereira Batista

Dr. Micael Santos Couceiro

Prof. Dr. Urbano José Carreira Nunes

Dissertation submitted in partial fulfillment for the degree of Master of Science in Electrical and Computer Engineering.

Coimbra, September of 2016

Acknowledgements

Firstly I would like to express gratitude to my advisors Professor Urbano Nunes and Dr. Ana Lopes for their essential guidance, experience and encouragement. This dissertation would be impossible without their dedication.

I also would like to express a special thanks to my colleagues André Conceição and Urbano Nunes for their help whenever I needed and to Luís Garrote for his support and wise advices.

I thank to Institute of Systems and Robotics for providing me the best conditions and resources that allowed me to accomplish this final stage of my master degree. This work has been supported by project Grant "AMS-HMI12: Assisted Mobility Supported by shared control and advanced Human Machine Interfaces" (FCT Project RECI \EEI-AUT\0181\2012), co-funded by Fundação para a Ciência e Tecnologia and FEDER through "Programa Operacional Factores de Competitividade do QREN com referência COMPETE: FCOMP-01-0124-FEDER-027501".

To all my friends that somehow shared these past 5 years with me I express my gratitude, with you I lived some good moments that I will definitely never forget.

Por fim, dirijo o meu maior agradecimento à minha familia, principalmente aos meus pais, por todo o apoio que me deram, possibilitando a frequência e conclusão deste curso.

Resumo

O aumento do desenvolvimento de plataformas robóticas móveis capazes de desempenhar actividades de navegação, localização e mapeamento em ambientes partilhados com humanos tem vindo a propulsionar áreas de investigação tais como a Robótica Móvel, Robótica Cooperativa, Interação Homem-Máquina, etc. Esta questão ganha maior relevância quando se reúnem as condições de criar uma plataforma robótica móvel própria, de raiz, tornando-a num sistema robusto que serve de matéria de investigação a nível interno e porventura, mais tarde, vir a ser colocado no mercado. O Interbot, robô social, é uma plataforma robótica móvel desenvolvida pelo ISR para actuação em ambientes interiores partilhados com humanos.

Nesta dissertação propõe-se um módulo de aquisição e processamento de imagem, mais especificamente de imagens de profundidade, com o objectivo de fornecer ferramentas úteis de localização e navegação para esta plataforma.

Em termos de hardware, é usado um sensor Microsoft Kinect One devidamente adaptado à plataforma. Posto isto, desenvolveu-se um sistema de processamento de imagens de profundidade de baixa complexidade. O sistema recebe a informação do ambiente envolvente através do sensor Microsoft Kinect One, estes dados são submetidos a uma primeira fase de processamento, onde são extraídas as estruturas planares da imagem (planos da imagem) e respectivos parâmetros (equação do plano, normal do plano, pontos do plano), e uma segunda fase de processamento onde é gerado um *laser scan* com base nos dados previamente processados. O utilizador tem a possibilidade de facilmente regular os parâmetros de configuração do sistema consoante as características do ambiente e a finalidade desejada. Todos os algoritmos propostos foram desenvolvidos e testados em MatLab e ROS (Robotic Operating System).

Palavras-chave: Robô, Robótica Cooperativa, Robótica Social, Navegação, Localização, ISR, ROS, MatLab, Plataforma Robótica Móvel, Microsoft Kinect One, Detecção de Planos,

Extração de Planos, Imagem de Profundidade, Processamento de Imagem.

Abstract

The increasing development of mobile robotic platforms able to perform activities such as localization, navigation or even map building in human shared environments has been propelling research areas such as Mobile Robotics, Cooperative Robotics, Human-Machine Interaction, etc. This issue is especially relevant when there are conditions to create an own mobile robotic platform from the scratch, making a robust system that serves internally for research purposes and perhaps to be placed on the market later on. The social robot, Interbot, is a mobile robotic platform developed by the ISR for indoor environments deployment.

In this dissertation an image processing and acquisition module, more specifically depth images is proposed, with the aim of providing useful localization and navigation tools for this platform.

In terms of hardware, it is used a Microsoft Kinect One sensor properly attached to the Interbot platform. Hereupon, a low complexity depth image processing system was developed. The system acquires the surrounding environment information using the Microsoft Kinect One sensor, this data is subjected to a first stage of processing, where the planar structures of the image (image planes) and their information (plane's equation, normal to plan and points within the plane) are extracted, then a second processing phase where an obstacle laser scan is generated based on the preprocessed data. The user has the ability to easily adjust the system configuration parameters depending on the environmental characteristics where he wants to use the system and depending on the desired purpose. All proposed algorithms are developed and tested in MatLab and ROS (Robot Operating System).

Keyword: Robot, Cooperative Robotic, Social Robotics, Navigation, Localization, ISR, ROS, MatLab, Mobile Robotic Platform, Microsoft Kinect One, Plane Detection, Plane Extraction, Depth Image, Depth Image Processing.

"Don't watch the clock; do what it does. Keep going!"

— Sam Levenson

Contents

List of Acronyms

1D	One Dimensional
2D	Two Dimensional
3D	Three Dimensional
BMS	Battery Management System
DAcM	Data Acquisition Module
DAnM	Data Analysis Module
DPM	Data Processing Module
FCT	Fundação para a Ciência e Tecnologia
FSPF	Fast Sampling Plane Filtering

FSPF+	Fast Sampling Plane Filtering Plus
KOS	Kinect Obstacle Scan
I/O	Input and Output
ISR	Institute of Systems and Robotics
NCC-RANSAC	Normal Coherence Check Random Sample Consensus
OES	Offline Experimental Setup
ODAnM	Offline Data Analysis Module
PNG	Portable Network Graphics
RANSAC	RANdom SAmple Consensus
RtES	Real Time Experimental Setup
RGB	Red, Green and Blue
ROS	Robot Operating System

SLAM	Simultaneous Localization and Mapping
SRM	Statistical Region Merging
USB	Universal Serial Bus

List of Figures

List of Tables

1 Introduction

This chapter presents an introduction to this dissertation. Some insights concerning the motivation and context of the developed work will be presented, as well the main goals and key contributions. Moreover an global overview of the structure of this work is also presented summarizing the content of each chapter.

1.1 Motivation and context

Today it is possible to find in research institutes, universities or even in the market mobile robotic platforms able to perform guiding tasks, monitoring, patrolling buildings, cargo transportation, etc. For these mobile platforms to operate efficiently, robust localization and navigation systems are required, aiming to minimize localization loss issues, to allow tracing motion paths quickly and safely from one point to another, and also to allow the surroundings map generation as close as the original one. Mapping issue is one of the basics of mobile robotics.

Map building is a process that can combine the use of the robot motion (odometry), range finder devices, that provide range data of the surrounding environment (laser scan) and also a scan matcher method. There are a wide range of mapping methods for such platforms, those most commonly used in ROS implementation are the *g_mapping* [21] and the *hector_mapping* [29].

A state of the art platform in this area is the CoBot Robot [47], [46], developed at Carnegie Mellon University (CMU). The robot is equipped with state of the art localization, navigation and mapping modules that allowed it to navigate autonomously, while performing simple tasks, for over 1000km [7] during a time span of 4 years.

The ISR has internally developed a mobile robotic platform, called Interbot - social robot [11], the social robot, that has been used for research purposes in various strands of mobile robotics. In this context, the development of an image processing visual module

based on Microsoft Kinect One sensor data, aiming to provide visual and geometrical tools for enhancing localization, navigation and map building by the Interbot platform, was proposed.

This dissertation proposes a system that firstly processes in real-time the depth images of the environment where the platform operates by extracting information from the planes of these images and secondly generates a 2D obstacle scan of the platform's surroundings. This scan is then used to generate, also in real-time, the map of the environment.

1.2 Goals

This dissertation aims to develop useful indoor mobile robot localization and navigation tools based on the processing of depth images to be implemented on mobile robotic platforms such as the Interbot Platform. The defined goals were:

1. To develop an algorithm that processes depth images and extracts the planar structure information (planes and normals to the planes) from these images;

2. To develop an algorithm that generates a multi-level laser scan based on the planes that were extracted by the previously referred algorithm;

3. To develop a planes' classification method;

4. To store all the generated data for future applications;

5. To conduct tests in order to evaluate the proposed algorithms.

1.3 Implementations and key contributions

The improvement of a depth image processing method [5], the enhancement of a 3D point image processing method [4] and their implementation on a ROS framework are the key contributions of this dissertation.

In order to develop a fully operational system, some ROS modules were used and other were developed. Figure 1.1 illustrates the main developed modules described in this dissertation. The course flow of this dissertation and content of each chapter is the following:

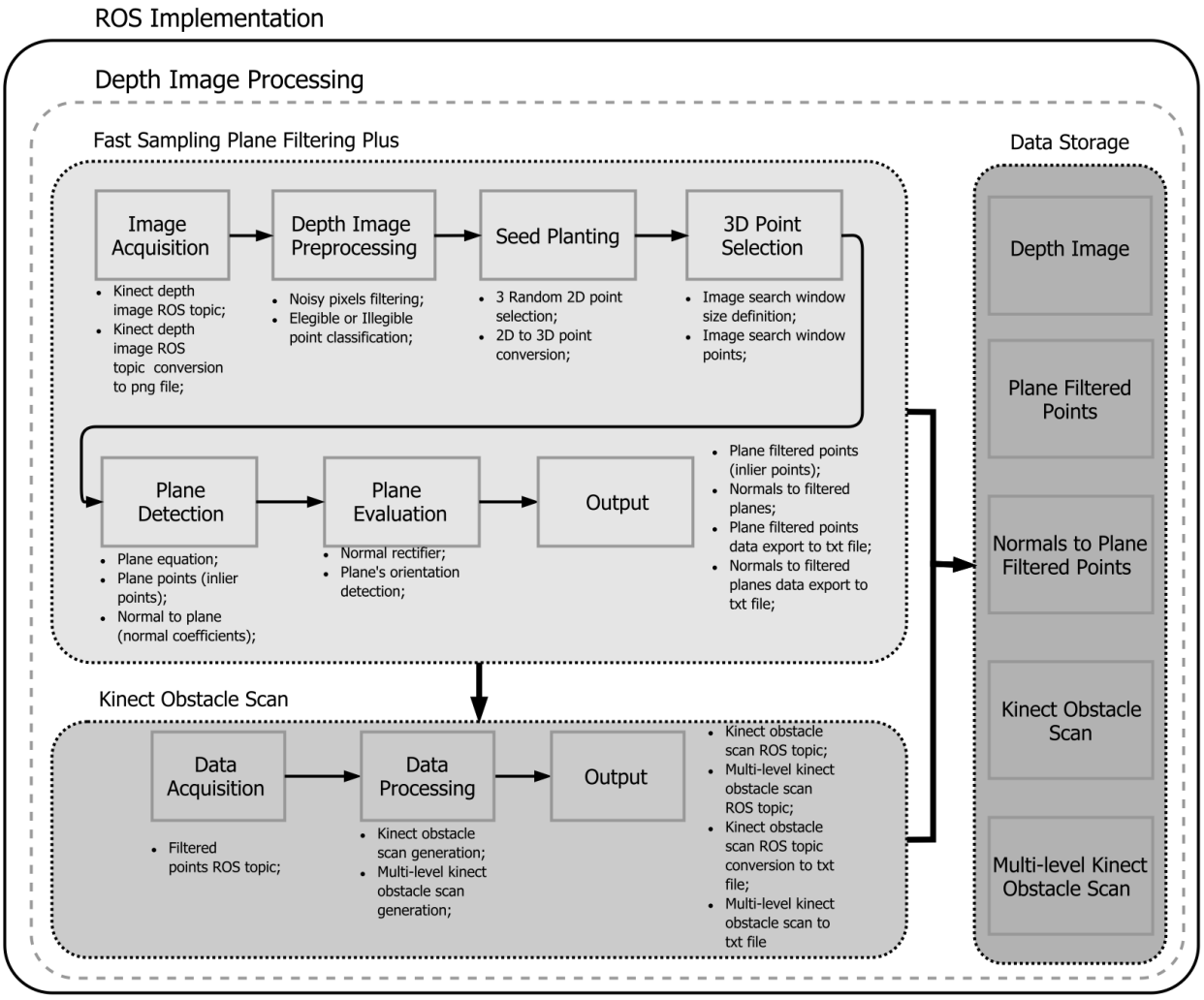


Figure 1.1: Image Depth Processing main modules and key contributions.

Plane Extraction (Chapter 3)

- Fast Sampling Plane Filtering Plus (FSPF+): development of an improved version of the FSPF algorithm [5].

- Depth image preprocessing algorithm: development of an algorithm based on depth image pixel's value constraints and its integration in the FSPF+ algorithm.

- 3D selection points: development of an algorithm for searching 3D points within a provided image window and integration of this algorithm in the FSPF+ algorithm.

- Plane detection: integration of a RANSAC algorithm in the FSPF+ algorithm for plane detection, selecting inlier and outlier points also providing plane's information such as plane equation, normal to detected plane and points' plane.

- Plane normal rectifier: development of a plane rectifier algorithm base on the coef-

ficients of the plane and its position within the 3D range image. Integration of this algorithm in the FSPF+ algorithm.

- Plane orientation detection: development of an algorithm for plane orientation detection based on plane normal's direction and its localization within the 3D range image. This algorithm is capable of a five different plane orientation detection.

Obstacle Scan Generation (Chapter 3)

- Kinect Obstacle Scan (KOS): development of an improved version of the Obstacle scan algorithm [4]. This improvement is based on a functionality that allows a multi-level 2D obstacle scan.

Data Storage (Chapter 3 and 4)

- Image plane structure data storage: filtered 3D image plane points, normals to 3D image filtered plane points and kinect obstacle scan points are storage in text files for future work applications.

- Depth image storage: the processed depth image is storage in portable network graphic (PNG) format for later analysis purposes.

- Hokuyo's laser scan points storage: the laser scan points are storage in a text file for later test results comparison.

ROS Implementation (Chapter 4)

- kinect2_bridge package: integration of a ROS package *kinect2_bridge* for color, depth and point cloud images acquisition.

- kinect_node package: development of a ROS package, *kinect_node* that implements the FSPF+ and KOS developed algorithms.

- plane_filtering: a ROS node that implements the FSPF+ algorithm.

- kinect_obstacle_scan: a ROS node that implements the KOS algorithm.

- hector_mapping: integration of the ROS package *hector_mapping* for map building purposes.

- RVIZ: integration of the ROS visualization tool *Rviz* for offline and real-time results analysis.

In Chapter 5, several experiments for testing and validation of the proposed modules were performed and their results are presented and discussed.

2 State of the art and background

This chapter introduces the fundamental topics for depth based plane detection with application in indoor map building, navigation and localization. It reviews the state of the art on 3D image segmentation and also important background theory on image representation, RANSAC for plane detection and a plane filtering algorithm that supports the developed work in this dissertation.

2.1 3D Image segmentation

3D perception is one of the crucial fields of research towards fully autonomous and robust service robot operation. Without perception of the surrounding environment, service robots will not be able to navigate, manipulate or interact with humans successfully.

This way some computer vision techniques such as image segmentation, more specifically plane detection, is used to acquire information that will generate data allowing robots to have perception of their surroundings.

Segmentation is the separation of one or more regions or objects in an image based on a discontinuity or a similarity criterion [19]. A region in an image can be defined by its border (edge) or its interior; if the image's interior is known, it is always possible to define the border, and vice versa.

Plane detection is one of the most popular approaches in retrieval of environment information once planar structures are abundant in human-made environments. Moreover it is useful to model the environment with basic geometry, such as planes, spheres, curved surfaces, etc. This provides important tools to understand and to manage the environment from a robot's point of view. Plane detection often yields a robust method, which is crucial in several practical works. Planes could be extracted from data provided by different sensors including video cameras [16], laser range finders [51] and 3D time-of-flight cameras [37].

Plane extraction methods can be classified according to the literature [35], [38] as: edge

based, region based and hybrid methods. Relevant works on plane extraction are categorized in Table 2.1. Figure 2.1 presents the structure of a typical plane extraction method.

Table 2.1: Plane extraction methods for range images.

Method	**Description**
Edge based [14], [27]	Detects the edges of a range image and uses them to bound the planar regions. It assumes that edges are borders between different planar surfaces that have to be segmented separately.
Region based [1], [28], [31], [37], [50]	Selects some seed points in range data and grows them into regions based on the homogeneity of local features (surface normal, point-to-distance, etc).
Hybrid [3], [20], [35]	Combines both edge based and region growing techniques.

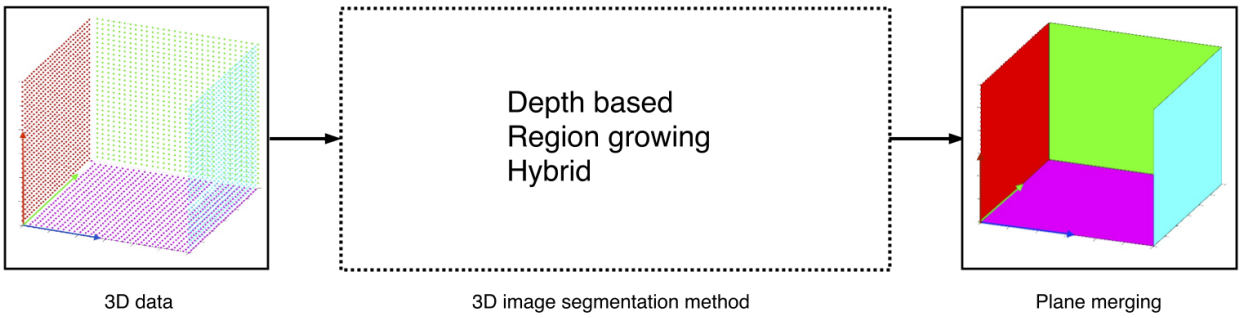


Figure 2.1: Flowchart of the typical structure of a plane extraction algorithm.

2.1.1 Edge based methods

In these kind of extraction methods two steps are required: 1) *Edge detection*, to detect edge pixels using image operators such as the Gradient [9], the Laplacian [24], the Laplacian of the Gaussian [23] and Canny filtering [10]; 2) *Edge linking*, to link adjacent edge pixels into edges. Edges in a predefined neighborhood are linked if both magnitude and direction of the gradient criteria is satisfied.

The main challenge occurring with edge based methods is to generate closed contours from the edges. As referred before, the boundary and edge points are first identified and then the extracted border points are used to segment the surface according to different features (e.g. normal and curvature). Every region within a closed contour forms a segment. These methods work well in images with good contrast between object and background and they represent a similar approach to how humans segment images. In general, these methods cannot guarantee closure of boundaries for surface extraction, resulting in the need of further information. Moreover most of the edge based approaches are computationally expensive and therefore not suitable for real time applications [2], another problem is noisy data. Once

it is crucial to identify closed contours for these methods, sensor noise reduces robustness significantly.

As a result region based approaches have played a dominant role in the vast majority of image analysis systems.

2.1.2 Region based methods

The principle of region based or region-growing methods is to start with a seed region and to grow it by neighborhood when the neighbors satisfy some conditions. They use local neighborhood properties to seek the homogeneity within a specific feature or find variation among the features, and merge spatially close points.

In other words, region based methods could start from one of this set of seed options (starting pixels) [19]: 1) Predefined seeds; 2) All pixels as seeds; or 3) Randomly chosen seeds. Then a two steps procedure is initialized: 1) Find starting points; 2) Include neighboring pixels with similar features (e.g. gray level, texture, color). Here a similarity measure must be selected (e.g. variance or standard deviation within a region, within a value range, intensity difference within a region). There are two different variants of these procedures [19]: 1) Select seeds from the whole range of gray levels in the image and then grow regions until all pixels in image belong to a region; 2) Select seed only from objects of interest (e.g. bright structures) and then grow regions only as long as the similarity criterion is fulfilled. In range images, the neighbors of each point with pixel coordinates are available. In case of unorganized 3D data, there is no information about the neighborhood in the data structure. The most common method to compute neighbors in 3D is to compute a Kd-tree [26] to search k nearest neighbors. The creation of a Kd-tree lies in $O(NlogN)$ and the search of k nearest neighbors of one point lies on $O(logN)$. Region-growing methods can be categorized on the following groups: Seeded region-growing [1], Unseeded region-growing [31] and model based [8], [15] and [38].

1. Seeded region growing method [1] — this method takes a set of seeds as input along with the image. The seeds mark each of the objects to be segmented. The regions are iteratively grown by comparison of all unallocated neighboring pixels to the regions. The difference between a pixel's intensity value and the region's mean δ is used as a measure of similarity. The pixel with the smallest difference measured in this way is assigned to the respective region. This process continues until all pixels are assigned to a region. Because seeded region growing requires seeds as additional input, the

segmentation results are dependent on the choice of seeds, and noise in the image can cause the seeds to be poorly placed.

2. Unseeded region growing method [31] — this method is characterized by not require explicit seeds. It starts with a single region A_1 — the pixel chosen here does not markedly influence the final segmentation. At each iteration it considers the neighboring pixels in the same way as seeded region growing. It differs from seeded region growing in that if the minimum δ is less than a predefined threshold T then the pixel is added to the respective region A_j, otherwise, the pixel is considered different from all current regions A_i and a new region A_{n+1} is created with this pixel. One variant of this technique, proposed in [45] is based on pixel intensities. The mean and scatter of the region and the intensity of the candidate pixel are used to decide whether the pixel is added to the region, and the region's mean and scatter are recomputed or the pixel is rejected, and is used to form a new region.

3. Model based methods — a model based methods work well for a scene where objects can be described by mathematical models such as plane, sphere, curved surface. RANSAC [15], NCC-RANSAC [38] and Hough Transform based [8] may be viewed as a region growing approaches since they clusters data points with similar property into a plane [38]. The advantage of these region growing methods is that they are fast when there are many planes to extract, robust to noise and extract the largest connected component immediately. But some of them only use the distance from point to plane to extract planes, which decreases the accuracy of correct planar region detection [13]. Some problems with RANSAC based approaches [6], [15], [38] are that they are not model consistent (the best model fit is done even if the data represents something else), the model to be searched has to be known beforehand and it may fails when a scene contains multiple intersecting planar surfaces [18].

