José Manuel Ribeiro Rosa

Indoor Perception and Navigation Strategies for Assistive Robotics

Coimbra 2017

UNIVERSIDADE DE COIMBRA

• U C •

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Indoor Perception and Navigation Strategies for Assistive Robotics.

José Manuel Ribeiro Rosa

Coimbra, October 2017

Indoor Perception and Navigation Strategies for Assistive Robotics.

Supervisor:

Prof. Dr. Urbano José Carreira Nunes

Jury:

Prof. Dr. Gabriel Pereira Pires

Prof. Dr. João Pedro de Almeida Barreto

Prof. Dr. Urbano José Carreira Nunes

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

Coimbra, October 2017

Agradecimentos

Quero começar por agradecer ao orientador desta dissertação de Mestrado, Professor Doutor Urbano Nunes pela orientação e disponibilidade providenciada em todos os momentos da sua elaboração. Quero também fazer um agradecimento ao Instituto de Sistemas e Robótica, por todo o material e meios disponibilizados, bem como aos colegas do Laboratório de Sistemas de Tempo real pelos momentos de descontração e por toda a ajuda prestada, em especial ao Luis Garrote por todos os conselhos, ajuda e preocupação ao longo deste ano. Um agradecimento a todos os meus amigos, aos de sempre e aos que este percurso académico me foi dando, em especial ao Daniel, Maiel e José por toda a força, ajuda e por todos os momentos bons e menos bons que partilhamos ao longo destes anos que garantidamente não irei esquecer. Por fim um especial agradecimento aos meus pais José e Maria por todo o amor e ajuda que sempre me deram, por estarem sempre presentes e preocupados e pelo esforço dedicado à formação dos filhos, às minhas irmãs, Eduarda e Sofia, e à Susana por todo o amor e apoio dados. A eles devo tudo o que sou hoje e sem eles nada disto teria sido possível.

Um grande Obrigado!

Resumo

O desenvolvimento de tecnologias de percepção é um dos principais focos de investigação em robótica. Com o aumento da aplicação da robótica em ambiente humano, existe uma maior preocupação em garantir a segurança de todos os agentes que podem interagir com o robô.

Considerando o caso em que um robô é um elemento de suporte na reabilitação de um paciente, mais especificamente um andarilho, a prevenção de acidentes surge como a maior dificuldade a superar, mas um requesito absoluto. Para superar este problema, o andarilho deve garantir que segue todas as instruções de acordo com as regras de segurança e tome todas as decisões com base no conhecimento do ambiente envolvente.

Esta dissertação teve como objectivo propor o desenvolvimento de um sistema de navegação seguro aplicado a um andarilho robótico, desenvolvido no Instituto de Sistemas e Robótica (ISR), chamado "ISR-AIWALKER". Para alcançar o nosso objectivo, dois métodos distintos foram desenvolvidos durante esta dissertação: percepção do ambiente local e navegação robô-assistida. A percepção do ambiente local é a responsável por identificar todos os obstáculos, enquanto a navegação robô-assistida pelo robô garante que a plataforma robótica não colide com eles, e reencaminha o utilizador para um caminho seguro. Além dos sensores já incorporados no ISR-AIWALKER, foram introduzidos dois novos sensores (Microsoft Kinect One e Leddar Tech Leddar IS16) para recolher dados sobre o ambiente envolvente da plataforma robótica a fim de garantir a segurança do paciente e de outros dentro do mesmo espaço.

Todos os algoritmos propostos foram desenvolvidos em ambiente ROS (*Robot Operating System*) e testados em locais onde podemos encontrar obstáculos que tornam difícil ao utilizador se deslocar diariamente (calhas, escadas, rampas).

Palavras-chave: Robô autónomo, Navegação, Localização, ROS, Plataforma Robótica Móvel, Microsoft Kinect One, Leddar, Detecção de Planos, Mapeamento.

Abstract

The development of perception technologies is one of the main focuses of research in robotics. With the increasing application of robotics in human populated environments, there is a heightened concern to ensure the safety of all agents who may interact with the robot.

Considering the case where a robot is a support element in the rehabilitation of a patient, more specifically with a walker, the accidents prevention emerges as the strongest difficulty to overcome, but an absolute requirement. To overcome this problem, the walker must ensure that follows all instructions accordingly with safety rules and makes all decisions based on the knowledge of the surrounding environment.

This dissertation aimed to propose the development of a safe navigation system applied in a robotic walker, developed at the Institute of Systems and Robotics (ISR) called "ISR-AIWALKER". To achieve our goal, two distinct software architectures were developed during this dissertation: local environment perception and robot-assisted navigation. The local perception module is the one responsible for detecting the environment obstacles while the robot assisted navigation ensures that the robotic platform will not collide with them, rerouting the user to a safe path.

In addition to the sensors already incorporated in the ISR-AIWALKER, two new sensors (Microsoft Kinect One and LeddarTech Leddar IS16) were introduced to collect information about the surrounding environment of the robotic platform in order to guarantee the safety of the patient and others within the same space.

All of the proposed algorithms were developed in a ROS environment (Robotic Operating System) and tested in places where we can find obstacles that make it difficult for the user to move on a daily basis (gutters, stairs, ramps).

Keyword: Autonomous robot, Navigation, Localization, ROS, Mobile Robotic Platform, Microsoft Kinect One, Leddar, Plane Detection, Mapping.

“Scientists have become the bearers of the torch of discovery in our quest for knowledge. ”

— Stephen Hawking

Contents

List of Acronyms

1D	One Dimensional
2D	Two Dimensional
3D	Three Dimensional
FCT	Fundação para a Ciência e Tecnologia
I/O	Input and Output
ISR	Institute of Systems and Robotics
PCL	Point Cloud Library
RANSAC	RANdom SAmple Consensus
RGB	Red, Green and Blue
ROS	Robot Operating System
DWA	Dynamic Window Approach
HMI	Human Machine Interface

List of Figures

List of Tables

1 Introduction

This chapter provides an overview of this dissertation. We provide the motivation and context of the developed work as well as the goals and key contributions, summarizing each chapter in order to contextualize the reader.

1.1 Motivation and context

In the last years, robotics has been one of the engineering areas with intensive research and remarkable evolution, for example as regards autonomous navigation of mobile robots. However there are still some limitations, such as the ability to properly model and predict possible dangers for users. Despite the recent advances in the area, the development of robust models for representation of the surrounding environment of the robot is still an open problem and a topic with great challenges, mostly on the "interpretation" of uncertainty in sensor measurements and their correct representation. Common environment model representations follow an approach based on occupancy grid maps, which makes it possible to determine, based on the probabilistic models of the used sensors, specific information about the existence of free and occupied space in the surrounding environment.

Inside the broad scope of Mobile Robotics, the Assistive robotics research has gained in recent years a special focus since, in this area, research topics are addressed aiming to improve the quality of life for the elderly and for people with debilitating diseases.

When we talk about assistive robots one of the main concerns is the safety of the user, since it must be ensured that the user does not crash or fall into non-trivial obstacles, for example that a semi-autonomous wheelchair does not lead the user to fall from stairs or to collide with obstacles, like tables or walls. To ensure safety the assistive robots must be equipped with robust algorithms and diverse sensors such as sonars and cameras that allow the system to make decisions when it comes to planning the best path to follow.

This dissertation proposes a new approach to the problem of mapping collision-able non-trivial obstacles from a 3D point cloud into a 2D environment representation throughout an intermediary 2.5D representation, that may be useful for motion planning either by direct computation or throughout cost maps.

The proposed method can enhance safety in mobile robot applications since dangerous obstacles that are not normally detected but can originate disastrous consequences for the user will be mapped (e.g., stairs, gutters or floor outlets) and avoided in real-time using motion planning.

1.2 Goals

The purpose of this dissertation was to design modules for perception and navigation of the ISR-AIWALKER platform to ensure safe navigation in indoor environments. The defined objectives encompass the development and implementation of several methods/algorithms, computationally efficient, and respective validation in real scenarios:

- Algorithms for detection of non-trivial obstacles such as stairs and gutters ;
- An algorithm for the conversion of 3D point clouds in a model/representation of the environment suitable for motion and path planning;
- An algorithm for local navigation using the generated occupancy grid maps.

1.3 Implementations and key contributions

The design of modules for $2.5D$ to $2D$ occupancy grid map conversion using 3D point clouds, a navigation algorithm capable of helping the user to make decisions based on the occupancy grid map created and the user's own intent, the development and validation in ROS environment as well as the publication in IEEE conference ICARSC were the key contributions of this dissertation. Some ROS modules were used and others were developed or modified in order to develop a fully functional assistive robotic platform system. Figure 1.1 illustrates the main developed modules described in this dissertation.