Region based methods can result in over or under segmentation and region border can be hard to locate. They are also sensitive to the choice of initial seed points. However model based methods seem to be applicable in more general manner. Figure 2.2 presents a general structure of the plane extraction methods outlined in Table 2.2.

Table 2.2: Region-growing approachs.

Approach \ Paper	**Description**
Fast Planar Clustering and Polygon Extraction from Noisy Range Images Acquired in Indoor Environments [28]	This approach clusters range image data into coplanar points and extracts polygons from range images. The neighboring 3D points in the range image are grouped into small patches of size k and the Hessian plane parameters of each patch are computed. The algorithm searches for coplanar patches by comparing the normal of neighboring patches and their mean distance. The search for coplanar patches will end when all patches are assigned to a cluster. At the end, the boundary points are extracted for each cluster.
Fast Plane Detection and Polygonalization in noisy 3D Range Images [37]	A very fast approach to surface extraction for 3D maps. As stated in [37]: *"It consist in two stages: 1) A new plane fitting algorithm is used where the standard mathematical formulation of the problem is revised to an incremental version, allowing an efficient region growing. Second, the segmented regions are polygonalized exploiting the fact that mobile robots capture the 3D data as a sequence of 3D range images."*.
Learning compact 3D models of indoor and outdoor environments with a mobile robot [22]	In this work a region growing algorithm to extract polygons from range images is used. The range image coordinates are converted into polygons in two steps. First, an expensive nearest neighbor search is performed to cluster the points belonging to individual planes. Then the boundary points are extracted to form irregular polygons. Second, the neighboring smaller polygons are merged into larger polygons by looking for common edges. The algorithm is time-consuming with a time complexity of $O(n^2)$, where n is the number of coordinate points in the range image.
RANSAC [15]	This is a model based region growing based algorithm that detects planes by randomly sampling some points to construct a plane and then use the remaining points to test whether the constructed plane is a good estimate.
NCC-RANSAC [38]	Checks the normal coherences for all data points of inliers patches (on the fitted plane) and removes the data points whose normal directions are contradictory to that of the fitted plane. This process results in a number of separate inlier patches, each of which is treated as a seed plane. A recursive plane clustering process is then executed to grow each of the seed planes until all planes are fully extracted.

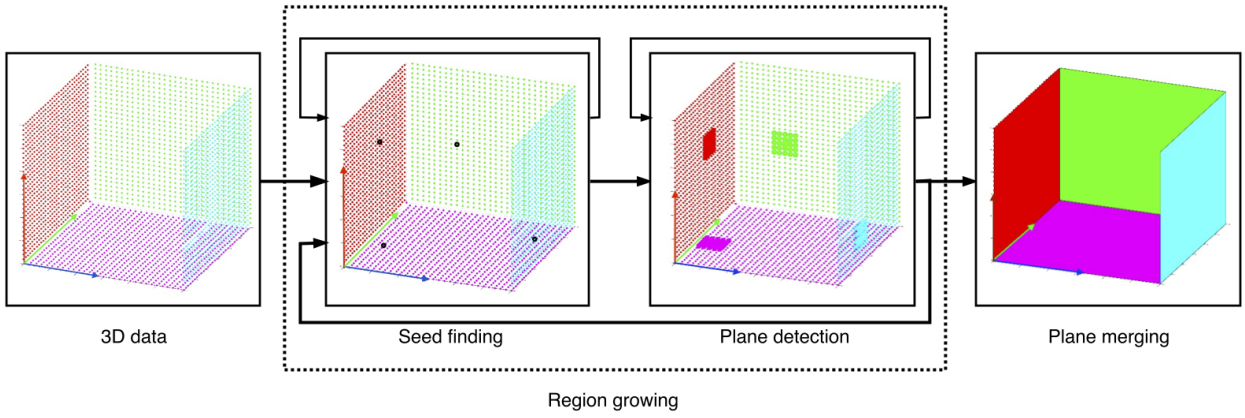


Figure 2.2: Flowchart of the general structure of a region-growing plane extraction algorithm.

2.1.3 Hybrid methods

Hybrid methods aim to overcome the limitation of both edge-based and region-based methods combining techniques from both these methods [32]. In [35] an hybrid robust algorithm is proposed, this is able to localize region borders and edge points accurately, it efficiently handles the regiown-growing over and under segmentation problem and reduces outliers effects for the whole process. An improved robust estimator, which iteratively detect and extract planar surfaces, is presented in [20]. The major problems of hybrid techniques are the accuracy of the segmentation, efficiency in terms of speed of region growing around the pixels [52] and the high computational requirements when merging the over-segmented regions [48].

2.2 Image representation

This section addresses to basic image concepts as well as 2D\3D image mathematical notation used in this dissertation.

A digital image is represented by a 2D rectangular array of discrete values. Both image space and intensity ranges are quantified into a discrete set of values. We begin with an ideal notion of an analog image created by an ideal optical system which we assume to have infinite precision. Digital images are formed by sampling this analog image at discrete locations and representing the intensity at a location as a discrete value [45]. All real images are affected by physical processes that limit precision in both position and intensity. According to [45]: 1) An analog image is a 2D image $F[x, y]$ which has infinite precision in spatial parameters x and y and infinite precision in the intensity at each spatial point (x, y); 2) A digital image is a 2D image $I[r, c]$ represented by a discrete 2D array of intensity samples, each of which is represented using a limited precision; 3) A picture function is a mathematical representation $f[x, y]$ of a picture as a function of two spatial variables x and y. These variables are real values defining points of the picture and $f[x, y]$ is usually also a real value defining the intensity of the picture at point (x, y); 4) A multispectral image is a 2D image $I[r, c]$ which has a vector of values at each spatial point or pixel. If the image is actually a color image, then the vector has 3 elements, the red (R), green (G) and blue (B) values.

A coordinate system must be used to address individual pixels of an image; to operate on it in a computer program, to refer it in a mathematical formula, or to address it relative

to device coordinates. The most common used 2D coordinate systems are shown in Figure 2.3. In this dissertation a raster oriented coordinate system (Figure 2.3a) is adopted where each pixel at row position i and column position j is represented by $I(i, j)$. Hence, a color digital image I is represented as shown in Figure 2.4. On the other hand, a depth digital image I is represented as shown in Figure 2.5. This depth image was extracted from the color image represented in 2.4. Despite the depth image just having two colors (white and black), does not mean that its pixels' depth only assume two different values.

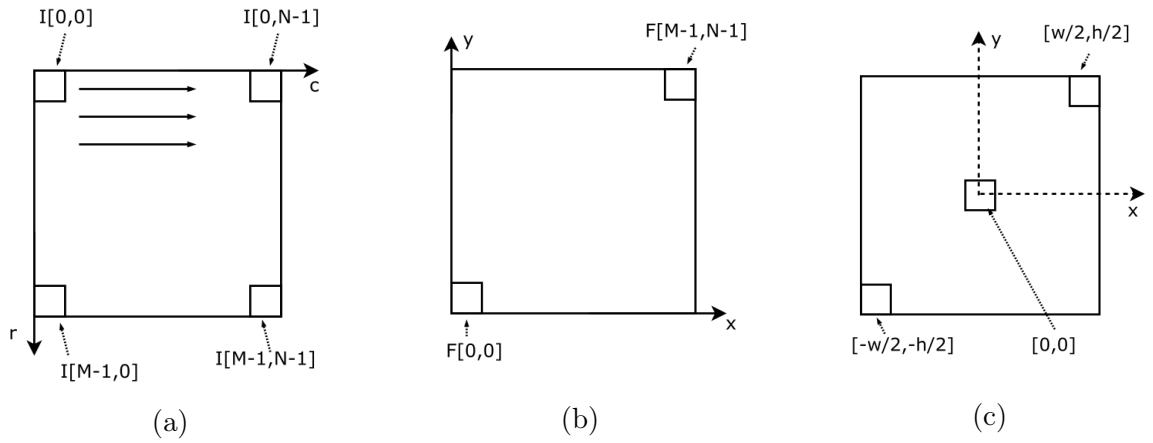


Figure 2.3: 2D coordinate systems used in images: (a) raster oriented uses row and columns coordinates starting at $[0, 0]$ from the top left. (M,N) are the corresponding width and height of the image in pixels; (b) Cartesian coordinate frame with $[0, 0]$ at the lower left. (M,N) are the corresponding width and height of the image in pixels; (c) Cartesian coordinate frame with $[0, 0]$ at the image center, where (w,h) are the corresponding width and height of the image in pixels.

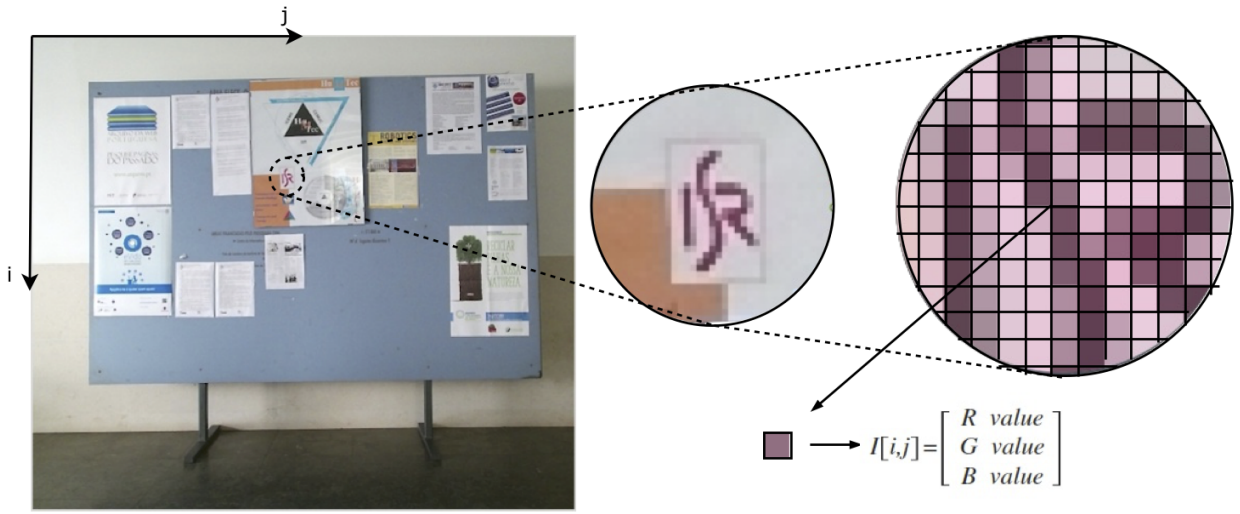


Figure 2.4: Color image representation.

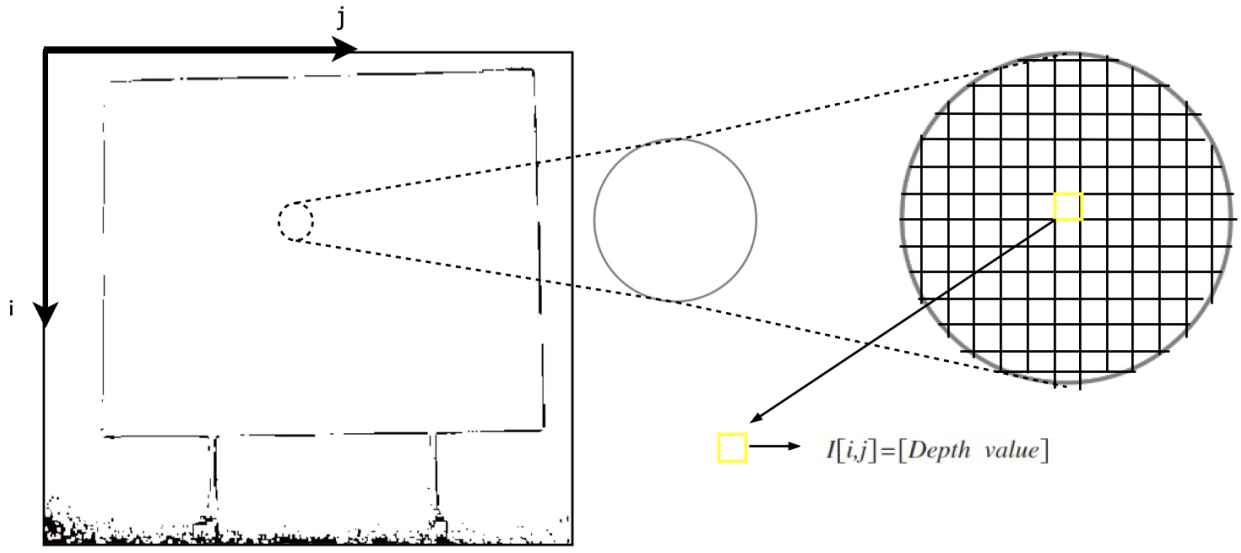


Figure 2.5: Depth image representation.

Starting from the coordinate system of a depth image and its points it is possible to convert them obtaining the 3D representation of this image regarding a coordinate system of the camera's frame. Thus a depth image pixel p at position (i, j) will get its 3D coordinates $[s^x, s^y, s^z]$ according to the following set of equations [4]:

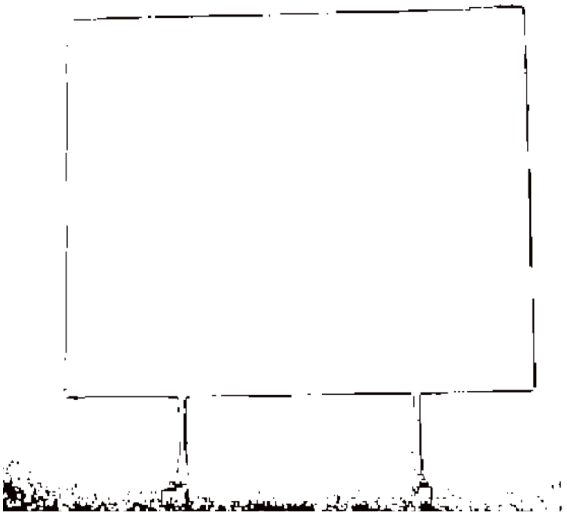
$$s^x = p(i,j)\left(\frac{j}{w-1} - 0.5\right)\tan\left(\frac{f_H}{2}\right) \tag{2.1}$$

$$s^y = p(i,j)\left(\frac{i}{h-1} - 0.5\right)\tan\left(\frac{f_V}{2}\right) \tag{2.2}$$

$$s^z = p(i,j) \tag{2.3}$$

where the size of the depth image in pixels is $h \times w$, $p(i, j)$ is the depth value of the depth image pixel in position (i, j), f_H and f_V are the horizontal and vertical camera's fields of view, respectively. In Figure 2.6 the conversion of a depth image scenario to its 3D correspondence is shown.

(a)

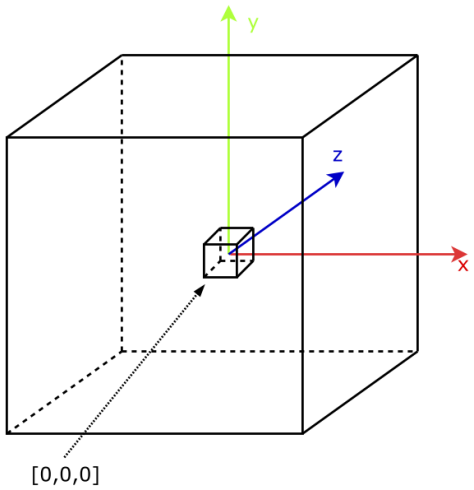


(b)

(c)

(d)

(e)



(f)

Figure 2.6: Different types of images and points of view: (a) Color image. (b) Depth image. (c) 3D image of the depth image points (b). (d) Top view of (c). (e) Top left view of (c). (f) Coordinate system of image (c).

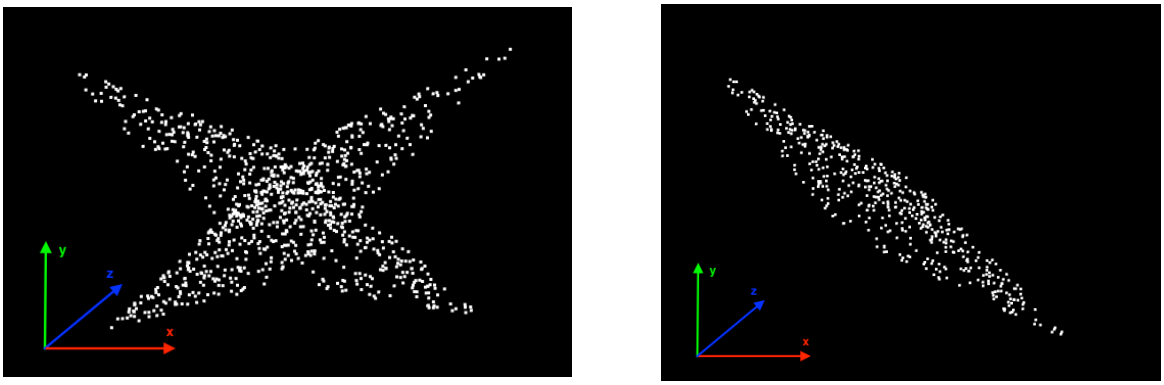
2.3 RANSAC for plane detection

The RANdom SAmple Consensus was proposed published by Fischler and Bolles at SRI International in 1981 [15]. The RANSAC algorithm is a resampling technique [12] to estimate parameters of a model by random sampling observed data. Given a dataset of 3D points *inputData* whose elements contain both inlier and outlier points, a distance threshold ϵ and a minimum number of samples β required to fit a mathematical model, RANSAC uses a voting procedure to find the optimal fitting result providing the model parameters such as plane equation coefficients, normal to plane coefficients N, and the same number (β) of points P that define the sampled plane. Data elements in the dataset are used as votes for one or multiple models. The implementation of this voting procedure is based on two assumptions: 1) The noisy features will not vote consistently for any single model; 2) There are enough features to agree on a good model. The RANSAC algorithm is essentially composed by two steps that are iteratively repeated:

1. In the first step, a sample subset containing β data items P is randomly selected from the input dataset *inputData*. A fitting model and the corresponding model parameters are computed using only the elements of this sample subset. The cardinality of the sample subset will always be equal or greater than 3 once RANSAC is searching for a mathematical model described as a plane.

2. In the second step, the algorithm checks which elements of the entire dataset *inputData* are consistent with the model instantiated by the estimated model parameters obtained from the first step. A data element will be considered as an outlier, if it does not fit the fitting model instantiated by the set of estimated model parameters within a maximum distance threshold ϵ.

The set of inliers obtained for the fitting model is called consensus set $\hat{P}$. The RANSAC algorithm will iteratively repeat the above two steps until the obtained consensus set in certain iteration has the maximum number of inliers. RANSAC achieves its goal by repeating the following steps: 1) Select a random subset of the original dataset *inputData*. Call this subset the hypothetical inliers; 2) A model is fitted to the set of hypothetical inliers; 3) All other data are then tested against the fitted model. Those points that fit well the estimated model, according to some model-specific function, are considered as part of the consensus set; 4) The estimated model is reasonably good if enough points (from 90% up to 99% of



(a) (b)

Figure 2.7: An overview of the RANSAC for plane detection application: (a) 3D points dataset *inputData*. (b) Inlier points $\hat{P}$ of the best fitting model.

the points) have been classified as part of the consensus set; 5) Afterwards, the model may be improved by reestimating it using all members of the consensus set. This procedure is repeated a fixed number of times *maxIterations* [1], each time producing either a model which is rejected because too few points are part of the consensus set, or a refined model together with a corresponding consensus set size. In the latter case, we keep the refined model if its consensus set is larger than the previously saved model.

The application of the RANSAC for plane detection is shown in Figure 2.7, Figure 2.7a shows the 3D points dataset *inputData* that is provided to the RANSAC algorithm and Figure 2.7b the inlier points $\hat{P}$ of the best fitting model are shown.

2.4 Fast Sampling Plane Filtering

The Fast Sampling Plane Filtering Algorithm (FSPF) was developed by Biswas [4]. This algorithm is used to filter depth images providing the plane filtered points, plane to point normals and a set with all outlier depth image points. The procedure of this method is outlined in **Algorithm** 1 and its configuration parameters are presented in Table 2.3.
It starts by taking a depth image I of size $h \times w$ as its input parameter (line 1) then initializes three lists, one that will contain all the image plane filtered points F_P, one with its normals to the filtered planes N_P, and a third one with all the image outlier points O_P (lines 2-4). While the maximum number of sampled neighborhoods $n_{neighbor}$ or the maximum number of image filtered points n_{filt} are greater than the number of sampled neighborhoods k or greater than the number of filtered points n, respectively, the algorithm will perform an iterative procedure (lines 7-25). The number of sampled neighborhoods is incremented (line

[1] $\approx$ 1000 iterations on a MatLab application [30] and $\approx$ 10000 iterations on a C++ application [36].

Table 2.3: Configuration parameters for **Algorithm** 1

Parameter	**Description**
n_{filt}	Maximum total number of filtered points
$n_{neighbor}$	Maximum number of neighborhoods to sample
l	Number of local samples
Ω	Neighborhood for global samples (in pixels)
Δ	Plane size in world space for local samples
ϵ	Maximum distance threshold for planes
α_{in}	Minimum inlier fraction to accept local sample

8). Then an initial random 2D point *p0* is generated within the limits of the image and two more points *p1* and *p2* are selected within a neighborhood of size Ω around the position of *p0* (lines 9-11). The 3D points [s_0, s_1, s_2] for each 2D point [*p0*, *p1*, *p2*] are then computed (line 12) according to the set of equations (2.1) , (2.2) and (2.3) presented in Section 2.2. The normal of the plane is computed based on the three 3D points (line 13), followed by the mean depth $\bar{z}$ of the same 3D points is computed as well (line 14). The size of a search window $H \times W$ is defined based on the following parameters: the mean depth $\bar{z}$, the size of I, both horizontal f_H and vertical f_V fields of view, and the minimum expected size Δ of the planes in the world (lines 15-16). A version of the *RANSAC* algorithm [numInliers, $\hat{P}$, N]$\leftarrow$ RANSAC[p0, W, H, l, ϵ] is used for providing a set of 3D inlier points $\hat{P}$ that make part of the filtered plane, its corresponding normal N and the total number of inlier points *numInliers*. In addition to *p0*, W and H this subroutine requires also the number of local samples l and the plane distance threshold ϵ. If the total number of inlier points *numInliers* is greater than a defined threshold $\alpha_{in} \times l$, where α_{in} is the minimum fraction of inlier points to accept a set of sampled points, then all the plane inlier points $\hat{P}$ are added to the list F_P, its normals to the filtered planes N are added to the list N_P, and the the number of filtered plane points n is incremented (lines 19-21), otherwise all image filtered points $\hat{P}$ are added to the outlier points list O_P (line 23). When the while condition (line 7) turns false, the algorithm exits the iterative loop and the three lists containing all the image plane filtered points F_P, the corresponding normals to planes N_P, and image outlier points O_P are returned as output parameters (line 26). This algorithm was the starting point to the work developed in Chapter 3.