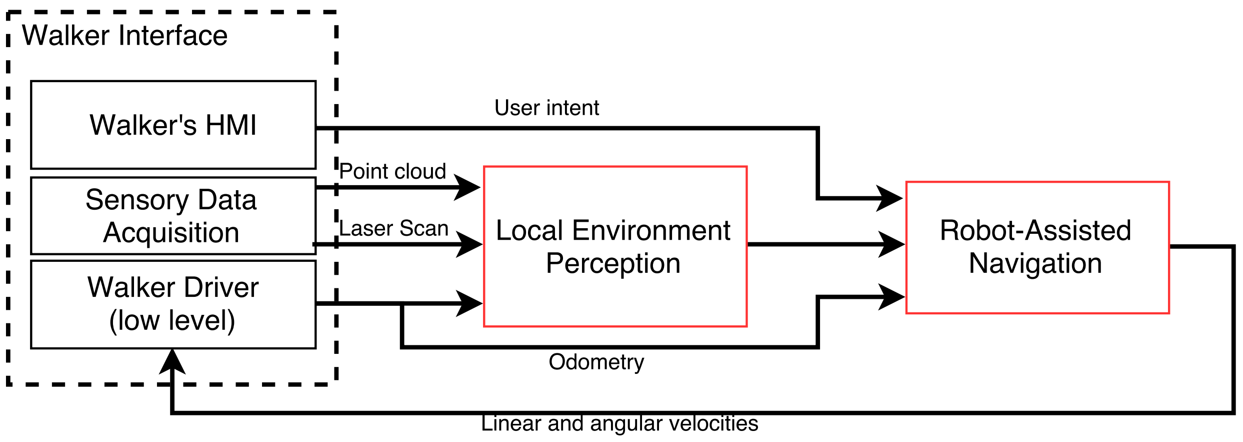

Walker Interface
Walker's HMI
Sensory Data Acquisition
Walker Driver (low level)
User intent
Point cloud
Laser Scan
Local Environment Perception
Robot-Assisted Navigation
Odometry
Linear and angular velocities

Figure 1.1: Main modules and key contributions developed.

The outline of this dissertation and content of each chapter is the following:

State of the Art (Chapter 2)

- A description of the perception sensors used in this dissertation.
- Brief description of the existing environment representations.
- A description of the different methods of segmentation.
- Brief description of the existing navigation methods for collision-avoidance.
- A system overview of the ISR-AIWALKER platform containing a brief description of the main modules that compose the software architecture.

Perception (Chapter 3)

- Description of the proposed method for environment perception.

Navigation (Chapter 4)

- Description of the developed local motion planning algorithm used for the ISR-AIWALKER navigation.

Validation (Chapter 5)

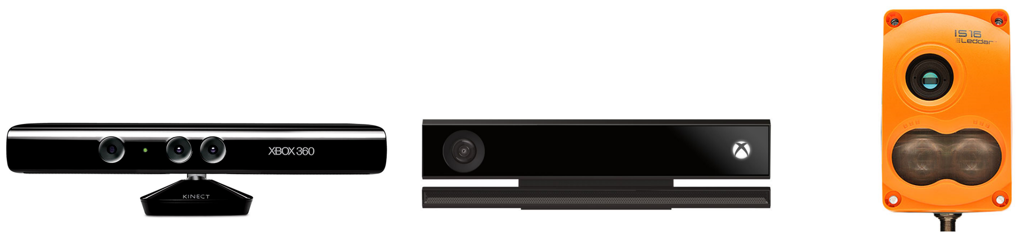
- Experiments and validation of the proposed modules as well as the validation of the complete system (perception and navigation).

2 State of the art

In this chapter we present the different concepts used during this work. This will serve as theoretical background to better contextualize the approaches described in the rest of this dissertation.

2.1 Sensors for environment perception

The ability of a robot to recognize and evaluate its surrounding environment is not possible without sensors. The correct selection of sensors is an important topic because these sensors must be chosen accordingly with the robot's workspace. There are two types of sensors that can be used: passive and active sensors.



(a) (b) (c)

Figure 2.1: Used sensors to represent the surrounding environment of the ISR-AIWALKER.

The selected sensors for the implementation of the perception algorithms were the active sensors Microsoft Kinect 360, in Fig.2.1a, Microsoft Kinect One, in Fig.2.1b and LeddarTech's Leddar IS16, in Fig.2.1c. The Kinect outputs a 3D point cloud used in section III to detect obstacles with different heights and shapes and we used the Leddar IS16 sensor to ensure safety on close distances and in environments where the Kinect cannot detect the obstacles, for example, on dark environments. The characteristics of the used sensors are shown in Table2.1.

Sensor	Leddar	Kinect 360	Kinect One
Sampling rate	$50Hz$	$30Hz$	$30Hz$
Field of vision	45°	43°/57°	60°/70°
Range	$0m - 50m$	$0.7m - 6m$	$0.5m - 4.5m$

Table 2.1: Specifications of the used sensors.

2.1.1 Microsoft Kinect 360 and Kinect One

The Microsoft Kinect 360 is a low cost camera that has the capability to provide RGB-D images and depth information simultaneously, which make it suitable for robotic applications in indoor environments. The Kinect 360, shown in Fig.2.1a, includes an RGB camera, an infrared (IR) emitter, an IR depth sensor, a multi-array microphone and a motorized tilt system. The IR emitter emits infrared light beams in a pattern of speckles that are reflected back to the sensor and read by the IR camera. This reflected pattern is correlated against a reference pattern stored in the memory of the Kinect, obtained by capturing a plane at a known distance from the sensor. For each speckle projected on an object whose distance is different than that of the reference plane, its position in the IR image will be shifted, originating a disparity image. From the disparity image, it is possible to compute the distance to the sensor, and therefore the 3D coordinates for each pixel, applying a triangulation method.

Feature	Kinect 360	Kinect One
RGB image resolution	640 x 480 @30fps	1920 x 1080 @30fps
Depth image resolution	640 x 480 @30fps	512 x 424 @30fps
Depth operation range	0.7m-6m	0.5m-4.5m
Viewing angle	43°/57°FOV	60°/70°FOV
Vertical tilt range	27°	-

Table 2.2: Microsoft Kinect 360 and Kinect One specifications.

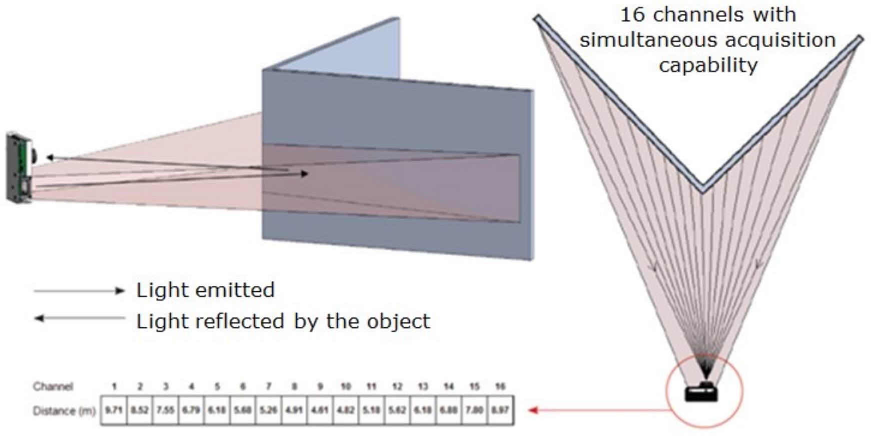
The Kinect One sensor, in Fig. 2.1b, features a number of changes in comparison to the first-generation presented in the previous section. The Kinect One uses infrared to read its environment but it has greater higher accuracy compared to the original Kinect, processing 2 gigabits of data per second. The device features a 512x424 pixel time-of-flight (TOF) camera, has an increased field of view, a 1080p resolution video camera that can be used for video recording, better depth map pixel resolution and better robustness against sunlight. This main difference makes the Kinect One more precise than the Kinect 360 but, on the other side, it uses much more computational space when we are recording datasets due to its higher resolution. In order to mitigate this problem we only used the standard definition, SD, PointCloud provided by the Kinect One which has an average data rate of 10 Hz.

2.1.2 LeddarTech's Leddar IS16

The LeddarTech's Leddar IS16, in Fig.2.1c, is a light detection and ranging (LiDAR) sensor that uses led technology, instead of the traditional laser technology, to measure distances between itself and obstacles in the sensor's field of view. The software package used to obtain the distances (LeddarTech ROS package) calculates the time taken by a pulse of light to travel to an object and back to the sensor lent.

The output of the Leddar IS16 is a 16-channel array of segments that returns the distance from every single channel to the obstacle in front of the sensor. With this information we were able to see and track the profile of one or multiple objects that the sensor is detecting.

The Fig.2.2 illustrates the illumination area and detection segments of the sensor.


16 channels with simultaneous acquisition capability
Light emitted
Light reflected by the object

Channel	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Distance (m)	9.71	8.52	7.55	6.79	6.18	5.68	5.26	4.91	4.61	4.82	5.18	5.62	6.18	6.88	7.80	8.97

Figure 2.2: LeddarTech's Leddar IS16 illumination area from the sensor datasheet

In the following table is possible to see the detailed characteristics of the sensor.

Feature	Value
Range	0 - 50m
Field of view	45°
Sample Rate	up to 50Hz
Number of segments	16 independent segments

Table 2.3: LeddarTech's Leddar IS16 specifications.

2.2 Environment representations

The representation of the environment has been the focus of many research publications, with many representations being proposed in the last 30 years. This representation consists in creating a representation of the surrounding environment based on sensors information so that the robot can navigate knowing the hazards that surround it. We can categorize these representations in 3 main classes: direct [23], topological [22] and grid-based [10].

2.2.1 Direct Representation

In the direct representation approach [23], the raw sensor measurements (e.g., provided by LiDAR, stereo-cameras) are aggregated in a way that can be described as a point cloud [6], without extracting any characteristics of the environment. This method does not model free or unknown space, which makes the navigation of a mobile robot more difficult and can lead to several collisions with unseen obstacles.

2.2.2 Topological Representation

The topological representation, represents locations or nodes in graphs where each node can be attributed to landmarks, and metric or appearance-based places [30]. This type of representation has the advantage of being compact and scalable (resolution corresponds directly to the complexity of the environment), which allows fast planning (often in a global scope) and integration in multiple hierarchical planning strategies. On the other hand, it may be difficult to construct and maintain in large scale environments.