Algorithm 1 Fast Sampling Plane Filtering from [6]

1: **procedure** PLANEFILTERING(I)
2: $F_P \leftarrow \{\}$ ▷ Plane filtered points
3: $N_P \leftarrow \{\}$ ▷ Normals to filtered planes
4: $O_P \leftarrow \{\}$ ▷ Outlier points
5: $n \leftarrow 0$ ▷ Number of plane filtered points
6: $k \leftarrow 0$ ▷ Number of neighborhoods sampled
7: **while** $(k < k_{max}) \wedge (n < n_{max})$ **do**
8: $k \leftarrow k + 1$
9: p0 = [rand(0,h-1),rand(0,w-1)]
10: p1 = p0 + [rand(-Ω,+Ω),rand(-Ω,+Ω)]
11: p2 = p0 + [rand(-Ω,+Ω),rand(-Ω,+Ω)]
12: Reconstruct s0, s1, s2 from p0, p1, p2 ▷ Using equations (2.1), (2.2) and (2.3)
13: $r = \frac{(s_1 - s_0) \times (s_2 - s_0)}{\|(s_1 - s_0) \times (s_2 - s_0)\|}$ ▷ Compute plane normal
14: $\bar{z} = \frac{s_0^z + s_1^z + s_2^z}{3}$ ▷ Compute mean depth
15: $W = w\left(\frac{\Delta}{\bar{z}}\right)\tan\left(f_H\right)$ ▷ Search window width
16: $H = h\left(\frac{\Delta}{\bar{z}}\right)\tan\left(f_V\right)$ ▷ Search window height
17: [numInliers, $\hat{P}$, N]← RANSAC[p0, W, H, l, ϵ]
18: **if** $numInliers > \alpha_{in} l$ **then**
19: Add $\hat{P}$ to F_P
20: Add N to N_P
21: $n \leftarrow n + numInliers$
22: **else**
23: Add $\hat{P}$ to O_P
24: **end if**
25: **end while**
26: **return** F_P, N_P, O_P
27: **end procedure**

2.5 Obstacle Scan

The Obstacle Scan algorithm was developed by Biswas [4] . Using the filtered points of the depth image provided by the FSPF it is possible to generate an obstacle scan similar to the laser rangefinder scan. This obstacle scan will be a list of n_N points where each point $S_N[j]$ value is the distance of the closest obstacle along the ray from the robot in the direction of $f_H\left(\frac{i}{n_N} + \frac{1}{2}\right)$. The Obstacle Scan covers an angular field of view f_H with n_N bins, thus providing an angular resolution of $\frac{f_H}{n_N}$. The **Algorithm** 2 lists the Obstacle Scan procedure that receives a list of image filtered plane points F_P its input parameter (line 1). A list of n_N points is initialized with a default value (infinite) for every one of its elements (line 2). For all elements of F_P an interactive procedure will start (lines 4-10). If any point of F_P has a distance smaller than the minimum depth value ϵ_N then this point is disregarded (line 5). If not, the angle θ between the y-axis and the x-axis of the camera frame is computed (line

7), followed by the distance ρ between the camera and the current point (line 8). The index j of the current point is computed based on the number of elements n_N of the scan points S_N, its θ and the camera's horizontal field of view f_H (line 9). The distance value of the scan for a defined direction $S_N[j]$ is given by the minimum value between the current $S_N[j]$ value and the current ρ value (line 10). The Obstacle Scan algorithm returns as its output parameter, a the list of distance points S_N (line 10). This algorithm was also the basis for the algorithm developed in Section 3.2.

Algorithm 2 Obstacle Scan from [4]

1: **procedure** ObstacleScan(F_P)
2: $S_N[j] \leftarrow \infty \forall_j \in [0, n_N]$
3: **for all** $s_i \in F_P$ **do**
4: **if** $s_i^z < \epsilon_N$ **then**
5: continue
6: **end if**
7: $\theta = atan2(s_i^y, s_i^x)$
8: $\rho = \sqrt{(s_i^x)^2 + (s_i^y)^2}$
9: $j = floor\left[n_N \left(\frac{\theta - \frac{f_H}{2}}{f_H}\right)\right]$
10: $S_N[j] \longleftarrow min(S_N[j], \rho)$
11: **end for**
12: **return** S_N
13: **end procedure**

3 Depth Image Processing

This chapter describes the developed algorithms and the code that processes the depth images by detecting some features such as planes from the image scenario and their corresponding points.

The plane detection from depth images and the obstacle scan generation algorithms developed provide important and useful data that could be applied in some future work such as indoor mobile robot localization and navigation tasks.

3.1 Fast Sampling Plane Filtering Plus

The new FSPF+ algorithm was developed based on the FSPF [4] presented in Section 2.4. The FSPF+ has some improvements regarding the plane point filtering possibilities and also the depth image point preprocessing.

The description of the FSPF+ algorithm is supported by Figure 3.1, where the algorithm is generally presented and by **Algorithm** 3, where the algorithm is detailed and the main changes are outlined in bold. All the configuration parameters required by the FSPF+ are listed in Table 3.1.

It starts by taking a depth image I of size $h \times w$, a maximum depth value *maxDepth*, a minimum depth value *minDepth* and a maximum distance threshold value ϵ as its input

Table 3.1: Configuration parameters for **Algorithm** 3

Parameter	**Description**
nMin	Minimum number of points to consider a plane
nRequired	Minimum number of points that has to be analyzed
Ω	Neighborhood for global samples (in pixels)
Δ	Plane size in world space for local samples (in m)
ϵ	Maximum distance threshold for planes (in mm)
β	Minimum number of points to generate a mathematical model
f_H	Kinect 2 horizontal field of view (in degrees)
f_V	Kinect 2 vertical field of view (in degrees)

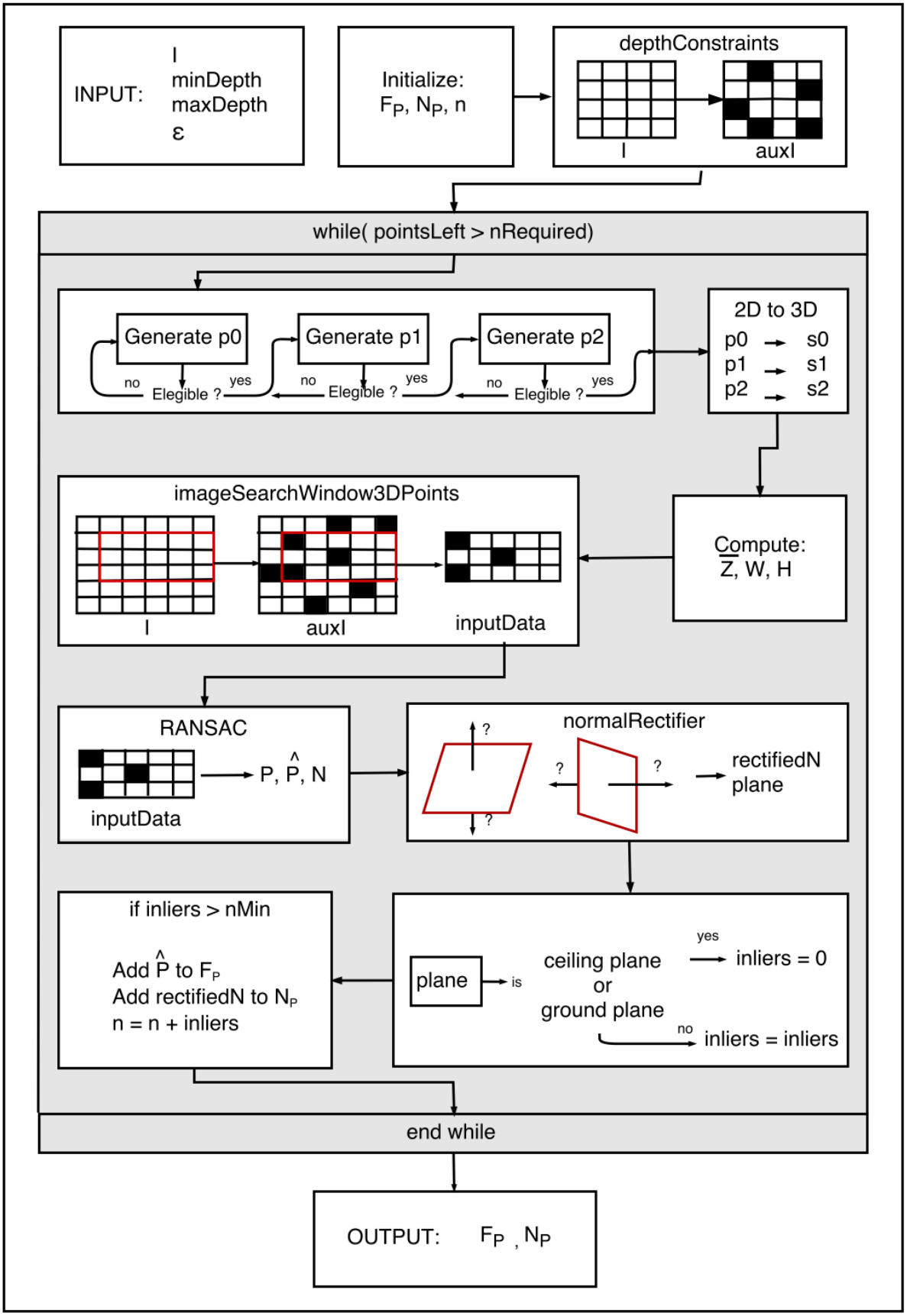
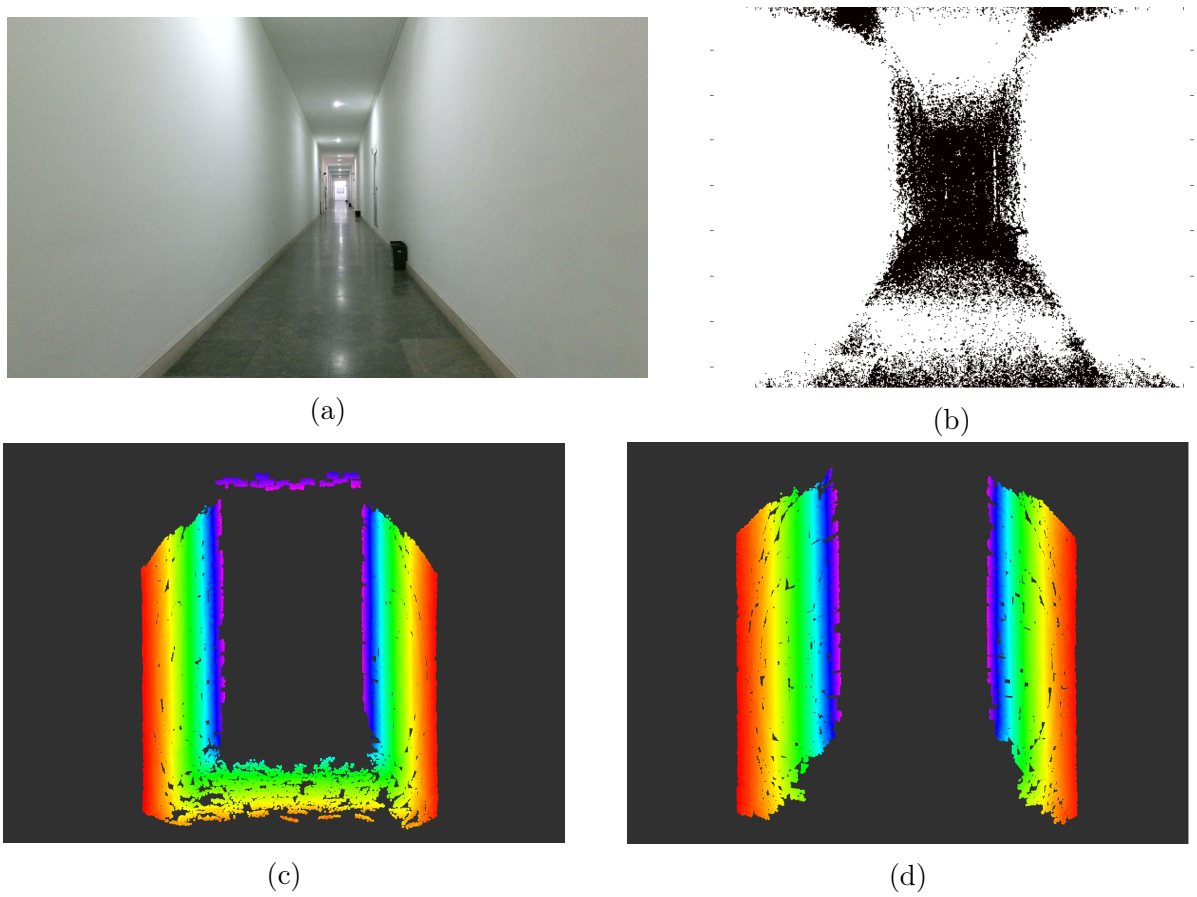


Figure 3.1: General structure of FSPF+.

parameters (line 1). Then two lists are initialized, one that will contain all the image plane filtered points F_P and another one with its normals to the filtered planes N_P (lines 2-4). The variable that will work as the stopping criterion *pointsLeft* is initialized with the total number of the depth image pixels (line 5). A subroutine [auxI]← depthConstraints[I, minDepth, maxDepth] is used to classify all the pixels from the image as "eligible" or "illegible" according to their depth value and *maxDepth*, *minDepth* input values (line 6). While the total number of unfiltered points *pointsLeft* is smaller than a given threshold *nRequired* the algorithm will perform an iterative procedure (lines 8-26). This approach continues by sampling three points from I: *p0* a random eligible point, *p1* and *p2* two also random eligible points selected within a neighborhood of size Ω around *p0* (lines 8-10). The 3D point coordinates [s^x, s^y, s^z] for each 2D point [*p0*, *p1*, *p2*] are then computed (line 11) according to the conversion equations 2.1 , 2.2 and 2.3. The size of a search window is defined based on the mean depth of the latter computed 3D coordinates, the size of I, both horizontal f_H and vertical f_V fields of view and the minimum expected plane size in world Δ (lines 12-14). A second subroutine [points]← imageSearchWindow3DPoints[p0, W, H, I, auxI, f_H, f_V] selects all eligible depth image points within a search window of size W x H around *p0* and computes the corresponding 3D points (*inputData*) (line 15). Given these points a *RANSAC* subroutine [P, $\hat{P}$, N]← RANSAC[inputData, ϵ, β] is used providing a set of 3D inlier points $\hat{P}$ that make part of the filtered plane, its corresponding normal N and the three points that define the filtered plane P (line 16). The number of image plane inlier points is defined as the size of the $\hat{P}$ set (line 17). The algorithm uses a fourth subroutine [rectifiedN, plane]← normalRectifier[N, P] which classifies each filtered plane as a ceiling plane, ground plane, left plane, right plane or front plane (line 18). Whether a plane is classified as ground plane or ceiling plane, both its filtered points and plane normal are discarded (line 20). If the total number of inlier points is bigger than a defined threshold *nMin* then all the plane inlier points are added to the list F_P, their normals to the filtered planes are added to the list N_P and both the number of filtered plane points n and the number of sampled neighborhoods k are incremented (lines 23-25). When the while condition ($pointsLeft > nRequired$) turns false (line 7), the algorithm exits the iterative loop and the two lists containing all the image plane filtered points F_P and the corresponding normals to planes N_P are returned as output parameters (line 28). In Figure 3.2 an example of the application of the FSPF+ algorithm is shown. The colors in which the FSPF+ results are represented are with gradient effect, from red to purple (hot and cold, respectively). Red represents a very close plane (0.5 meters) and purple represents a plane that is very far away (4.5 meters).

Algorithm 3 Fast Sampling Plane Filtering Plus

procedure PLANEFILTERINGPLUS($I, maxDepth, minDepth, \epsilon$)
  $F_P \leftarrow \{\}$ ▷ Plane filtered points
  $N_P \leftarrow \{\}$ ▷ Normals to filtered planes
  $n \leftarrow 0$ ▷ Number of plane filtered points
  $pointsLeft \leftarrow h \times w$ ▷ Initial number of unfiltered points
  [auxI]$\leftarrow$ **depthConstraints[I, minDepth, maxDepth]**
  while ($pointsLeft > nRequired$) **do**
    p0 = [rand(0,h-1),rand(0,w-1)]
    p1 = p0 + [rand(-Ω,+Ω),rand(-Ω,+Ω)]
    p2 = p0 + [rand(-Ω,+Ω),rand(-Ω,+Ω)]
    Compute s0, s1, s2, from p0, p1, p2
    $\bar{z} = \frac{s_0^z + s_1^z + s_2^z}{3}$
    $W = w\left(\frac{\Delta}{\bar{z}}\right)\tan\frac{f_H}{2}$
    $H = h\left(\frac{\Delta}{\bar{z}}\right)\tan\frac{f_V}{2}$
    [inputData]$\leftarrow$ **imageSearchWindow3DPoints[**$p0, W, H, I, auxI, f_H, f_V$**]**
    [P, $\hat{P}$, N]$\leftarrow$ RANSAC[inputData, ϵ, β]
    $inliers \leftarrow$ size of $\hat{P}$
    [rectifiedN, plane]$\leftarrow$ **normalRectifier[N, P]**
    if ($plane = ground$) $\wedge$ ($plane = ceiling$) **then**
      $inliers = 0$
    end if
    if $inliers > nMin$ **then**
      Add $\hat{P}$ to F_P
      Add rectifiedN to N_P
      $n \leftarrow n + inliers$
    end if
  end while
  return F_P, N_P
end procedure



(a) (b) (c) (d)

Figure 3.2: Input and output FSPF+ algorithm data: (a) Color image. (b) Depth image. (c) Results of FSPF+ algorithm applied to (b). (d) Results of FSPF+ algorithm applied to (b) after filtering the planes classified as ground and ceiling planes.

3.1.1 Contributions

Although the structure of the FSPF+ algorithm is very similar to FSPF's structure [4], some functionalities were added in order to improve the performance of the algorithm in several ways. The new functionalities are the following:

1. Starting from the input parameters of **Algorithm** 3 (line 1), the inclusion of the maximum and minimum depth range values allows the FSPF+ to be easily configured to most of the environment scenes where the algorithm has to run. The same occurs to the ϵ value, allowing the user to decide if the desired results have to be more or less accurate.

2. After working with the Microsoft Kinect [34] for a while and analyzing the provided depth images it was concluded that those images had a high number of pixels with noisy values, in order to overcome this problem it was developed a subroutine that analyzes each pixel of the provided depth image and then classifies each one of them as "eligible" or "illegible" regarding the defined depth range constraints on **Algorithm**

3 (line 5). The purpose of this subroutine is to do an initial image pixel filtering in order to accelerate the main image plane filtering procedure and to disregard the image pixels with noisy values. This subroutine is called *depthConstraints* and is outlined in **Algorithm** 4.

Algorithm 4 Depth Constraints

```
1:  procedure DEPTHCONSTRAINTS(I, minDepth, maxDepth)
2:      auxI ← {}                                   ▷ Empty auxiliar 2D image
3:      w ← width of image I
4:      h ← height of image I
5:      for i = 1 : h do
6:          for j = 1 : w do
7:              if I(i, j) <= minDepth || I(i, j) > maxDepth then
8:                  auxI(i, j) = 1                  ▷ Illegible pixel
9:              else
10:                 auxI(i, j) = 0                  ▷ Eligible pixel
11:             end if
12:         end for
13:     end for
14:     return auxI
15: end procedure
```

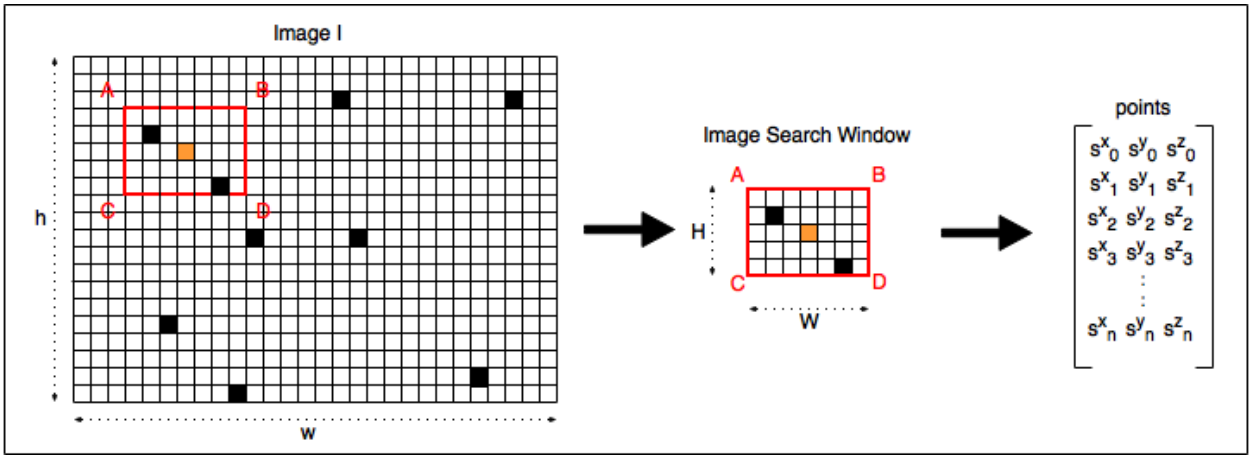
3. The stopping criterion was also modified once the original FSPF stopping criterion 2.4 was not strict enough to guarantee that all image planes were filtered. This way the stopping criterion was defined by the following equation:

$$nRequired = [(h \times w) - illegiblePoints] \times 0.01 \tag{3.1}$$

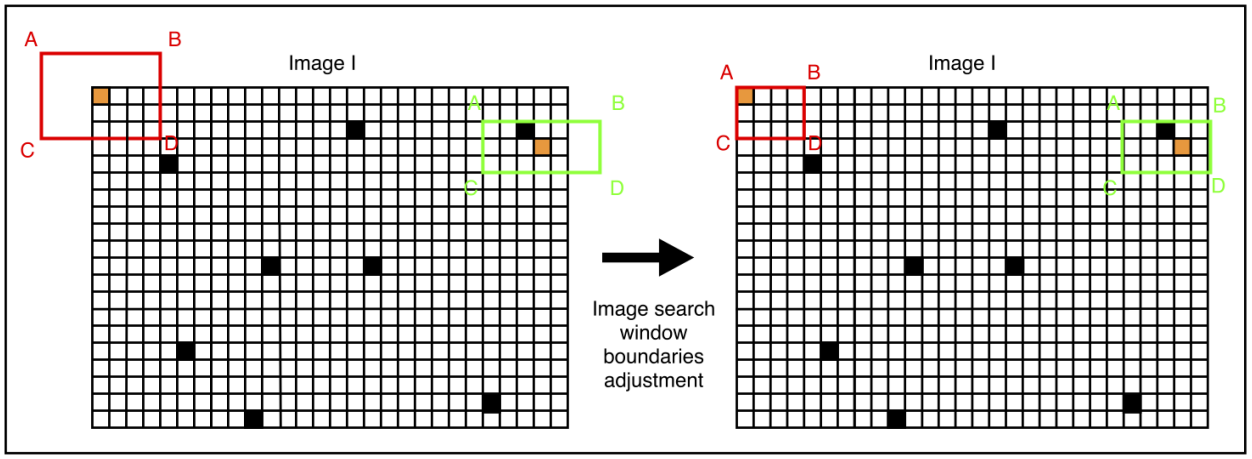
The resulting number of points *nRequired* is good enough to obtain a good detection of planes from the depth image. The fraction of required points (1%) that guarantees a good plane detection was empirically computed.

4. The dataset of 3D points *inputData* that will be provided to the RANSAC algorithm has to be defined based on a search window of size $H \times W$ around a middle point *p0*. However the boundaries of the image represent a problem once the point *p0* is randomly generated. In Figure 3.3a an example where the middle point *p0* (orange pixel) has a localization far from the image's boundaries is shown, this way the definition of the corresponding image search window is direct like the conversion of its 2D points to 3D points *inputData*. During the conversion process of 2D points to their 3D coordinates these points become illegible and the *pointsLeft* variable is decremented.

In situations when the point *p0* is located nearby the image's boundaries the search window sometimes has to be resized in order to not include invalid pixels. In Figure 3.3b two examples of invalid search windows and the adjustment of the same search windows are shown. The conversion procedure is then applied to these search windows. This subroutine is called *imageSearchWindows3DPoints* and outlined in **Algorithm** 5.



(a)



(b)

Figure 3.3: Representation of the imageSearchWindow3DPoints algorithm procedure: (a) p0 (orange pixel) far from the image boundaries. (b) Two examples of image search windows adjustment for p0 (orange pixels) near the image boundaries. The black pixels in both images represent illegible points previously defined.

5. Providing this 3D points dataset *inputData*, the plane distance threshold ϵ and the minimum number of samples β required to a function to fit a mathematical model the RANSAC algorithm computes the best fitting model providing the plane normal N, the plane inlier points $\hat{P}$ and the three 3D points P that define the plane. In the FSPF algorithm, an initial image plane is computed based on the three 3D points previously computed, then the *RANSAC* algorithm will figure it out which search window's points

Algorithm 5 Image Search Window 3D Points

```
1: procedure IMAGESEARCHWINDOW3DPOINTS(p0, W, H, I, auxI, f_H, f_V)
2:     inputData ← {}                                  ▷ Empty set of 3D points
3:     w ← width of image I
4:     h ← height of image I
5:     i ← i coordinate of p0
6:     j ← j coordinate of p0
7:     A ← ceil([i − H/2, j − W/2])                    ▷ Search window left upper corner
8:     B ← ceil([i − H/2, j + W/2])                    ▷ Search window right upper corner
9:     C ← ceil([i + H/2, j − W/2])                    ▷ Search window left bottom corner
10:    D ← ceil([i + H/2, j + W/2])                    ▷ Search window left bottom corner
11:    if I(i,j) is nearby the image's boundaries then
12:        Adjust A, B, C and D values
13:    end if
14:    for i = A(1,1) : C(1,1) do
15:        for j = C(1,2) : D(1,2) do
16:            if auxI(i,j) is Eligible then
17:                Computes s^x, s^y, s^z
18:                auxI(i,j) = 1                       ▷ Image pixel turns illegible
19:                pointsLeft = pointsLeft − 1
20:                inputData = [inputData; s^x s^y s^z]
21:            end if
22:        end for
23:    end for
24:    return inputData
25: end procedure
```

belong to this image plane. This approach could look similar to the one presented in this dissertation however it can be easily proven that if the computed initial 3D image points *s0*, *s1* and *s2* belong to different image planes, then the resultant plane will not correspond to a real image plane. A problem of this nature will not occur when using the subroutine *imageSearchWindow3DPoints* followed by the *RANSAC* implementation as proposed in the FSPF+ **Algorithm** 3.

6. A totally new feature is added to the FSPF+ thats consists in the possibility to classify the image planes provided by the *RANSAC* algorithm. Firstly the localization of plane in the image is computed, then the corresponding normal to the filtered plane is verified, whether its direction is correct or needs to be rectified. Finally the provided image plane is classified as ceiling plane, ground plane, left wall plane, right wall plane or front plane regarding both its normal coefficients and localization. This feature allows the FSPF+ algorithm to filter specific planar structures such as the image ground, ceiling or even frontal structures. In Figure 3.4 the image plane normal rectification

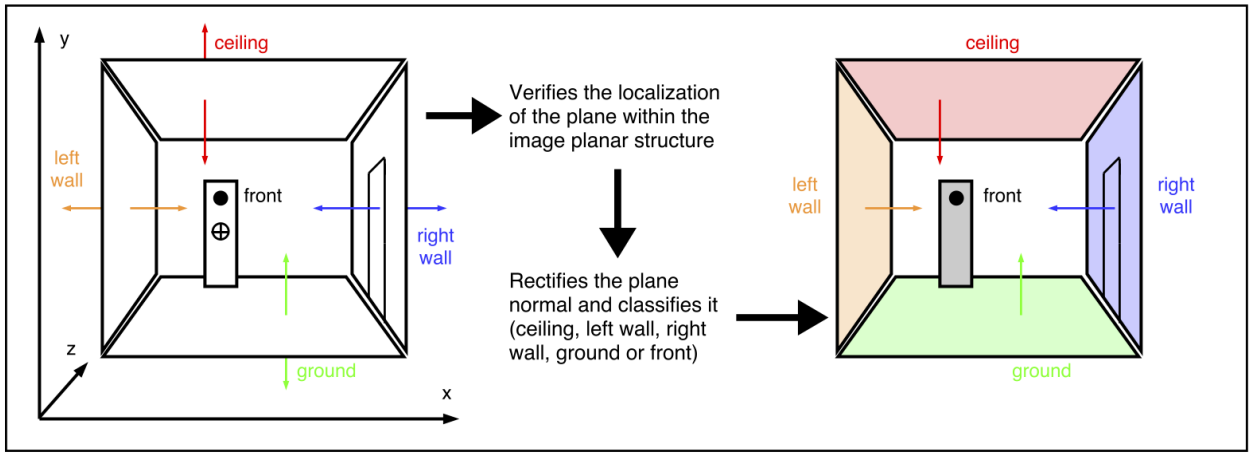


Figure 3.4: normalRectifier procedure description.

and classification are shown. This subroutine is called *normalRectifier* and is outlined in **Algorithm** 6 and a brief procedure description is shown Figure 3.4.

Algorithm 6 Normal Rectifier

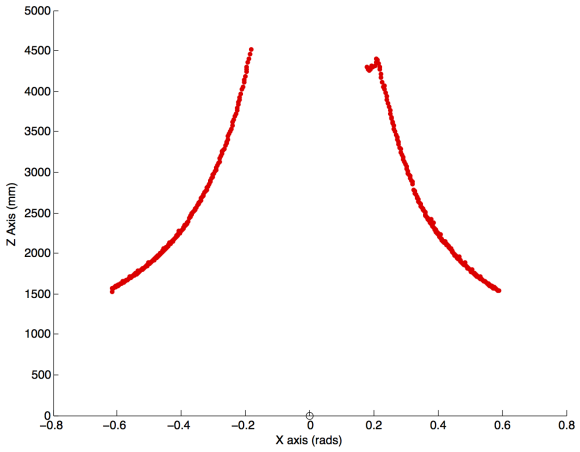
1: **procedure** NORMALRECTIFIER(N, P)
2: $plane \leftarrow \{\}$ ▷ Plane orientation flag
3: $rectifiedN \leftarrow \{\}$
4: Computes x_{mean}, y_{mean}, z_{mean} from P ▷ Plane middle point
5: **if** N has no correct direction **then**
6: Rectifies N by changing its signal
7: rectifiedN ← N rectification by signal changing
8: $Plane \leftarrow classification$
9: **else**
10: rectifiedN ← N
11: $Plane \leftarrow classification$
12: **end if**
13: **return** rectifiedN, plane
14: **end procedure**

3.2 Kinect Obstacle Scan

The Obstacle Scan previously presented in Section 2.5 was studied and a new version was developed, the Kinect Obstacle Scan (KOS). The KOS algorithm allows the user to generate a multi-level obstacle scan. The **Algorithm** 7 receives a list of image filtered plane points F_P provided by the FSPF+, a minimum depth value *minDepth* and the height value *scanHeight* of the desired obstacle scan from the Microsoft Kinect One frame [34] as its input parameters (line 1). A list of n_N points is initialized with a default value (infinite) for every one of its elements (line 2). For all elements of F_P an interactive procedure will start (lines 4-13). If any point of F_P has a distance smaller than the minimum depth value *minDepth*

then this point is disregarded (line 5), the point is also disregarded when its height does not respect the *scanHeight* constraint (line 8). Then the angle θ between the x-axis and the z-axis of the camera frame is computed (line 10), followed by the distance ρ between the camera and the current point (line 11). The index j of the current point is computed based on the number of elements n_N of the scan points S_N, its θ and the camera's horizontal field of view f_H (line 12). The distance value of the scan for a defined direction $S_N[j]$ is given by the minimum value between the current $S_N[j]$ value and the current ρ value (line 13). KOS algorithm returns as its output parameter, a list of distance points S_N (line 15).

(a)



(b)

Figure 3.5: Kinect obstacle scan generated from a depth image: (a) Color image. (b) KOS of (a) for a height of 30 cm from the ground level.

Algorithm 7 Kinect Obstacle Scan

1: **procedure** KINECTOBSTACLESCAN($F_P, minDepth, scanHeight$)
2: $S_N[j] \leftarrow \infty \forall_j \in [0, n_N]$
3: **for all** $s_i \in F_P$ **do**
4: **if** $s_i^z < minDepth$ **then**
5: continue
6: **end if**
7: **if** $((s_i^y < scanHeight - 100) \parallel (s_i^y > scanHeight + 100))$ **then**
8: continue
9: **end if**
10: $\theta = atan2(s_i^x, s_i^z)$
11: $\rho = \sqrt{(s_i^x)^2 + (s_i^z)^2}$
12: $j = ceil\left[n_N \left(\frac{\theta + \frac{f_H}{2}}{f_H}\right)\right]$
13: $S_N[j] \longleftarrow min(S_N[j], \rho)$
14: **end for**
15: **return** S_N
16: **end procedure**

4 Experimental setup

This chapter describes the different experimental setups that were developed to test the proposed methods. There are two different main setups, the Offline Experimental Setup (OES) and the Real-time Experimental Setup (RtES).

4.1 Offline experimental setup

The OES depicted in Figure 4.1 is composed by three main modules: the Data Acquisition Module (DAcM), the Data Processing Module (DPM) and the Data Analysis Module (DAnM).

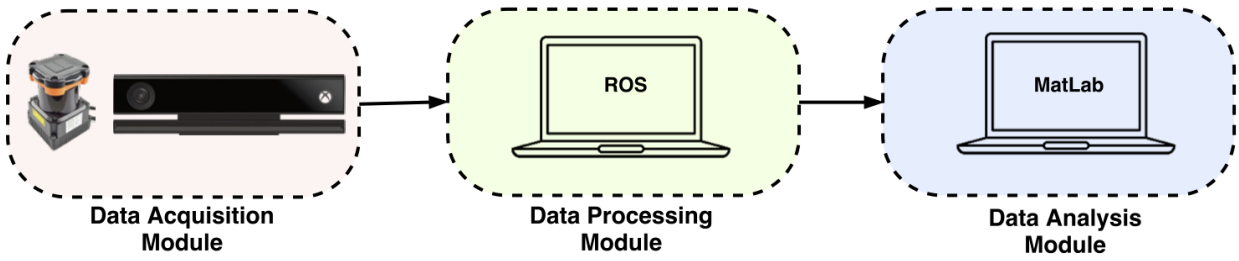


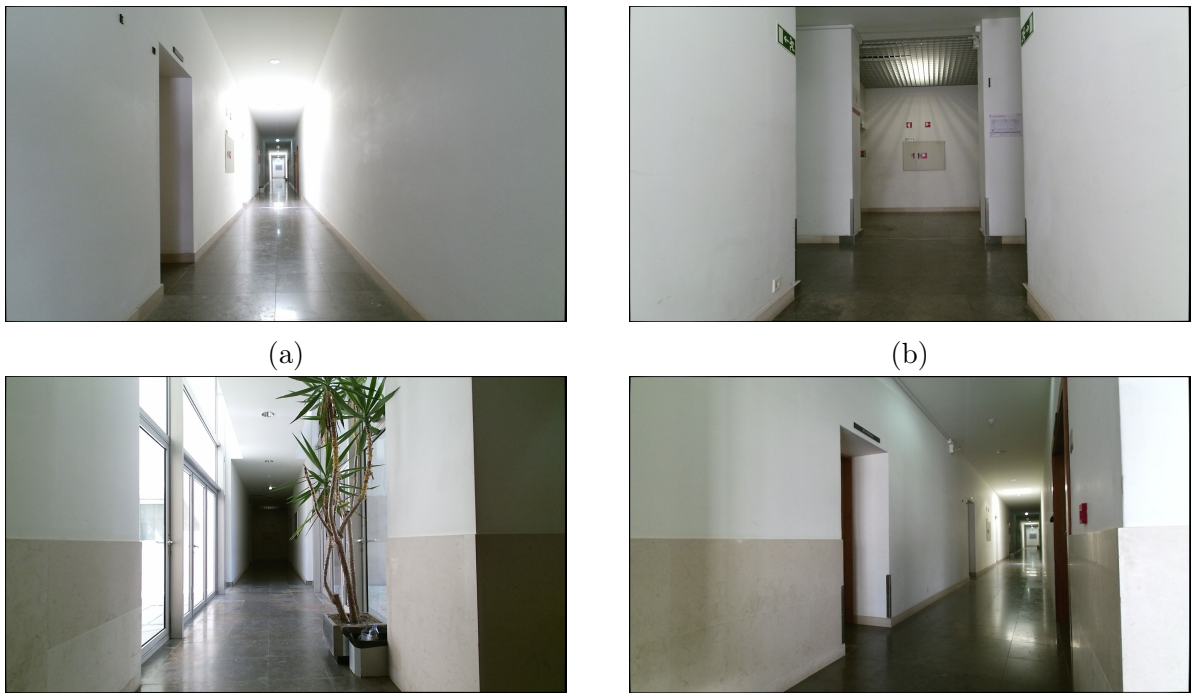
Figure 4.1: Offline experimental setup.

Figure 4.2: Caption of the experimental setups.

4.1.1 DAcM

In the DAcM, a Microsoft Kinect One [34] is used through the *kinect2_bridge* ROS package [49] to collect color and depth images into a Dataset #1 and the corresponding point clouds into a Dataset #2. A Hokuyo UTM-30LX [25] laser range finder is also used to

collect laser scans from the same environments, such as those shown in Figure 4.8, through the *hokuyo_node* ROS package [40] into a Dataset #3. The info flowchart of the DAcM is shown in Figure 4.4.



(a) (b) (c) (d)

Figure 4.3: Ground floor of the ISR. (a) Area A. (b) Area B. (c) Area C. (d) Area D.

4.1.2 DPM

The DPM is a ROS based implementation where the algorithms presented in Chapter 3 are adapted to a ROS supported language in order to process the DAcM's referred datasets. The information flow in the ROS implementation is shown in Figure 4.5 supported by Figure 4.2.

Hence a new *kinect_node* ROS package was created in order to implement the FSPF+ and the KOS algorithms presented in Chapter 3. This package is composed by two nodes:

1. *plane_filtering*: here the FSPF+ algorithm is implemented in C++ programming language. This node subscribes the */kinect2\sd\image_depth_rect* topic [sensor_msgs/Image Message type [41]] from the *iaiKinect2* ROS package [49], publishes */Fp_cloud* and */Np_cloud* topics [sensor_msgs/PointCloud Message type [43]] and exports the *filteredPoints* and the *planeNormals* text files with their values, respectively;

2. *kinect_obstacle_scan*: here the KOS algorithm is implemented. This node subscribes

the */Fp_cloud* topic, publishes the */kinect_scan* topic [sensor_msgs/LaserScan Message [42] type] and exports a text file with its values.

The *Rviz* is a 3D visualization tool for ROS that allows the user to see the values of the subscribed/published ROS topics.

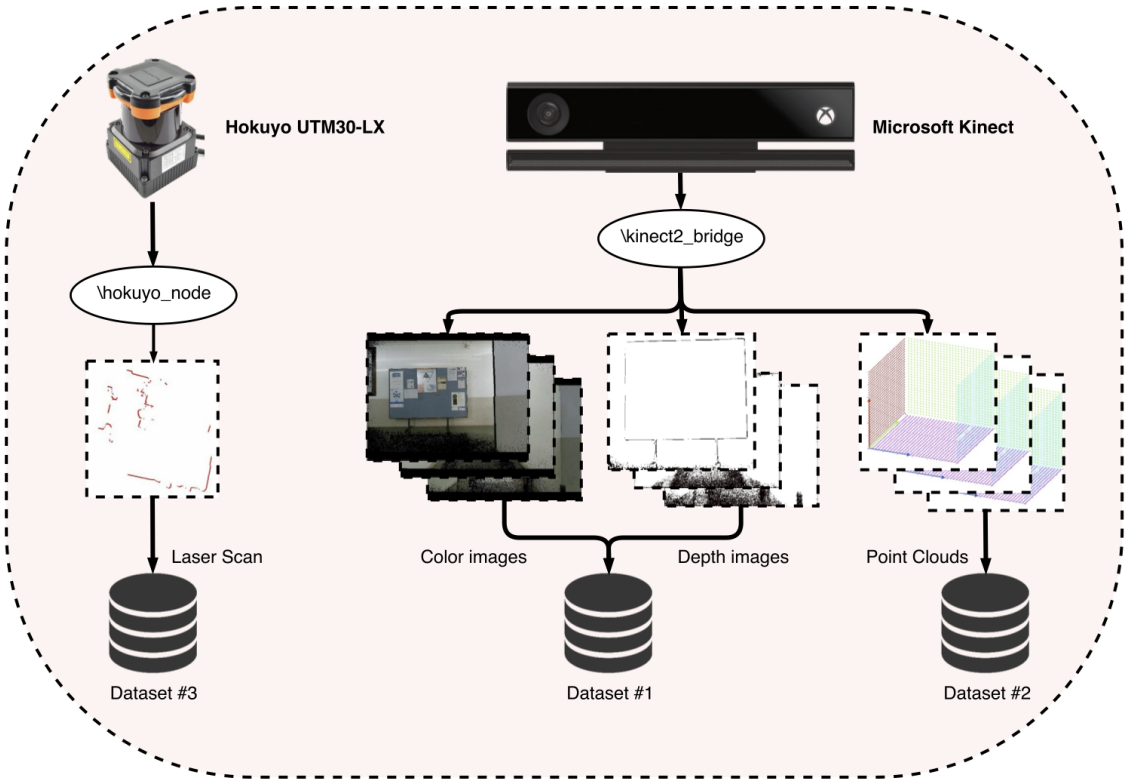


Figure 4.4: Data Acquisition Module.

4.1.3 DAnM

The DAnM analyzes the output data files of DAcM and DPM and computes the results of each desired test based on MatLab [33] routines. The info flowchart of this module is shown in Figure 4.6 also supported by Figure 4.2.