2.2.3 Grid Based representations

In a grid-based representation, the environment is subdivided into a set of smaller units that form a grid. The unit size, update and structure are normally method and implementation dependent. Based on the unit structure, the grid-based representation can be further extended into 2D cells, 2.5D cells (or voxels) and 3D voxels. Grid approaches are easy to build, represent, maintain and facilitates computation of shortest paths. However, grid-based representations present problems such as: is an inefficient representation for path planning; require large memory space for modeling large environments; and make it difficult to interface with most classical planning algorithms. In the literature, the 2D grid-map is the most widespread representation used, providing at its core, a probabilistic framework [32], with fast and constant-time access while being only applicable in planar environments (i.e., can not represent directly the concept of vertical obstacles). In this type of representation each cell contains the probability of occupancy, where values near zero correspond to a free cell, values near one an occupied cell and value in between may correspond to an unexplored region of the environment. The value of probability depends on the successive readings from sensors, increasing or decreasing according to sensor's observation models.

Other representations such as 2.5D grid-maps, also known as height or elevation maps, provide a framework to represent elevations and irregular terrain instead of occupancy, with properties similar to 2D maps, but with additional information of height on each cell, and a computationally lighter model than 3D grid-maps. However this type of representation cannot fully represent vertical overlapping [26]. Solutions to the overlapping problem have been proposed in the form of multi-layer maps [33]. More recently, 3D grid-maps, and in particular solutions such as octomaps [17], provide a reliable 3D representation at the expense of variable access time, increased computational complexity and increased planning complexity for ground robots.

In spite of the constant evolution, 2D grid-maps are still a viable choice for many cases in indoor and outdoor scenarios composed by local flat ground surfaces, but fail to incorporate vertical elements that are on a given robot pathway, even if they are trivially spotted by humans. Methods for the conversion of 3D point clouds to 2D grid maps are described, for example, in [1, 28].

2D Occupancy Grid Map

2D occupancy grid maps are probabilistic maps, meaning that each cell contains a high certainty probability of occupancy value between zero and one. This value of probability depends on the successive readings of the sensors, increasing or decreasing according to the surrounding environment of the robot, where this value can be changed frequently. This method was first introduced by Alberto Elfes and Hans Moravec in [24], and some development and testing were presented later in [9]. Years later, Sebastian Thrun, [32],[31], brought contributions to this method, introducing an algorithm where the occupancy of each grid cell is dependent of its neighbors in both two dimensions. This improvement reduced the error from the sensor readings, resulting in more accurate maps than those generated using traditional techniques.

2.5D Occupancy Map

The 2.5D map was developed in order to map non-flat surfaces, providing the height of the obstacles. This two-dimensional occupancy grid representation, studied extensively in the past two decades, unlike the traditional 2D maps provides additional information of the height of each cell of the map, creating a map of occupation more detailed than the traditional 2D maps and a lighter map than the 3D representation, being a middle term approach that dominate current applications. One of the first applications of this method, presented in [16], was developed for a planetary mission to map areas of the surface of Mars. Another implementation of this method was presented in [26], where polar grids are used since the rotary laser range finder returns distance measurements in a polar or “spherical-like” coordinates. The method was also discussed and improved for the decrease in density of points with the increase of distance by [15] for posterior mapping of rough terrains in elevation maps. Some representations provided by these applications are shown in 2.3.

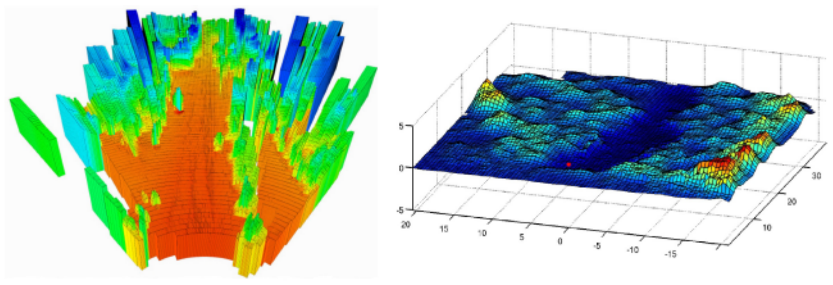


Figure 2.3: Left: 2.5D in polar grid of urban environment from [26]. Right: Elevation map of rough terrain with space-carving kernels from [15]

2.3 Segmentation

Segmentation is the process of partitioning a set of data into multiple segments and a useful tool in many areas including mobile robotics, industry, health care and astronomy. With this process, the representation of objects becomes easier to analyze because a cluster is assigned to every point accordingly with its characteristics (texture, color and so on) and these characteristics can be further evidenced in a more appealing manner, for example different colors for different depths of an image. These types of algorithms are usually based on one of two basic principles: discontinuity and similarity. The first approach is based on edge detection and the second are based on thresholding and region growing. When we deal with sensor information based on pointclouds to perform safe navigation through the environment, there is a need to analyze the output data of this type of sensor using segmentation to make sure that the robot does not discard obstacles that it can overcome (small bump or hill).