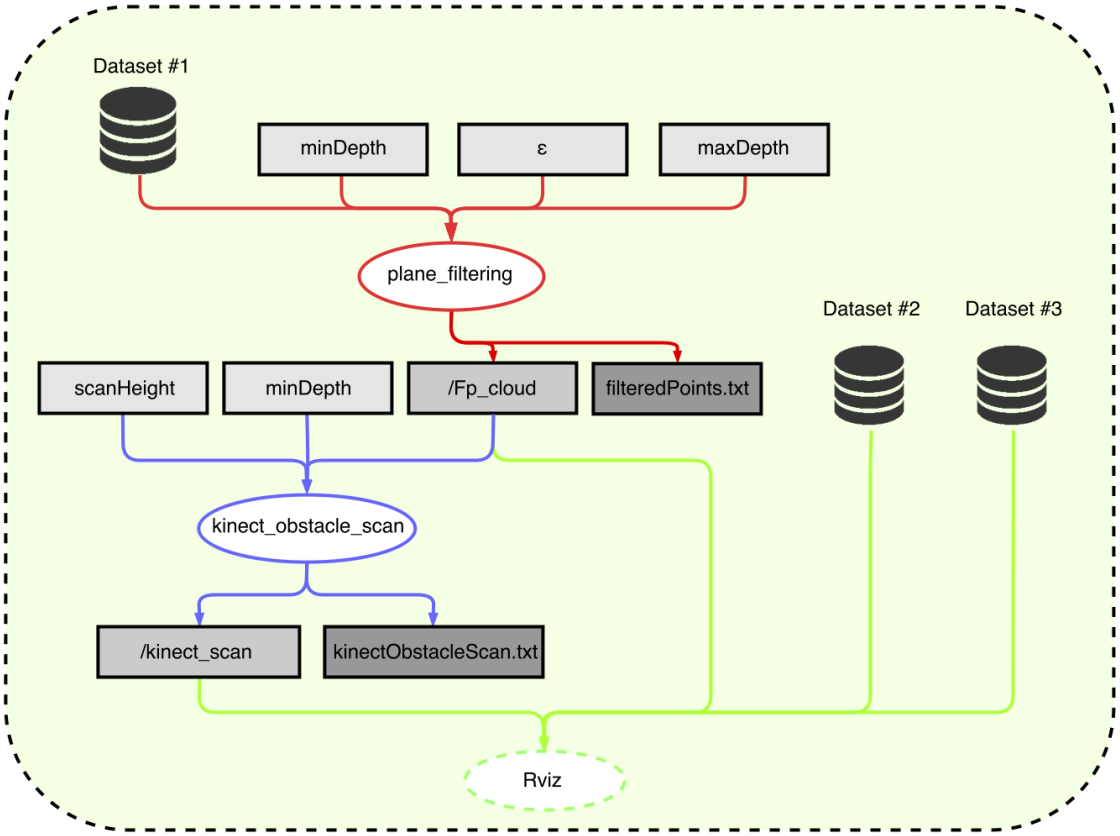


Figure 4.5: Data Processing Module, information flow in ROS implementation.

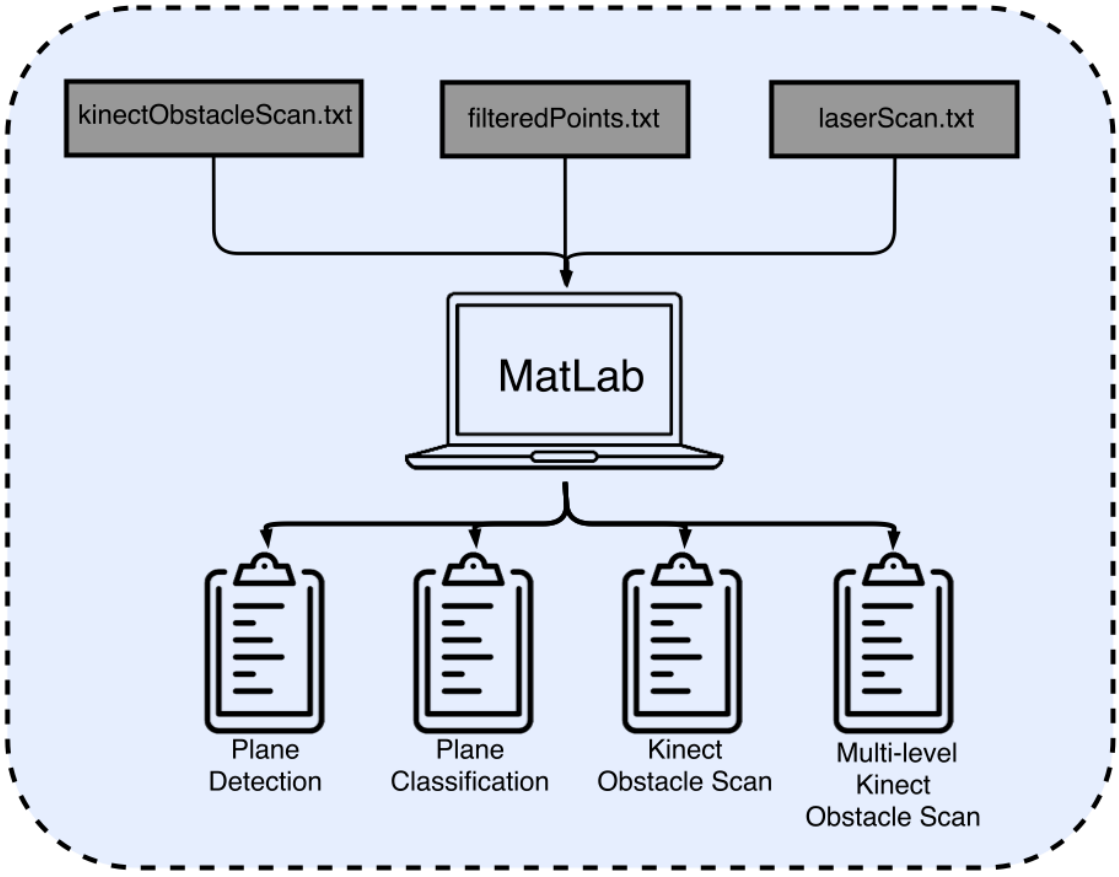


Figure 4.6: Data Analysis Module.

4.2 Real-time experimental setup

The RtES depicted in Figure 4.7 is composed by three main modules: the Data Acquisition Module (DAcM) and the Data Processing Module (DPM) mounted on the Interbot Platform [11] and the Offline Data Analysis Module (ODAnM). The following figures are supported by Figure 4.2.

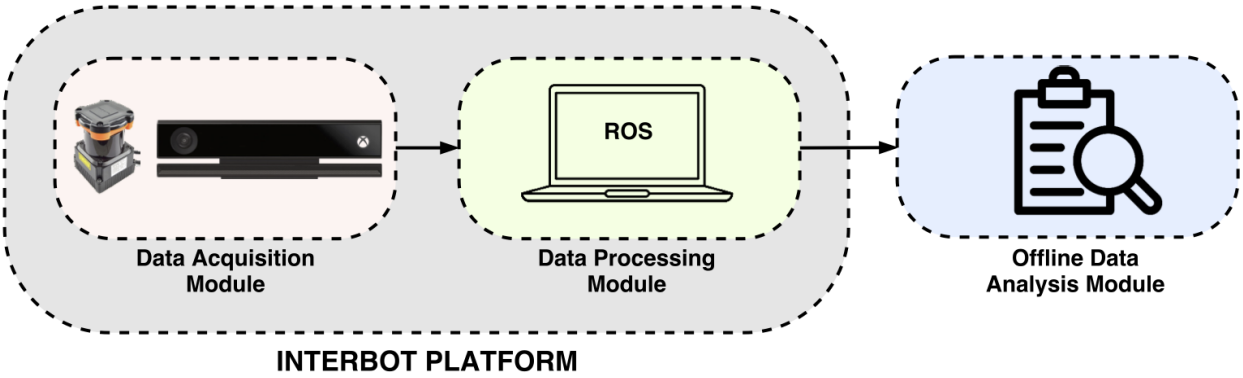


Figure 4.7: Real-time experimental setup.

4.2.1 DAcM

In DAcM the Microsoft Kinect One [34] collects color and depth images and the corresponding point clouds through the use of the *kinect2_bridge* ROS package [49], while the Interbot Platform [11] navigates along the ground floor of ISR, Figure 4.8b, at the same time the Hokuyo laser range finder [25] collects the laser scans using the *hokuyo_node* ROS package [40]. The ROS topics published in the DAcM are then subscribed in the following modules of the RtES. The flowchart of the DAcM is shown in Figure 4.9.

4.2.2 DPM

In DPM the given ROS topics from DAcM are processed in a real-time ROS based implementation presented in Figure 4.10. Some of the ROS packages used in this module were already described in the previous Section 4.1.2.

The *hector_mapping* ROS package [29] which is used on the Interbot Platform [11] for localization and mapping issues is used here as well. This ROS package subscribes the */kinect_scan* topic and publishes the */map* topic [nav_msgs/OccupancyGrid Message type [39]] and the */slam_out_pose* topic [geometry_msgs/PoseStamped Message type [44]]. A png file *generated_map* with the generated map can also be also exported.

The Gazebo robotic simulation tool [17] is used to test the localization of the Interbot Platform using the *hector_mapping* topics in a simulation environment. It is important to refer that the *Rviz* allows a real-time visualization of the processed data.

(a) (b)

Figure 4.8: Data acquisition in the ground floor of the ISR: (a) Interbot Platform. (b) Interbot collecting data.

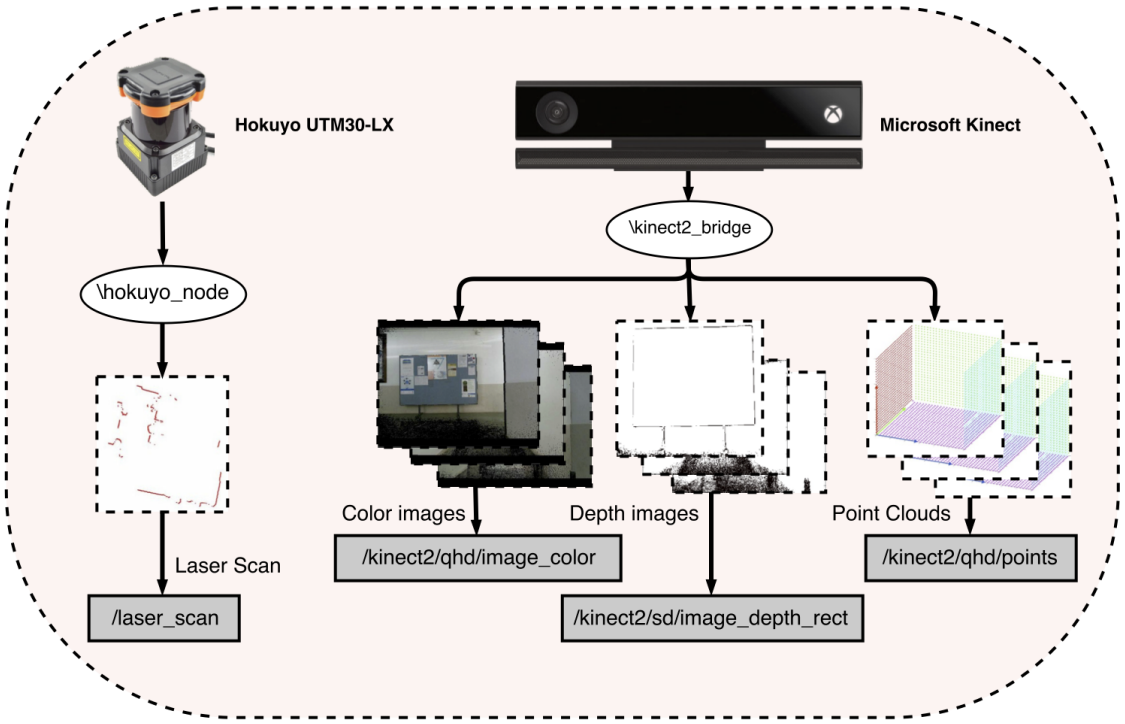


Figure 4.9: Data Acquisition Module.

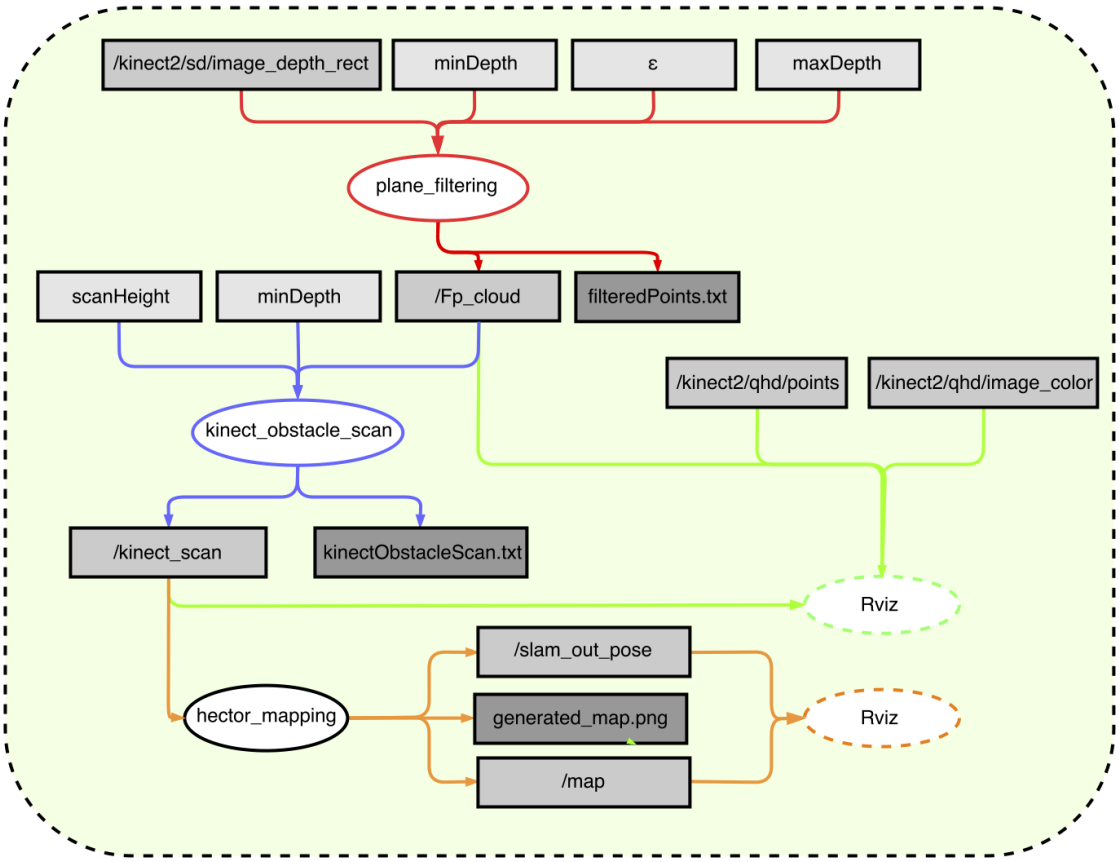


Figure 4.10: Data Processing Module, information flow in ROS implementation.

4.2.3 ODAnN

The I/O image files provided from DPM are analyzed lately through an inspection procedure (not real-time) in order to conclude about the performance of the test done. This analysis is done apart from the Interbot Platform [11]. The data flowchart of ODAnM is presented in Figure 4.11.

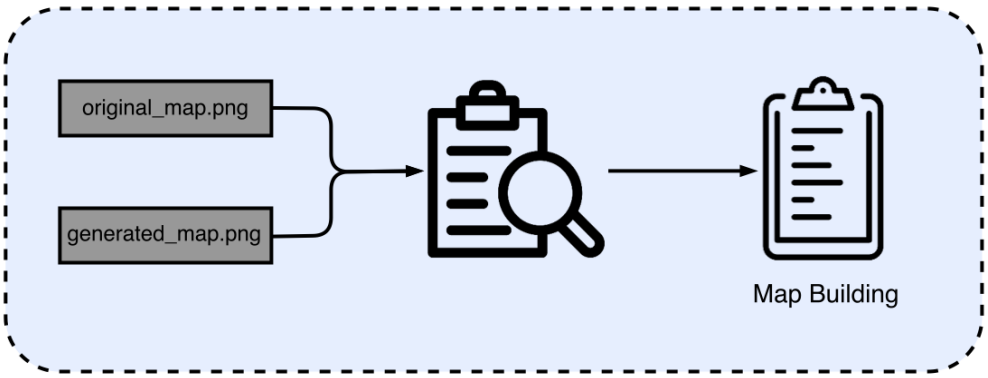


Figure 4.11: Offline Data Analysis Module.

4.3 Interbot Platform

The Interbot Platform [11] is a robotic moving platform developed in the ISR. The motors' drivers, the battery management system and the platform's framework were all developed from scratch for this robot. This platform could be described by three main modules: 1) BMS and motor drivers; 2) Raspberry Pi 2; 3) I/O sensors & laptop. The main components of the Interbot Platform are presented in Figure 4.12 and listed in Table 4.1.

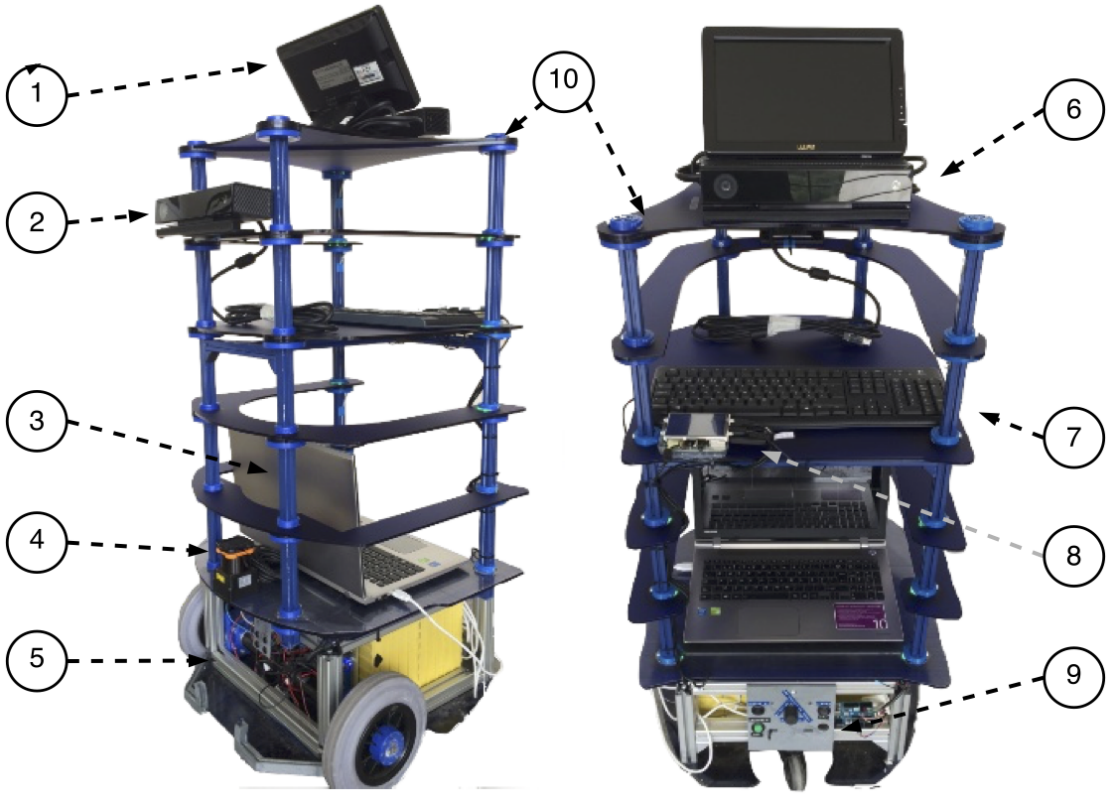


Figure 4.12: Interbot Platform main components.

Table 4.1: Interbot Platform main components.

#	Name	#	**Name**
1	Touchscreen monitor	6	Rear Microsoft Kinect One
2	Front Microsoft Kinect One	7	Keyboard
3	Laptop	8	Raspberry Pi 2
4	Hokuyo UTM30-LX	9	Power interface
5	BMS & Motor Drivers	10	Framework

Further information about the main modules of the Interbot Platform and its components can be found in [11].

5 Tests and results

This chapter presents all the tests that have been done to verify and validate the proposed methods. It is divided in two main sections based on the experimental setups that were used to test the algorithms. Firstly the FSPF+ and KOS algorithms are tested in offline mode, using the OES. Secondly, both algorithms are tested in real-time using the RtES.

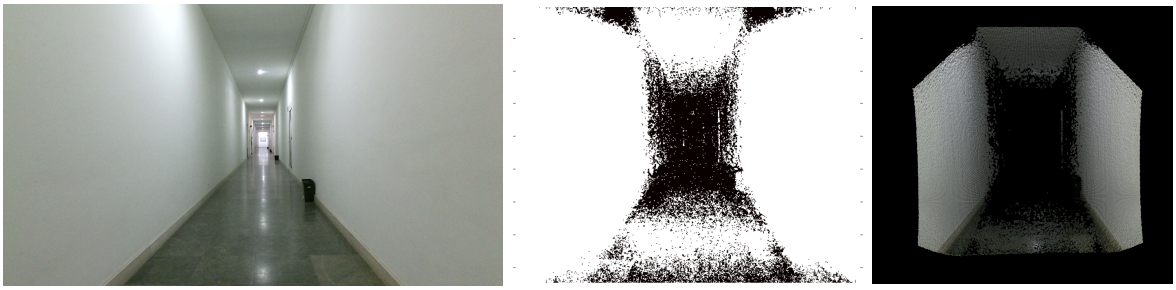
5.1 Offline tests

5.1.1 Plane detection and plane orientation detection

The FSPF+ is tested using the OES. The collected datasets have information about nine scenes from the ground floor of the ISR. The FSPF+ was tested on each one of the nine scenes using a value of ϵ of 5 mm and 10 mm. The defined input parameters of the algorithm are $minDepth = 400$ mm, $maxDepth = 4500$ mm, the configuration values of the FSPF+ are listed in Table 5.1. The main goals of this set of tests is to compare the original scenes with the results of the FSPF+ algorithm regarding the plane detection and plane's orientation detection, and to verify the influence of the ϵ value. For each scene, the color and depth image, and point cloud are presented. Next to this original data two sets of figures are depicted presenting, for the two values of ϵ, the results of the algorithm before and after the plane's orientation detection. As referred before the colors in which the FSPF+ results are represented are with gradient effect, from red to purple (hot and cold, respectively). Red represents a very close plane (0.5 meters) and purple represents a plane that is very far away (4.5 meters).

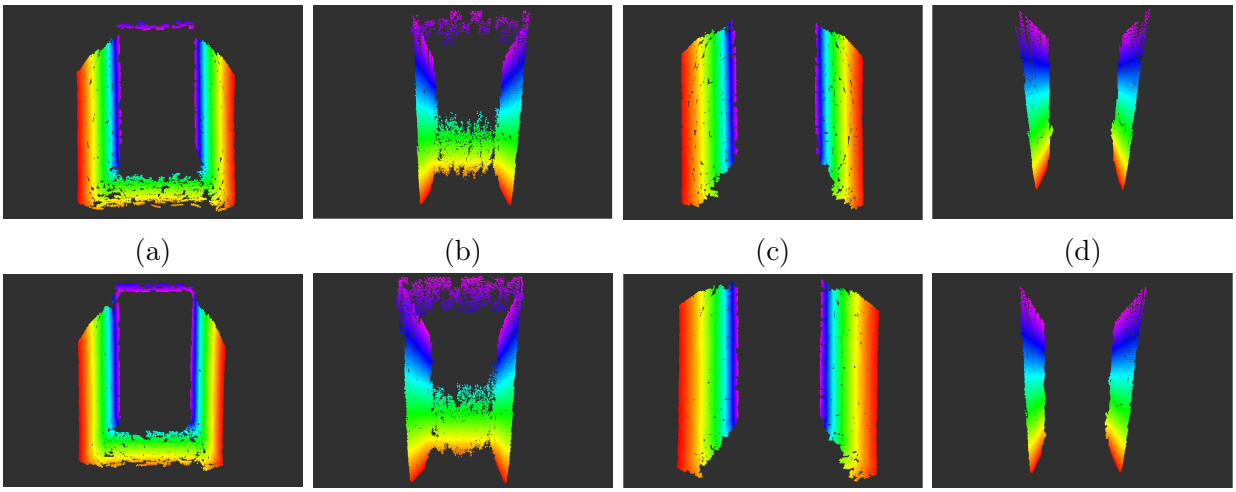
Table 5.1: Configuration values of **Algorithm** 3

Parameter	Value	Parameter	Value
nMin	40	ϵ	5-10
nRequired	99 % of eligible points	β	3
Ω	60	f_H	70.6
Δ	0.5	f_V	60



(a) (b) (c)

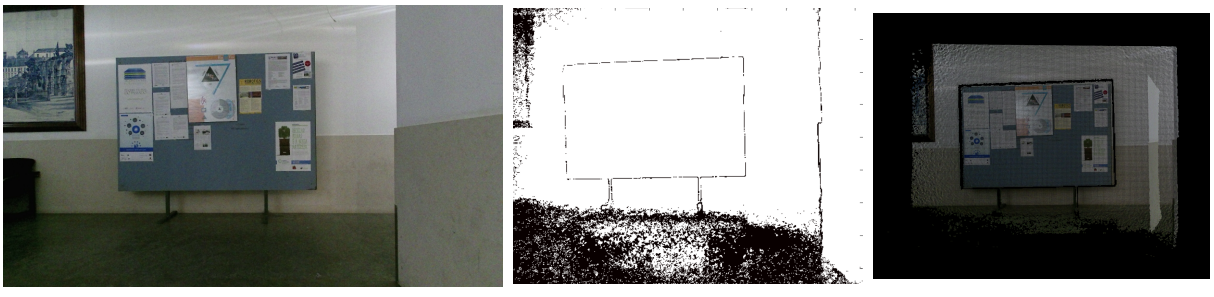
Figure 5.1: Scene #1: (a) Color image. (b) Depth image. (c) Point cloud.



(a) (b) (c) (d)

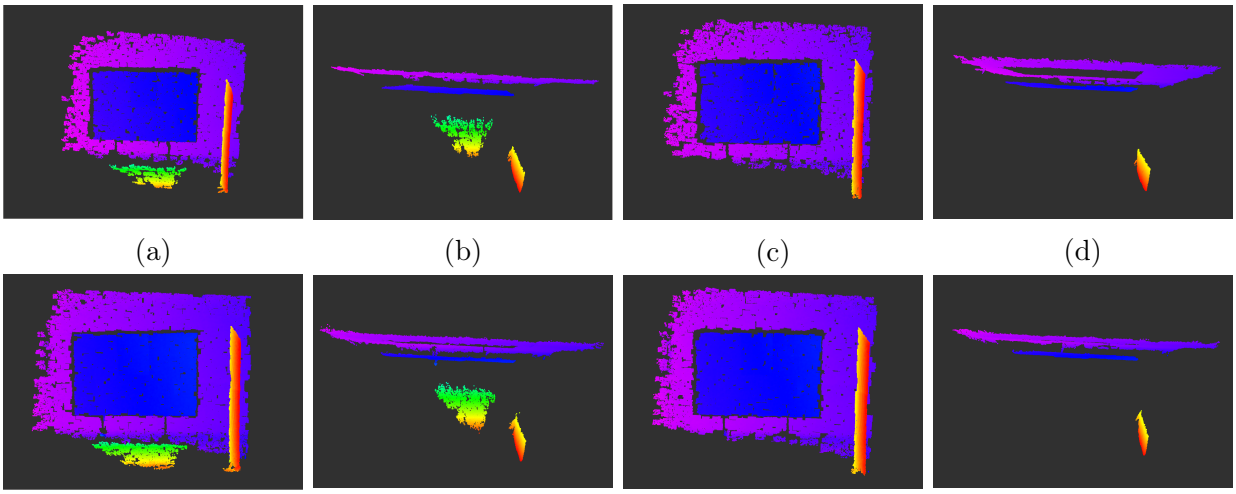
(e) (f) (g) (h)

Figure 5.2: Results of scene #1 for $\epsilon = 5$ (a) to (d) and for $\epsilon = 10$ (e) to (h) . (a)-(e) Front view of filtered points. (b)-(f) Top view of the filtered points. (c)-(g) Front view of filtered points after plane classification. (d)-(h) Top view of filtered points after plane classification.



(a) (b) (c)

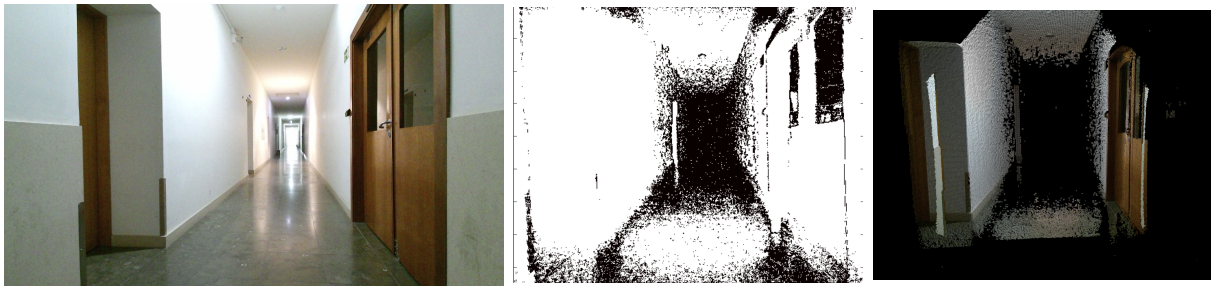
Figure 5.3: Scene #2: (a) Color image. (b) Depth image. (c) Point cloud.



(a) (b) (c) (d)

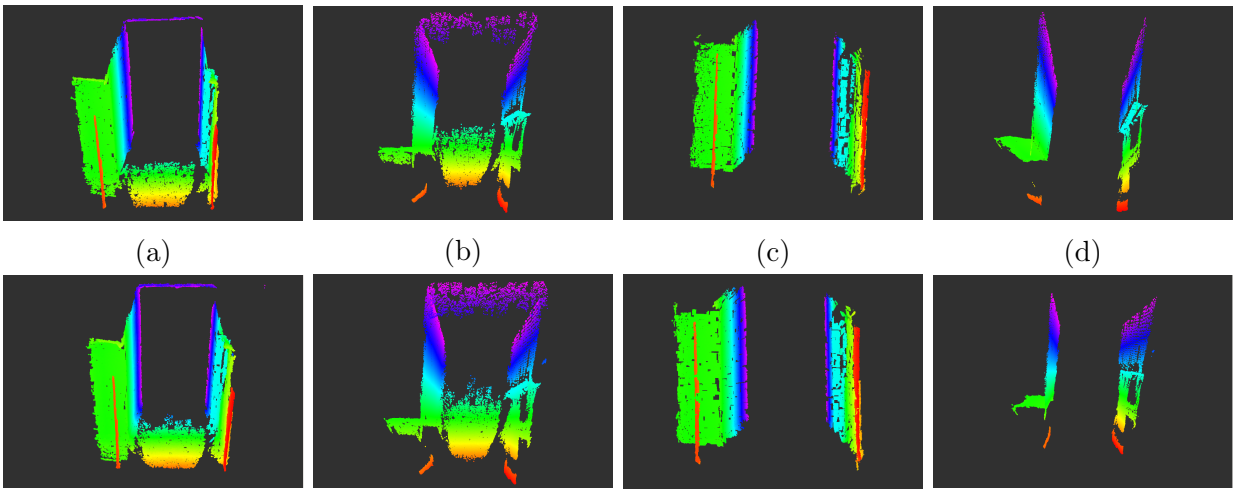
(e) (f) (g) (h)

Figure 5.4: Results of scene #2 for $\epsilon = 5$ (a) to (d) and for $\epsilon = 10$ (e) to (h) . (a)-(e) Front view of filtered points. (b)-(f) Top view of the filtered points. (c)-(g) Front view of filtered points after plane classification. (d)-(h) Top view of filtered points after plane classification.



(a) (b) (c)

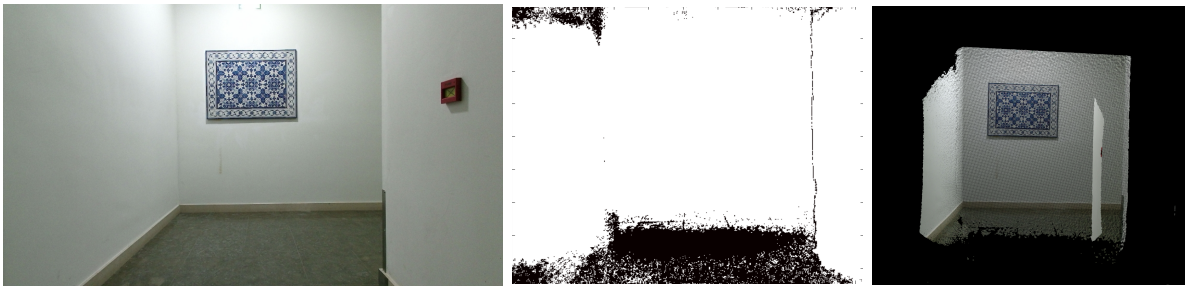
Figure 5.5: Scene #3: (a) Color image. (b) Depth image. (c) Point cloud.



(a) (b) (c) (d)

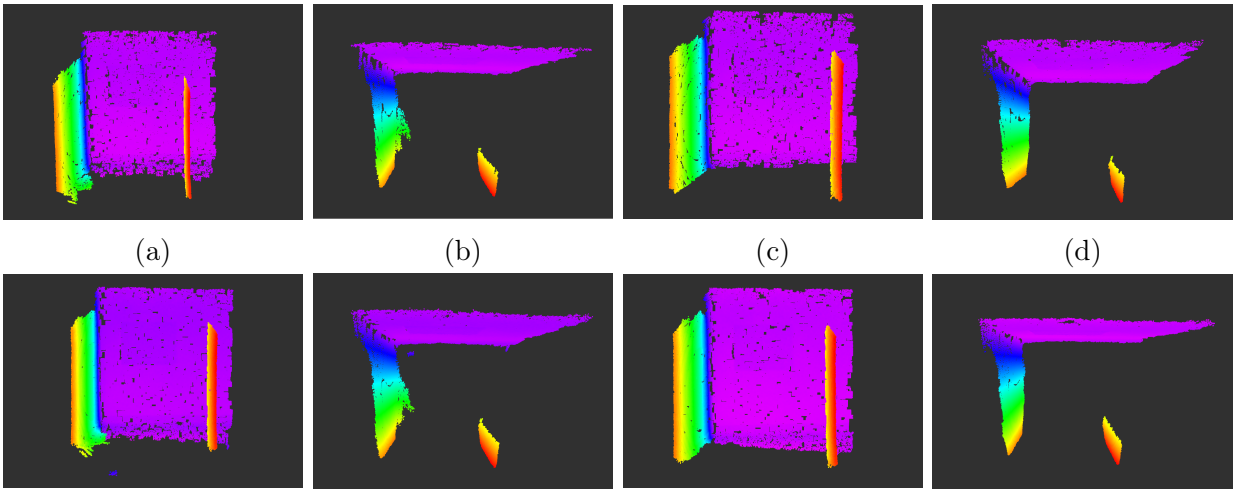
(e) (f) (g) (h)

Figure 5.6: Results of scene #3 for $\epsilon = 5$ (a) to (d) and for $\epsilon = 10$ (e) to (h) . (a)-(e) Front view of filtered points. (b)-(f) Top view of the filtered points. (c)-(g) Front view of filtered points after plane classification. (d)-(h) Top view of filtered points after plane classification.



(a) (b) (c)

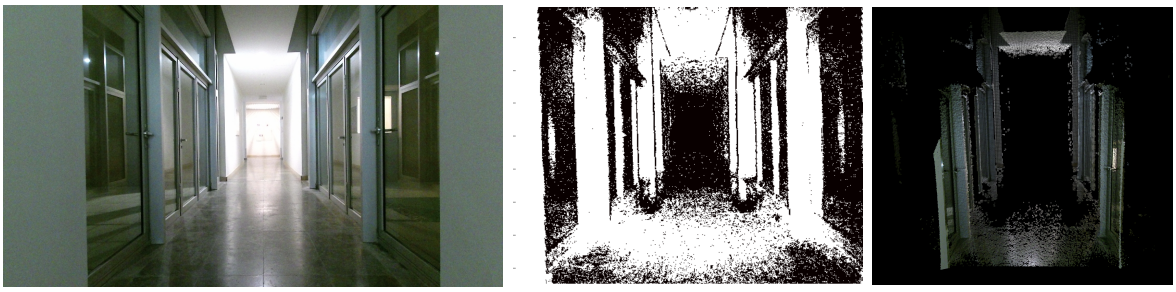
Figure 5.7: Scene #4: (a) Color image. (b) Depth image. (c) Point cloud.



(a) (b) (c) (d)

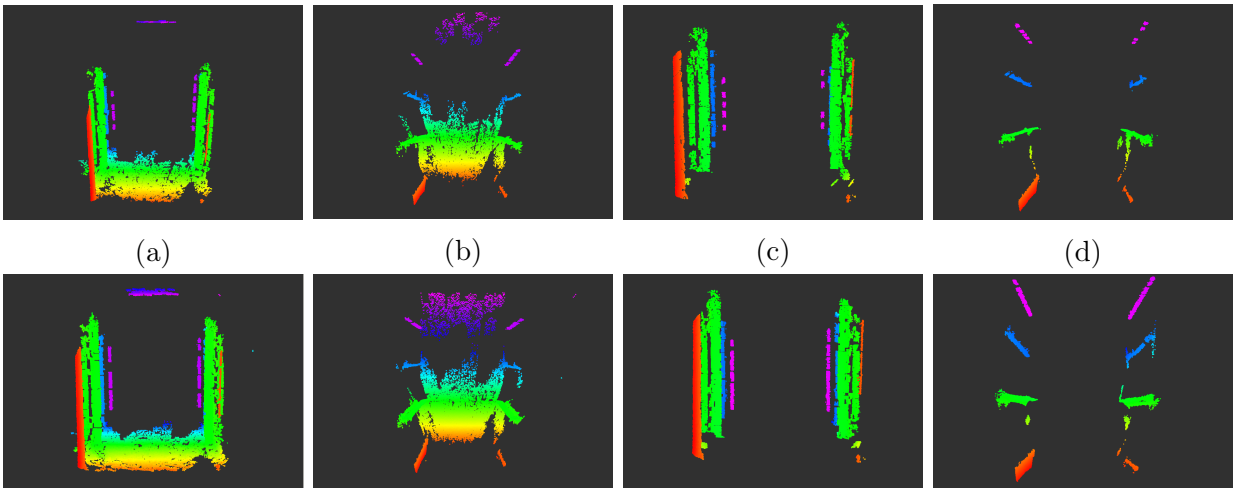
(e) (f) (g) (h)

Figure 5.8: Results of scene #4 for $\epsilon = 5$ (a) to (d) and for $\epsilon = 10$ (e) to (h) . (a)-(e) Front view of filtered points. (b)-(f) Top view of the filtered points. (c)-(g) Front view of filtered points after plane classification. (d)-(h) Top view of filtered points after plane classification.



(a) (b) (c)

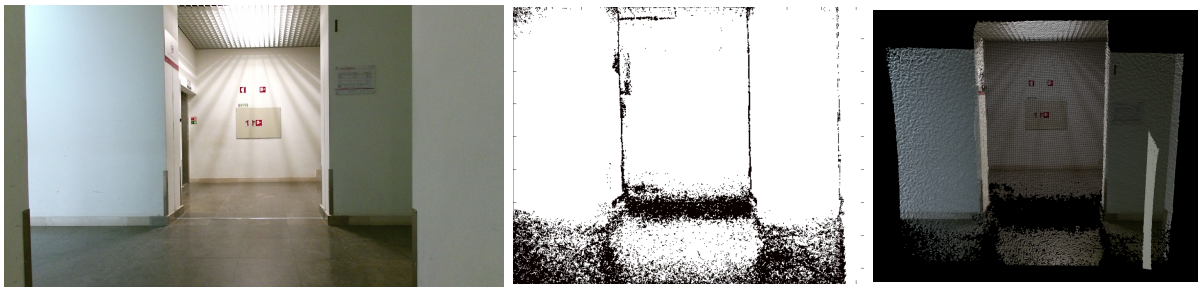
Figure 5.9: Scene #5: (a) Color image. (b) Depth image. (c) Point cloud.



(a) (b) (c) (d)

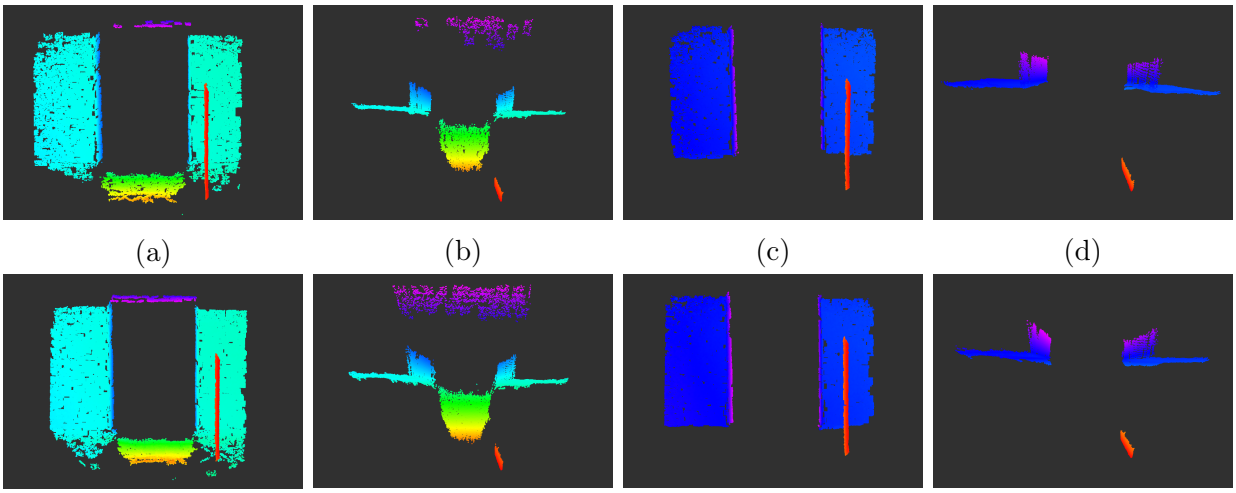
(e) (f) (g) (h)

Figure 5.10: Results of scene #5 for $\epsilon = 5$ (a) to (d) and for $\epsilon = 10$ (e) to (h) . (a)-(e) Front view of filtered points. (b)-(f) Top view of the filtered points. (c)-(g) Front view of filtered points after plane classification. (d)-(h) Top view of filtered points after plane classification.



(a) (b) (c)

Figure 5.11: Scene #6: (a) Color image. (b) Depth image. (c) Point cloud.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 5.12: Results of scene #6 for $\epsilon = 5$ (a) to (d) and for $\epsilon = 10$ (e) to (h) . (a)-(e) Front view of filtered points. (b)-(f) Top view of the filtered points. (c)-(g) Front view of filtered points after plane classification. (d)-(h) Top view of filtered points after plane classification.

Table 5.2: Results of the FSPF+ algorithm for $\epsilon = 5$.

Scene #	Image points	Illegible points	Inlier points	Outlier points	Remaining points	Elapsed time (ms)
1	217088	24.7%	46.58%	27.94%	1634	89.5
2		19.9%	44.5%	34.66%	1736	122.9
3		31.6%	33.94%	33.77%	1484	134.2
4		10.9%	59.9%	26.54%	1932	104.9
5		55.99%	13.8%	29.76%	955	120.8
6		36.85%	31.98%	30.52%	1370	130.8

Table 5.3: Results of the FSPF+ algorithm for $\epsilon = 10$.

Scene #	Image points	Illegible points	Inlier points	Outlier points	Remaining points	Elapsed time (ms)
1	217088	24.7%	56.41%	18.86%	1634	41.80
2		19.9%	58.61%	21.39%	1736	57.61
3		31.6%	41.65%	26.74%	1484	53.99
4		15.6%	73.8%	15.23%	1932	52.23
5		55.99%	18.99%	25.09%	955	53.59
6		36.85%	40%	23.13%	1370	54.07

After reviewing the planes detection results it is possible to conclude that these results are quite satisfactory comparing the same with the the point cloud of each scene. As expected, the results for $\epsilon = 5$ reveal to be better look than for ϵ once the number of points per plane decreases and the plane detection acquracy increases. Moreover it is also possible to refer that the density of planes increases for $\epsilon = 10$.

In all these scenes, planes classified as ground and ceiling planes are filtered as it could be seen in figures (c), (d), (g) and (h) of all scene results figures. The plane orientation detection reveals to be efficient once these kind of planes were removed almost completely.