2.3.1 Region growing segmentation

The purpose of the region growing segmentation is to merge the points that are close enough in terms of the smoothness constraint (such as texture, color, shape). This means that the neighborhood of a pixel is examined and added to a region class if no edges are detected. The process is then iterated for each boundary pixel in the region.

The first step in region growing is to sort the points by their curvature value, because the region's growth begins from the point that has the minimum curvature value. After this sorting we are in condition to start the growth of the region and the point is added to the set called seeds. The region growing segmentation analyzes the neighborhood of a pixel and this process is done by measuring the angle between its normal and the normal of the current seed point, adding the current point to the current region if the angle is smaller than the threshold value. This process is then repeated for the curvature value and if this curvature is less than the threshold value the pixel is added to the seed and the current seed is removed from the seeds set. When the seeds set is empty it means that the region has finished its growth and the process is then repeated from the beginning.

The pseudo code for this process is shown in **Algorithm 1** ,from [27], considering the point cloud, P, the point normals, N, the points curvatures, c, the neighbor finding function, Ω, the curvature threshold, c_{th} and the angle threshold, Φ_{th}.

Algorithm 1: Region Growing Segmentation Algorithm from [27]

```
   Input: $P$, $N$ , $c$, $\Omega$ , $c_{th}$ , $\phi_{th}$
1  Initialization:
2  $R \leftarrow \Phi$ $A \leftarrow 1, ..., |P|$
3  Algorithm:
4  while {A} is not empty do
5      Current region $R_c \leftarrow \theta$
6      Current seeds $S_c \leftarrow \theta$
7      Point with minimum curvature in $A \rightarrow P_{min}$
8      $S_c \leftarrow S_c \cup P_{min}$
9      $R_c \leftarrow R_c \cup P_{min}$
10     $A \leftarrow A \setminus P_{min}$
11     for i=0 to size ($S_c$) do
12         Find nearest neighbors of current seed point $B_c \rightarrow \Omega(S_c i)$
13         for j=0 to size ($B_c$) do
14             Current neighbour point $P_j$
15             if A contains $P_j$ and $\arccos(|(NS_c i, NS_c j|) < \Omega_{th}$ then
16                 $R_c \leftarrow R_c \cup P_j$ $A \leftarrow A \setminus P_j$
17                 if $cP_j < c_{th}$ then
18                     $S_c \leftarrow S_c \cup P_j$
19     Add current region to global segment list $R \leftarrow R \cup R_c$
   Output: R
```

This type of region growing offers several advantages over conventional segmentation techniques like the ability of separate regions with different properties like the height of the points or the distance between points, the possibility of choosing different criteria at the same time, and the capacity to grow the region with a small number of seed points. Moreover, this method performs well with the respect to noise. On the other hand the algorithm is computationally heavy for dense point clouds and must be tuned to deliver a usable result.

2.4 Plane detection

Several methods of plane extraction were developed in the last years [3] [29] [11], being that plane detection is important in terms of mobile robotics because, with the development of sensors capable of producing large datasets with millions of points, there is a need to reduce the processed amount of data to increase the quality of the created applications. Plane detection is important namely for ground plane detection where this method can be used to map the free space on the ground plane, as will be demonstrated in Section III. A more ambitious approach can be carried out if we combine the detection of planes with the segmentation, which can allow to map ramps or obstacles that a robot is able to overcome, improving the navigation safety of a robot.

2.4.1 Planar Polygon Extraction and Merging

The planar Polygon Extraction and Merging from Depth Images method, presented in [3], is a solution used in indoor scenarios for multiple plane detection using depth images that extract dominant planar features from 3D point clouds. This method considers that for each observed depth image only planar regions are extracted in order to merge the model of the environment. The approach can be divided in three steps[3]:

1. For each observed depth image, compute polygons to best approximate the dominant planes in that image.

2. Find correspondences between the polygons in the current image with the polygons in the map.

3. Iteratively merge corresponding polygons from successive images, thus growing the map.

The approach starts by using the Fast Sampling Plane Filtering (FSPF) algorithm [2] to extract points that belong to the local neighborhoods of planes from depth images. Then computes the plane parameters of each neighborhood by eigenvector decomposition of the scatter matrix and define the polygon boundary. Each scatter matrix of the created polygon is then stored in a decoupled manner that allow to merge polygons over successive observations and compute the least square fit over all observed points, over time, without maintain a list of them. After this, the correspondences between the polygons of the latest depth image and the polygons in the map are determined, using an image render based raycasting method, and if no match is found during this process then the polygons are added as new polygons of the map.