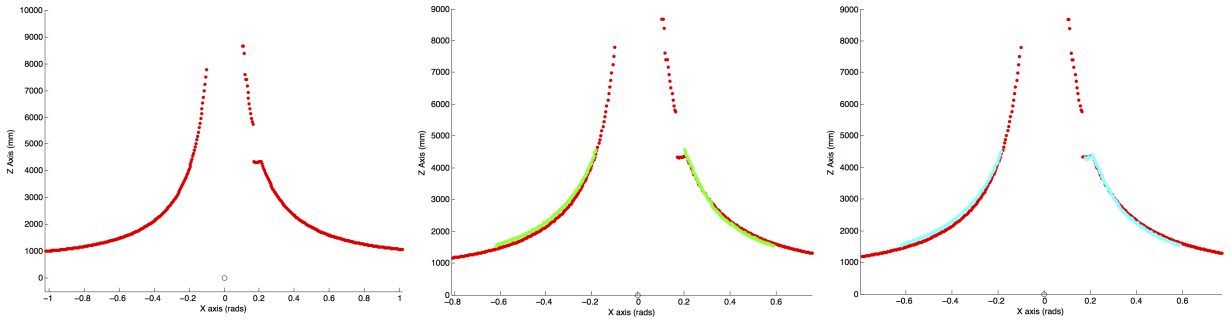
In terms of time consumption, as it is presented in Table 5.2 and in Table 5.3 as the ϵ value doubles from 5 to 10 the elapsed time decreases in a 0.45 scale factor.

Considering the results of Table 5.3, the FSPF+ algorithm runs at 19 Hz which is considered to be a real time procedure.

5.1.2 Kinect obstacle scan

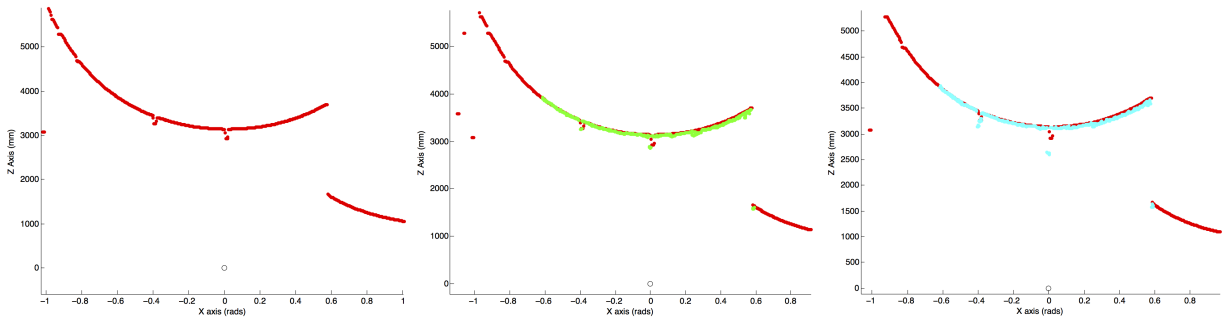
In this section the KOS algorithm is tested using the OES. The input parameters of this algorithm were defined as *minDepth* = 500 mm and *scanHeight* = -0.500 m. The results of the algorithm for both values of ϵ are presented also the Hokuyo's laser scans of the same six scenes are depicted in the following figures. The goal of this set of tests is to compare

the KOS with the Hokuyo's scan and to verify how could the KOS's performance improve or worsen for different values of ϵ. These results are depicted in Figures 5.13 up to Figure 5.18.



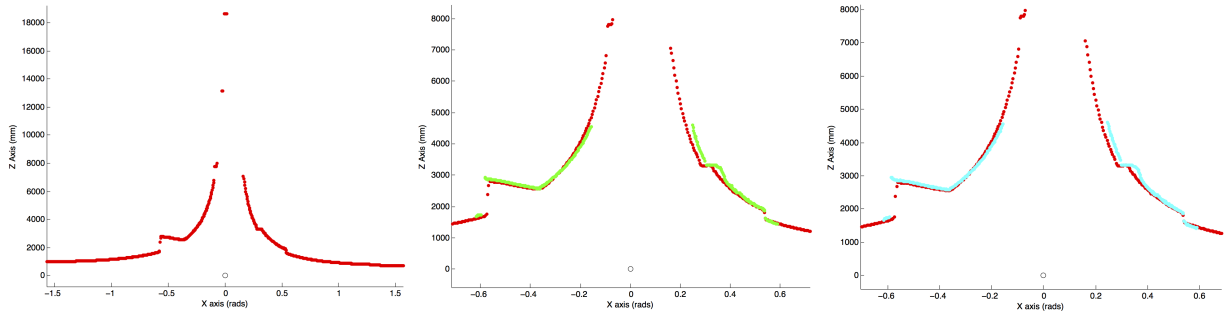
(a) (b) (c)

Figure 5.13: KOS results of scene #1: (a) Laser scan. (b) KOS for $\epsilon = 5$ over the laser scan. (c) KOS for $\epsilon = 10$ over the laser scan.



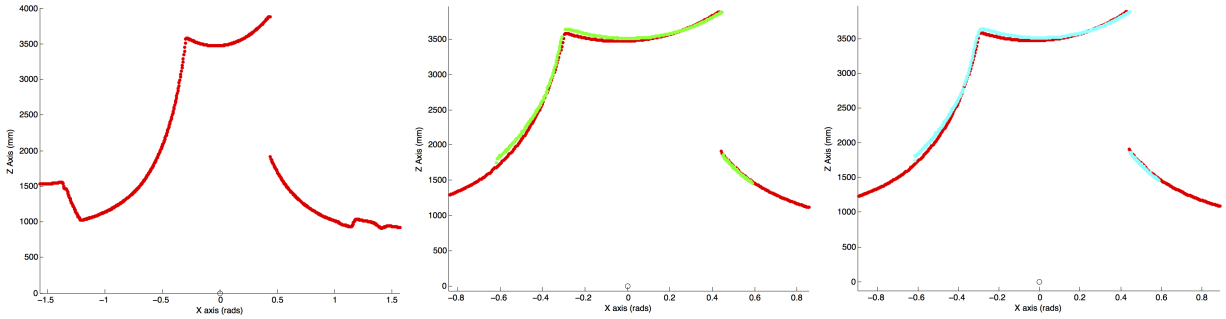
(a) (b) (c)

Figure 5.14: KOS results of scene #2: (a) Laser scan. (b) KOS for $\epsilon = 5$ over the laser scan. (c) KOS for $\epsilon = 10$ over the laser scan.



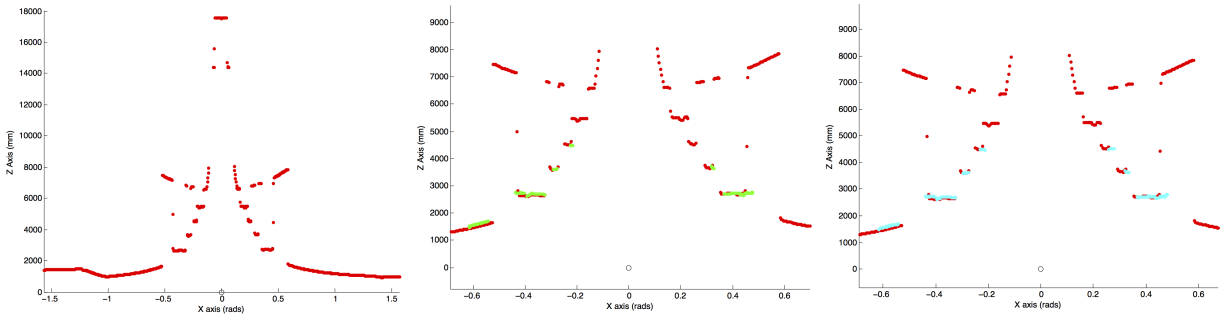
(a) (b) (c)

Figure 5.15: KOS results of scene #3: (a) Laser scan. (b) KOS for $\epsilon = 5$ over the laser scan. (c) KOS for $\epsilon = 10$ over the laser scan.



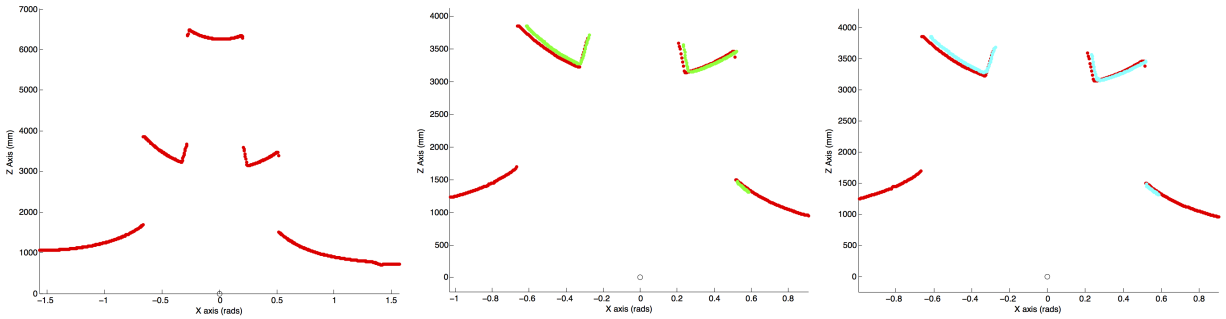
(a) (b) (c)

Figure 5.16: KOS results of scene #4: (a) Laser scan. (b) KOS for $\epsilon = 5$ over the laser scan. (c) KOS for $\epsilon = 10$ over the laser scan.



(a) (b) (c)

Figure 5.17: KOS results of scene #5: (a) Laser scan. (b) KOS for $\epsilon = 5$ over the laser scan. (c) KOS for $\epsilon = 10$ over the laser scan.



(a) (b) (c)

Figure 5.18: KOS results of scene #6: (a) Laser scan. (b) KOS for $\epsilon = 5$ over the laser scan. (c) KOS for $\epsilon = 10$ over the laser scan.

The provided data show that the KOS algorithm has very good results when comparing with the Hokuyo's laser scans. As previously referred, the results improve when the ϵ value decreases.

The multi-level kinect obstacle scan functionality was tested in scenes #7 and #8 depicted in Figure 5.19 and in Figure 5.20. For both scenes the corresponding color and depth images

are presented also the Hokuyo's laser scan and the KOS are presented, the last one for different scan heights. The results of these tests are presented in the following figures.

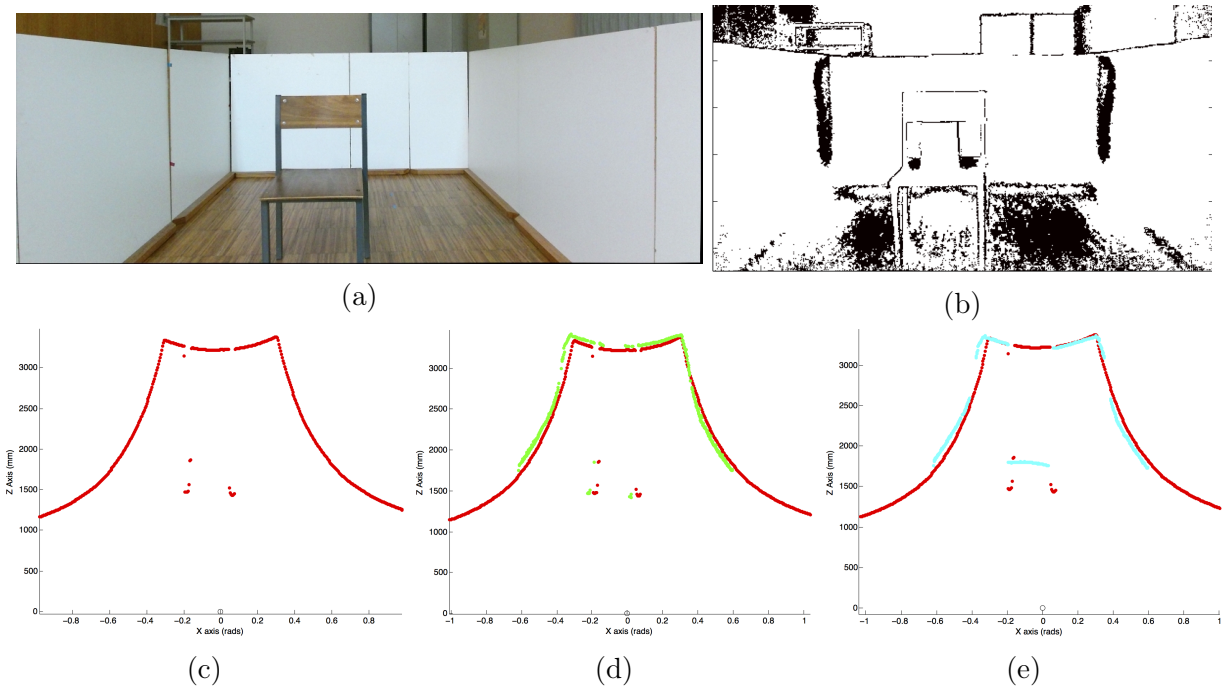


Figure 5.19: Scene #7: (a) Color image. (b) Depth image. (c) Laser scan. (d) Laser scan and KOS for same height. (e) Laser scan and KOS for an higher scan height.

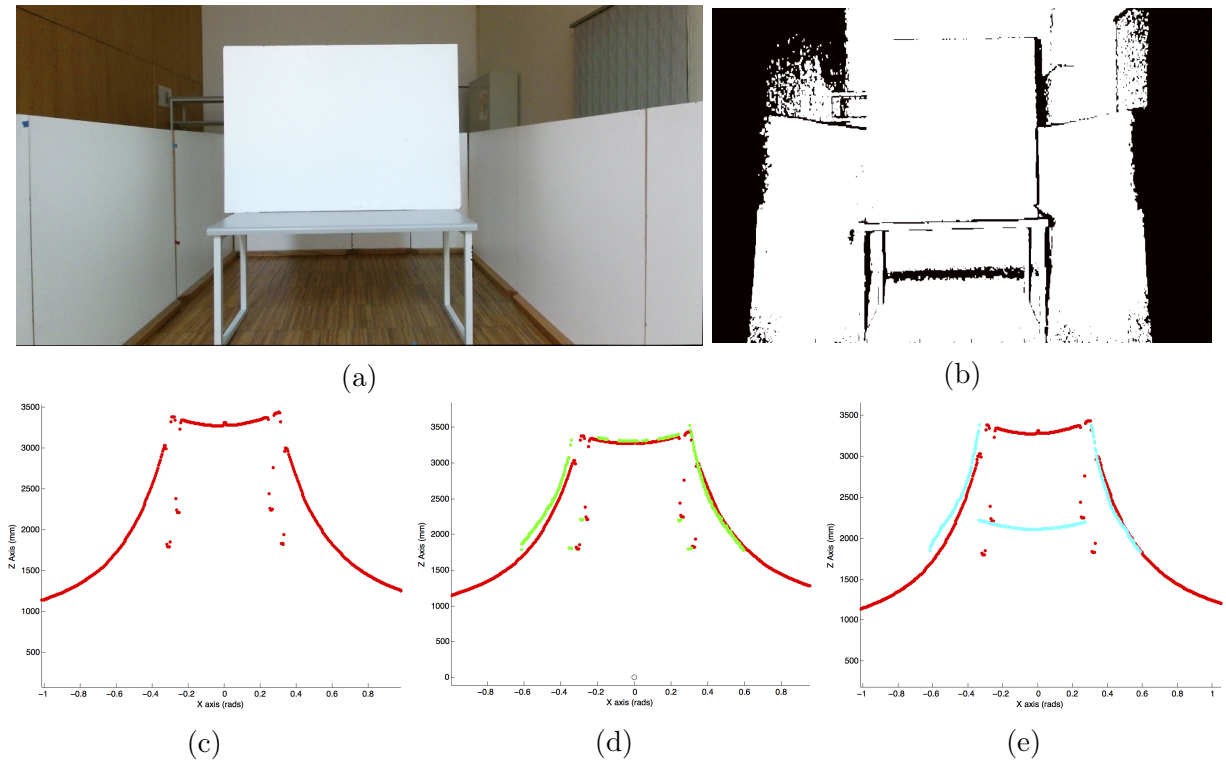


Figure 5.20: Scene #8: (a) Color image. (b) Depth image. (c) Laser scan. (d) Laser scan and KOS for same height. (e) Laser scan and KOS for an higher scan height.

In this set of tests the results shows a slight deviation between the KOS and the laser

scans in the left walls of the scenes, this was mainly due to the inclination which the white test plates have and also due to a small deviation of the Microsoft Kinect One frame regarding the Hokuyo's frame. Both tests were done with a ϵ value equals to 5.

An additional plane orientation detection test was performed. The scene #9 presented in Figure 5.21a shows a test scenario with an obstacle in the middle. First the FSPF+ algorithm is used filtering just ground and ceiling planes, then its results are provided to the KOS algorithm which generates the kinect obstacle scan of the scene shown in Figure 5.21f. Next the same procedure is done this time also filtering the front planes of the scene. The result is depicted in Figure 5.21g. The comparison of both results could be seen in Figure 5.21h.

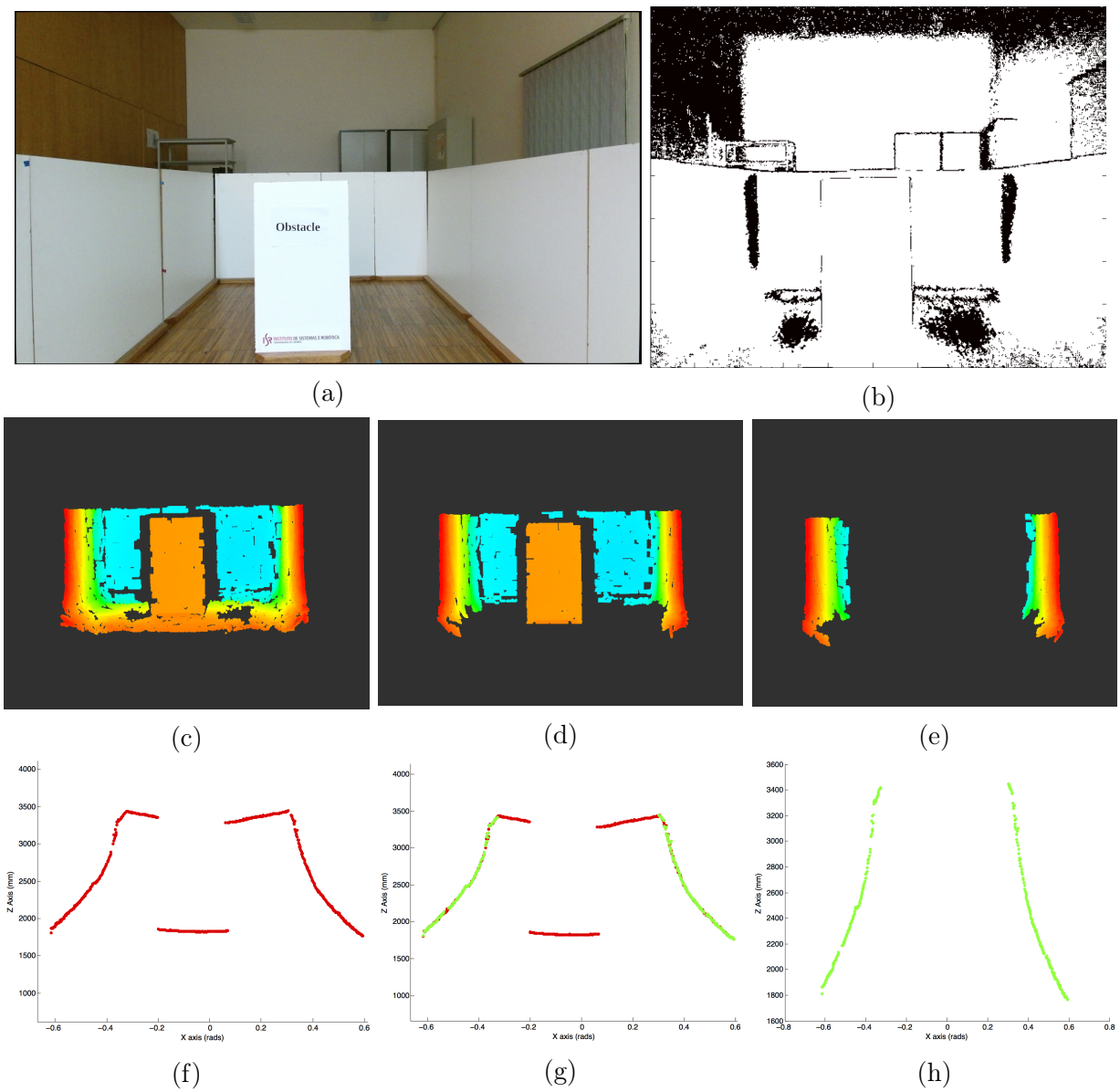


Figure 5.21: Scene #9: (a) Color image. (b) Depth image. (c) FSPF+ filtered points. (d) FSPF+ filtered points after removing ground and ceiling planes. (e) FSPF+ filtered points after removing ground, ceiling and frontal planes. (f) KOS of (c). (g) KOS of (d), the red points represent the frontal planes and the green points represents the remaining planes. (h) KOS of (e).

The obtained results conclude once again that the FSPF+ plane orientation detection has a good performance leading to good results.

5.2 Real-time tests

The FSPF+ and KOS algorithms were tested using the RtES. In these real-time tests maps are created based on the output of the developed algorithms and the hector_mapping ROS package [29]. The main aim of this set of tests the capacity of map building of the

proposed algorithm by comparing the blueprints of the ground floor of the ISR building with the map generated using the Hokuyo Laser Scan [25] and also with the map generated using the KOS. The configuration values of the FSPF+ algorithm are listed in Table 5.4. The input parameters of the FSPF+ algorithm were defined as: $minDepth$ = 500 mm, $maxDepth$ = 4500 mm and ϵ = 20 mm. The KOS's input parameters were defined as $minDepth$ = 0.5 m and $scanHeight$ = -0.3 m, corresponding to 0.5 m from the ground level and 0.3 m from the Hokuyo's height position.

Table 5.4: Configuration values of **Algorithm** 3.

Parameter	Value	Parameter	Value
nMin	40	ϵ	20
nRequired	90 % of eligible points	β	3
Ω	60	f_H	70.6
Δ	0.5	f_V	60

5.2.1 Map building

The blueprint of the ground floor of the ISR building in shown in Figure 5.22, the red box indicates the area where the tests took place.

Firstly a map using the Hokuyo's laser scan was created, this map is shown in Figure 5.23. As referred in Section 4.2.1 the laser scans were acquired while the Interbot Platform [11] was remotely controlled navigating along the selected area. Secondly two more maps were created using the KOS, these maps are depicted in Figure 5.24 and 5.25.

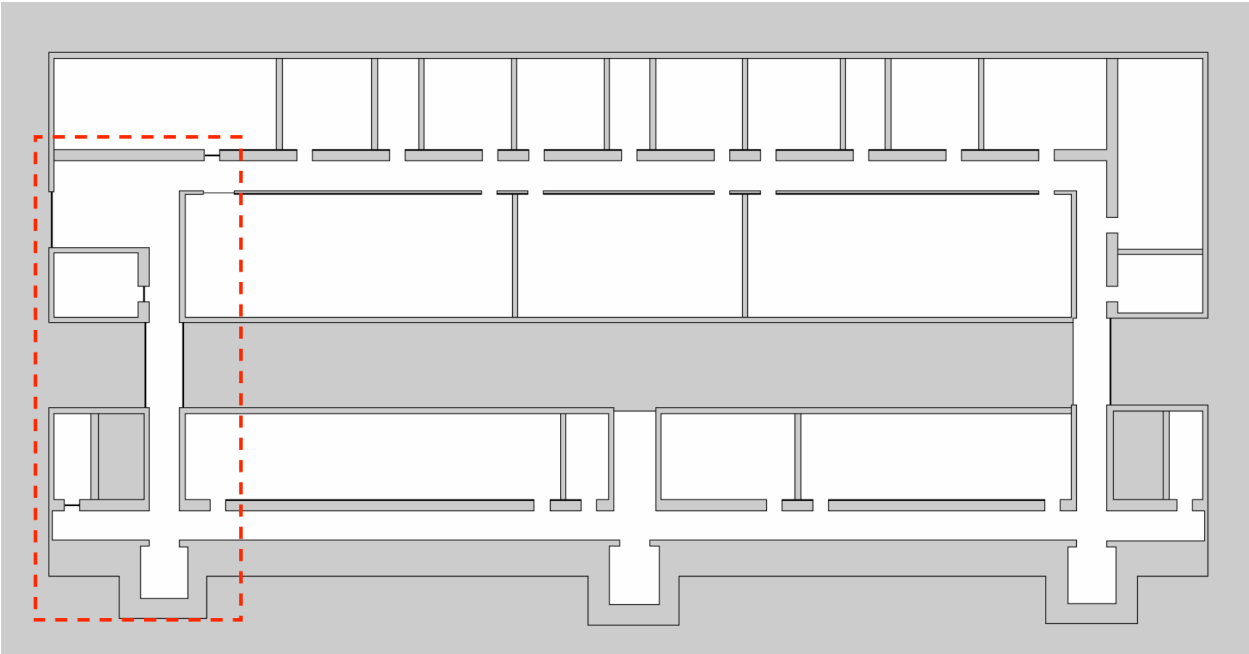


Figure 5.22: Blueprint of the ground floor of the ISR.

By comparing both KOS generated maps (Map #2 and Map #3), it is possible to say that both maps represent a good copy of the original building blueprint. It it important to refer that each one of these two maps was created by a single run from the red point to the blue point. On the other hand, Map #1 was generated after a more complex run along the area. It is also clear to conclude once the Hokuyo's range sensor has a maximum range of 30m versus a 4.5m maximum range of the Microsoft Kinect One sensor that the time needed to build the same area's map is significantly reduced, also localization issues, such as the pose loss, are decreased.

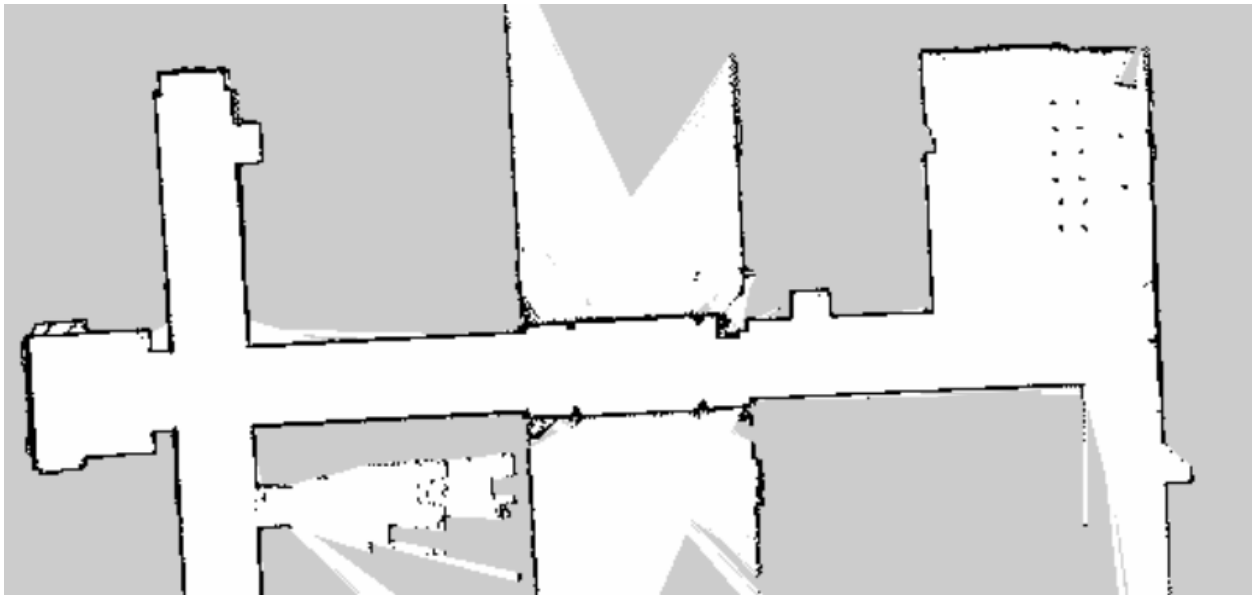


Figure 5.23: Map #1, map generated using the Hokuyo's laser scan.

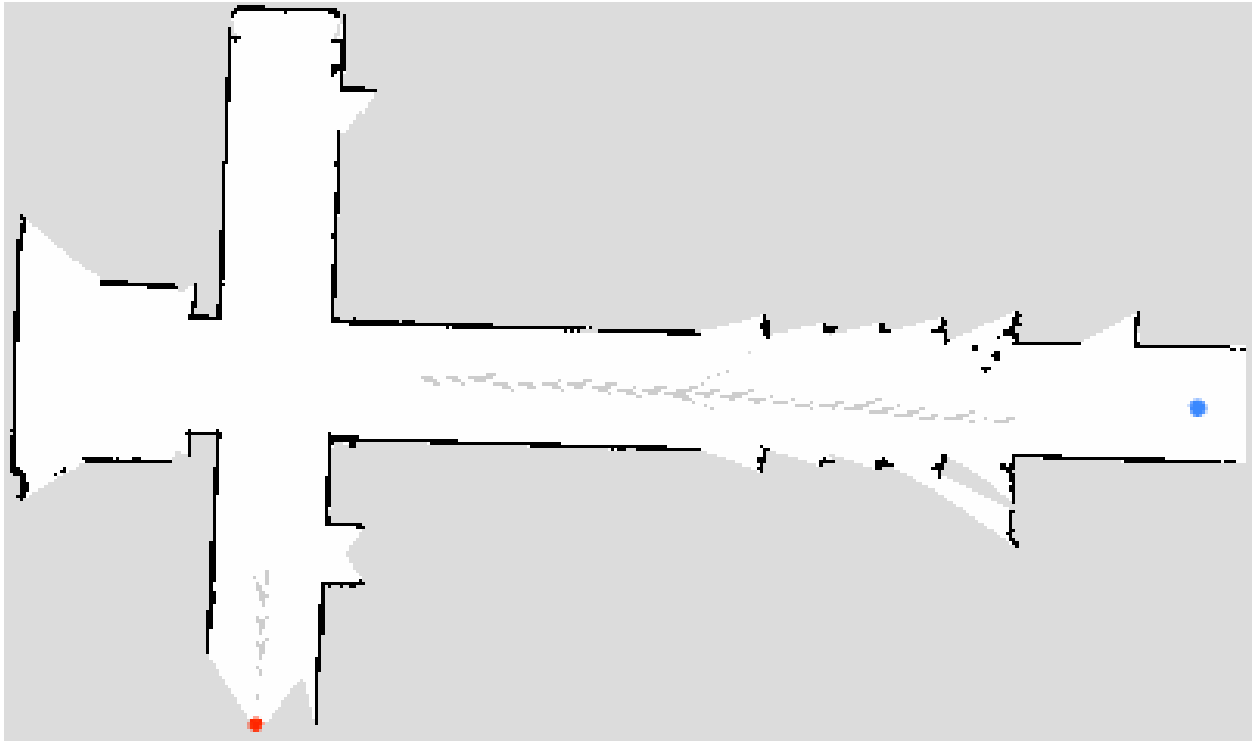


Figure 5.24: Map #2, map generated using the KOS where the red point is the starting point, and the blue point is the finish point.

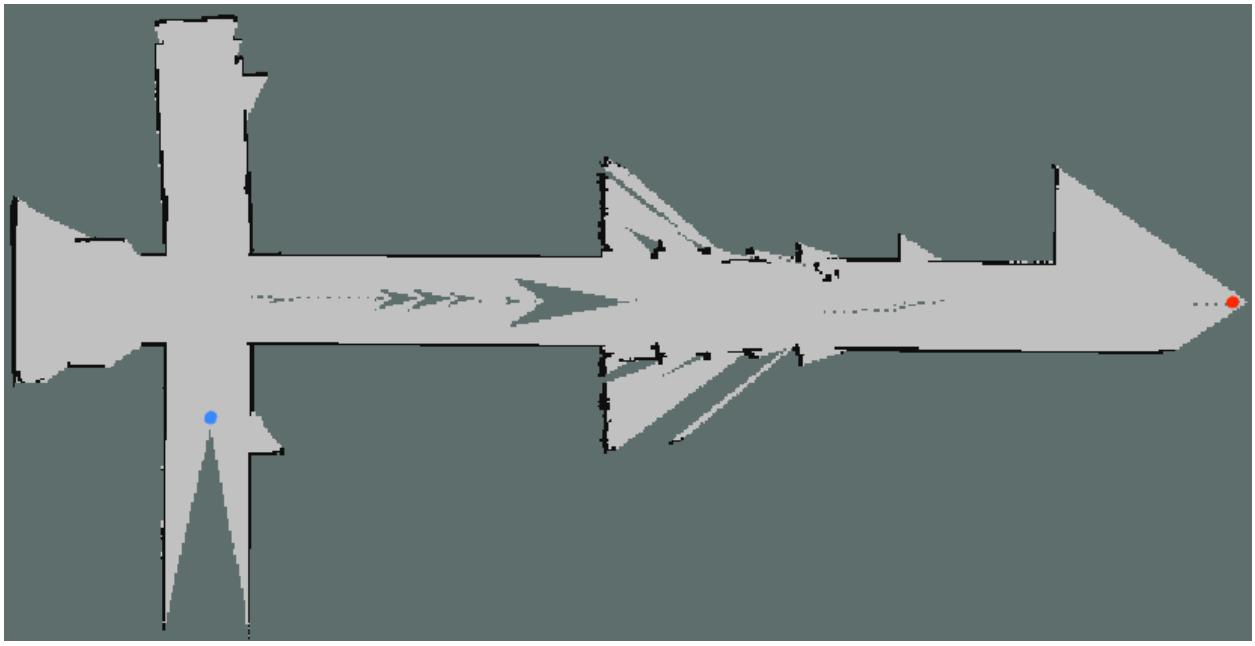


Figure 5.25: Map #3, map generated using the KOS where the red point is the starting point, and the blue point is the finish point.

Some mapped areas of Map #1 could not be mapped when used the KOS due to the fact of max range limitation combined with the characteristics of the *hector_mapping*. This algorithm does not use the robot's odometry for localization neither for navigation purposes. The map building procedure is completely guided by the laser scans, this way when the

building blueprint has long corridors with a few useful features the map generation procedure will result in erroneous mapping sections. However when none of these conditions are present the *hector_mapping* does have a good performance [11]. As it can be seen on Table 5.5 the developed system, when used for a map building application under the configuration values referred in Table 5.4 has a real-time performance of about 40Hz.

Table 5.5: Map building performance results.

Map #	FSPF+ processing time per frame (ms)	KOS processing time per frame (ms)	System performance (Hz)	Map creation elapsed time (s)
1			40	127
2	29	0.3	40	253
3	29	0.3	40	286

6 Conclusion and future work

6.1 Conclusion

This dissertation presented an improved depth image processing algorithm and a functional obstacle scan algorithm that provide useful tools for localization and navigation of indoor mobile robots. These algorithms were developed in order to be applied in the Interbot Platform, increasing its functionalities. However these algorithms might be used in mobile robots or system that use a depth image acquisition sensor. The ROS implementation of the developed work has a great importance once it has great tools not only for testing software but also to be published and shared within the scientific community in a robotic standard framework. The tests done gave empirical evidence that the whole system had a high performance contributing for a more robust Interbot Platform in terms of its navigation and localization modules.

6.2 Future work

A plane merging module would improve the functionalities of the current version of the FSPF+ algorithm, also a more robust plane's orientation detection algorithm might be developed, allowing the classification of slanting planes. Starting from the output data of the FSPF+ algorithm, namely the normals to filtered planes, a SLAM algorithm can be developed enabling the Interbot Platform to have in-house localization and mapping modules. An additional module that combines both KOS and Hokuyo's laser scan would definitely increase the robustness of the Interbot Platform mitigating the maximum range problem of the KOS and the one-level laser scan of the Hokuyo range sensor. A more complex work can be done regarding the 3D map building based on the FSPF+ plane detection. A thorough experimental set of tests to the proposed algorithms would also complement the results obtained in this dissertation.

7 Bibliography

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

[2] Georg Arbeiter, Steffen Fuchs, Joshua Hampp, and Richard Bormann. Efficient segmentation and surface classification of range images. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5502–5509. IEEE, 2014.

[3] Olga Regina Pereira Bellon, Alexandre Ibrahim Direne, and Luciano Silva. Edge detection to guide range image segmentation by clustering techniques. In *Image Processing, 1999. ICIP 99. Proceedings. 1999 International Conference on*, volume 2, pages 725–729. IEEE, 1999.

[4] Joydeep Biswas. Vector map-based, non-markov localization for long-term deployment of autonomous mobile robots. 2014.

[5] Joydeep Biswas and Manuela Veloso. Fast sampling plane filtering, polygon construction and merging from depth images.

[6] Joydeep Biswas and Manuela Veloso. Fast sampling plane filtering, polygon construction and merging from depth images.

[7] Joydeep Biswas and Manuela M. Veloso. Localization and navigation of the cobots over long-term deployments. *The International Journal of Robotics Research*, 32(14):1679–1694, 2013.

[8] Dorit Borrmann, Jan Elseberg, Kai Lingemann, and Andreas Nüchter. The 3d hough transform for plane detection in point clouds: A review and a new accumulator design. *3D Research*, 2(2):1–13, 2011.

[9] DH Brandwood. A complex gradient operator and its application in adaptive array theory. In *IEE Proceedings F: Communications Radar and Signal Processing*, volume 130, pages 11–16, 1983.

[10] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[11] André Conceição. Interbot mobile robot: Navigation modules. 2016.

[12] Konstantinos G Derpanis. Overview of the ransac algorithm. *Image Rochester NY*, 4(1):2–3, 2010.

[13] Jean-Emmanuel Deschaud and François Goulette. A fast and accurate plane detection algorithm for large noisy point clouds using filtered normals and voxel growing. In *Proceedings of 3D Processing, Visualization and Transmission Conference (3DPVT2010)*, 2010.

[14] Ting-Jun Fan, Gerard Medioni, and Ramakant Nevatia. Segmented descriptions of 3-d surfaces. *IEEE Journal on Robotics and Automation*, 3(6):527–538, 1987.

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

[16] David F Fouhey, Daniel Scharstein, and Amy J Briggs. Multiple plane detection in image pairs using j-linkage. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 336–339. IEEE, 2010.

[17] Open Source Robotics Foundation. Gazebo. `http://gazebosim.org`. [Online; accessed 12-August-2016].

[18] Orazio Gallo, Roberto Manduchi, and Abbas Rafii. Cc-ransac: Fitting planes in the presence of multiple surfaces in range data. *Pattern Recognition Letters*, 32(3):403–410, 2011.

[19] Rafael C Gonzalez and Richard E Woods. Digital image processing. *Nueva Jersey*, 2008.

[20] Paulo FU Gotardo, Olga Regina Pereira Bellon, Kim L Boyer, and Luciano Silva. Range image segmentation into planar and quadric surfaces using an improved robust estimator

and genetic algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(6):2303–2316, 2004.

[21] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE transactions on Robotics*, 23(1):34–46, 2007.

[22] Dirk Hähnel, Wolfram Burgard, and Sebastian Thrun. Learning compact 3d models of indoor and outdoor environments with a mobile robot. *Robotics and Autonomous Systems*, 44(1):15–27, 2003.

[23] Robert M Haralick and Linda G Shapiro. Image segmentation techniques. *Computer vision, graphics, and image processing*, 29(1):100–132, 1985.

[24] Michiel Hazewinkel. Witt vectors. part 1. *Handbook of algebra*, 6:319–472, 2009.

[25] Hokuyo. Hokuyo UTM30-LX. `https://www.hokuyo-aut.jp/02sensor/07scanner/utm_30lx.html`, 2012. [Online; accessed 12-August-2016].

[26] Runzhen Huang and Kwan-Liu Ma. Rgvis: Region growing based techniques for volume visualization. In *Computer Graphics and Applications, 2003. Proceedings. 11th Pacific Conference on*, pages 355–363. IEEE, 2003.

[27] Xiaoyi Jiang. An adaptive contour closure algorithm and its experimental evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1252–1265, 2000.

[28] Ravi Kaushik, Jizhong Xiao, Samleo L Joseph, and William Morris. Fast planar clustering and polygon extraction from noisy range images acquired in indoor environments. In *Mechatronics and Automation (ICMA), 2010 International Conference on*, pages 483–488. IEEE, 2010.

[29] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011.

[30] Peter Kovesi. A general purpose implementation of the RANSAC algorithm in MatLab. `http://www.peterkovesi.com/matlabfns/Robust/ransac.m`, 2003. [Online; accessed 04-September-2016].

[31] Zheng Lin, Jesse Jin, and Hugues Talbot. Unseeded region growing for 3d image segmentation. In *Selected papers from the Pan-Sydney workshop on Visualisation-Volume 2*, pages 31–37. Australian Computer Society, Inc., 2000.

[32] Yu Liu and Youlun Xiong. Automatic segmentation of unorganized noisy point clouds based on the gaussian map. *Computer-Aided Design*, 40(5):576–594, 2008.

[33] MathWorks. MatLab. `http://www.mathworks.com/products/matlab/`. [Online; accessed 12-August-2016].

[34] Microsoft. Microsoft Kinect for Xbox ONE. `https://developer.microsoft.com/en-us/windows/kinect`, 2014. [Online; accessed 11-August-2016].

[35] Abdul Nurunnabi, David Belton, and Geoff West. Robust segmentation for multiple planar surface extraction in laser scanning 3d point cloud data. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 1367–1370. IEEE, 2012.

[36] Point Cloud Library (PCL). An implementation of the RANSAC (RAndom SAmple Consensus) algorithm in C++. `http://docs.pointclouds.org/trunk/sample__consensus_2include_2pcl_2sample__consensus_2ransac_8h_source.html`, 2009. [Online; accessed 04-September-2016].

[37] Jann Poppinga, Narunas Vaskevicius, Andreas Birk, and Kaustubh Pathak. Fast plane detection and polygonalization in noisy 3d range images. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3378–3383. IEEE, 2008.

[38] Xiangfei Qian and Cang Ye. Ncc-ransac: a fast plane extraction method for 3-d range data segmentation. *IEEE transactions on cybernetics*, 44(12):2771–2783, 2014.

[39] ROS. Grid Occupancy Message. `http://docs.ros.org/api/nav_msgs/html/msg/OccupancyGrid.html`. [Online; accessed 03-September-2016].

[40] ROS. Hokuyo Node. `http://wiki.ros.org/hokuyo_node`. [Online; accessed 12-August-2016].

[41] ROS. Image Message. `http://docs.ros.org/api/sensor_msgs/html/msg/Image.html`. [Online; accessed 12-August-2016].

[42] ROS. LaserScan Message. `http://docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html`. [Online; accessed 12-August-2016].

[43] ROS. PointCloud Message. http://docs.ros.org/api/sensor_msgs/html/msg/PointCloud.html. [Online; accessed 12-August-2016].

[44] ROS. Pose Stamped Message. http://docs.ros.org/api/geometry_msgs/html/msg/PoseStamped.html. [Online; accessed 03-September-2016].

[45] Linda Shapiro and George C Stockman. Computer vision. 2001. *ed: Prentice Hall*, 2001.

[46] Manuela Veloso, Joydeep Biswas, Brian Coltin, Stephanie Rosenthal, Tom Kollar, Cetin Mericli, Mehdi Samadi, Susana Brandao, and Rodrigo Ventura. CoBot Robots. http://www.cs.cmu.edu/~coral/projects/cobot/. [Online; accessed 05-September-2016].

[47] Manuela Veloso, Joydeep Biswas, Brian Coltin, Stephanie Rosenthal, Tom Kollar, Cetin Mericli, Mehdi Samadi, Susana Brandao, and Rodrigo Ventura. Cobots: Collaborative robots servicing multi-floor buildings. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5446–5447. IEEE, 2012.

[48] S Vishvjit and A Nalwa. A guided tour of computer vision, 1993.

[49] Thiemo Wiedemeyer. IAI Kinect2. https://github.com/code-iai/iai_kinect2, 2014 – 2015. [Online; accessed 12-August-2016].

[50] Junhao Xiao, Benjamin Adler, Jianwei Zhang, and Houxiang Zhang. Planar segment based three-dimensional point cloud registration in outdoor environments. *Journal of Field Robotics*, 30(4):552–582, 2013.

[51] Michael Ying Yang and Wolfgang Förstner. Plane detection in point cloud data. In *Proceedings of the 2nd int conf on machine control guidance, Bonn*, volume 1, pages 95–104, 2010.

[52] EA Zanaty, MT El-Melegy, MR Girgis, and Walaa M Abd-Elhafiez. Hybrid image segmentation method based on seed region growing and edge detection.