2.4.2 Random Sample Consensus

The Random Sample Consensus algorithm (RANSAC), first introduced by Fischler and Bolles in 1981 [11], is a re-sampling technique that estimates parameters of a mathematical model by random sampling observed data assuming that all data is composed by inliers and outliers. As pointed out in [11], the RANSAC algorithm uses the smallest possible set of data and enlarges this set with consistent data points when it is possible, unlike conventional smoothing techniques that use as much data as possible to obtain an initial solution and then attempts to eliminate invalid data points.

RANSAC selects a random subset of data and uses it to estimate the model parameters using a voting procedure to find the optimal fitting result. This result is achieved by training a model that has the most count of samples and the smallest average error among the generated models. The basic RANSAC algorithm can be summarized by the 2 [7].

Algorithm 2: RANSAC Algorithm obtained from [7]

```
1 Select randomly the minimum number of points $\beta$ required to determine the model
    parameters.;
2 Solve for the parameters of the model.;
3 Determine how many points from the set of all points inputData fit with a
    predefined distance threshold $\epsilon$;
4 If the fraction of the number of inliers over the total number points in the set
    exceeds a predefined threshold $\tau$ , re-estimate the model parameters using all the
    identified inliers and terminate.;
5 Otherwise, repeat steps 1 through 4 (maximum of M times).;
```

2.5 Path planning

Path planning can be described as the process of finding the optimal path from the start point to a predetermined destination, or to a direction given by the user, while avoiding collisions along the pathway. Path planning may require a robust localization and a map in order to ensure that the robot will follow the computed path considering the velocity, position and the surrounding environment. The path planning methods can be divided into two main groups that can work alongside: global planners and local planners.

The global planning approach computes the complete path for the robot to follow from an initial to a final location. There are two main approaches for global path planning: sampling-based and search-based algorithms. Sampling based algorithms, such as probabilistic roadmaps [19], randomized path planners [5] and rapidly-exploring random trees [21], use a randomization of the configuration space. Search-based algorithms, for instance, A^* and D^*, generate a graph representation of the planning problem.

Local planning approach uses local environment information stored in the costmap to generate velocity commands for the robot navigation and do not require a robust robot's localization. For this reason, these algorithms require much less computational time and allow higher rates of sensor information than the global planning, which represents an advantage for fast response of the system to the sudden changes in the surrounding environment of the robot. Some examples of these approaches are: 1) potential field methods [20], where the robot is attracted by a positive force in the direction of the goal and the obstacles impose negative forces, resulting in a velocity given by the difference of these forces; 2) the vector field histogram [4], which uses an occupancy grid of the environment to construct an histogram representing the free space around the robot and it is used to compute velocities; and 3) The Dynamic Window Approach (DWA), which is addressed in Section 4.

3 Local Environment Perception

The perception of the robot's surrounding environment is an important task in order to provide a reliable model where safe motion primitives can be taken. Considering that the robot is inserted in a three dimensional environment but its motion is in a 2D plane parallel to the ground, it was necessary to create a method that generated a rich 2D representation of the environment (more suitable for realtime application) without losing information of the height of the obstacles, thus allowing to correctly map obstacles, such as tables, chairs and stairs but also non trivial obstacles such as gutters, where a standard 2D approach could lead to collisions or hazardous scenarios. In this section we will present the local environment perception approach considered in this dissertation.

3.1 3D Point Cloud data to 2D occupancy grid map: Proposed method

The development of a reliable and computationally efficient method for mapping $3D$ point clouds into a $2D$ grid-map is the main focus of the proposed method. This is a very important topic in terms of safe navigation because it allows us to correctly map obstacles that are in the robot's pathway but are neither trivially detected nor efficiently mapped by common 2D mapping approaches. In Fig. 3.1 the proposed pipeline is presented with the different steps of the 3D point cloud to 2D grid map proposed method: starting with a 2.5D representation obtained from a 3D point cloud followed by a ground-plane detection and a 2.5D-to-2D conversion, using a inverse sensor model (ISM).

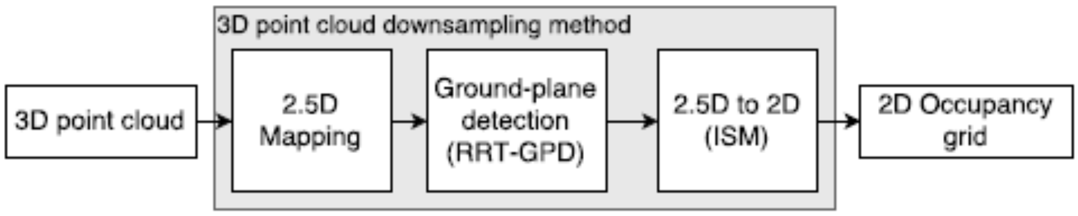

3D point cloud downsampling method
3D point cloud
2.5D Mapping
Ground-plane detection (RRT-GPD)
2.5D to 2D (ISM)
2D Occupancy grid

Figure 3.1: Stages of the proposed method.

The main modules parameters, including the inputs and outputs of every module, from the proposed method are represented in table 3.1.

	Obtained from	Size	Input	Output
3D Point Cloud	Kinect One	512 x 424	-	-
2.5D Mapping	-	300 x 300	3D Point Cloud	2.5D Elevation Map
Ground Plane Detection	-	1	2.5D Elevation Map	Ground Plane Detected
2.5D to 2D (ISM)	-	300 x 300	2.5D Elevation Map & Ground Plane Detected	2D Local Occupancy Grid Map

Table 3.1: Main modules parameters of the 3D Point Cloud data to 2D occupancy grid map proposed approach.

3.1.1 2.5D Mapping

The first module builds on the construction of a $2.5D$ environment representation from a $3D$ point cloud [26]. The $2.5D$ representation discards the concept of occupancy and provides an elevation measure for each cell. The proposed $2.5D$ grid is composed by mx x my cells with constant resolution d_r. Each cell $c_{2.5D}$ is addressed by i and j indexes and has the elements $c_{2.5D} \leftarrow \{z^-, z^+, N_z\}$ where z^- is the minimum elevation value, z^+ the maximum elevation value and N_z the number of samples that contribute with information to the cell. Given a $3D$ point cloud composed by a set of $3D$ Cartesian points $p_k = (x_k, y_k, z_k)^T, k = 1, 2, ..., n$ the values present in each cell are obtained by projecting the x and y components on the grid cell with consequent update of the maximum and minimum elevation, z^+ and z^-, and the number of samples $\|z\|$ projected to the given cell. The projection consists on the conversion of the x and y component from p_k into grid indexes i, j with $i \leq mx$, $j \leq my$, $mx; my > 0$. The $3D$ to $2.5D$ maps conversion process is illustrated by an example in Fig 3.2.

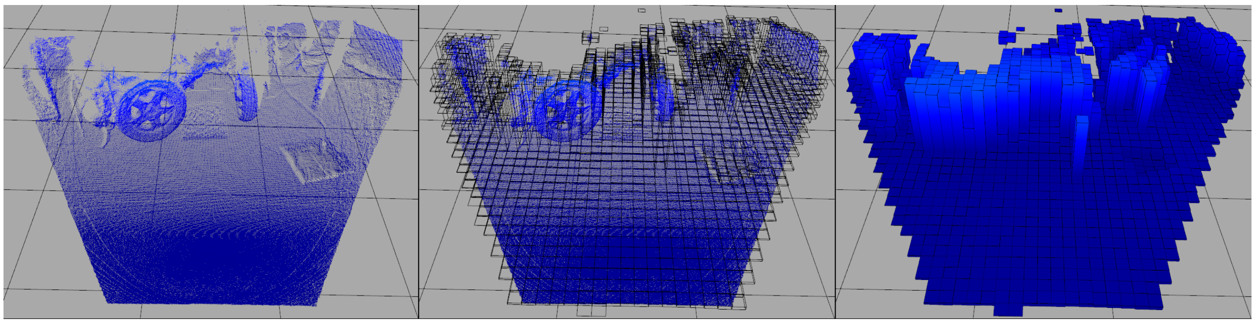


Figure 3.2: Representation of the conversion from 3D point clouds to 2.5D environment representation. From left to right, the input 3D point cloud (given by a Microsoft's Kinect One), the projection step where each point in the point cloud is projected into the corresponding cell, and the final result where each cell resembles a voxel defined by the diference between maximum and minimum elements projected onto the cell.

3.1.2 Rapidly exploring random tree based ground-plane detection (RRT-GPD)

When we convert an elevation map into an occupancy grid the definition of free and occupied $2.5D$ cells can be a problem because there is no reference to the location of the ground plane. This means that a robot will be unable to perceive what is navigable space and what is an obstacle.

We can assume, for a given frame, that a robot moves locally in a $2D$ plane and based on this constraint we present a new approach [13] to the problem of ground-plane detection that operates with a $2.5D$ map. This approach is inspired by the rapidly-exploring random tree (RRT) algorithm, which has been widely applied in motion planning [21]. The proposed approach is also inspired by the region growing [27], in the way that it provides a rapid flood-like expansion towards similar normals.

The inputs of the proposed RRT-GPD method include, the robot's pose, the plane's normal threshold, elevation threshold, the $2.5D$ environment model, the maximum number of iterations to find a suitable solution, and a node expansion distance. The RRT-GPD **Algorithm** 3 starts by finding a valid seed to form the root of the RRT (**nearestValidSeed**); it is to be noted that the use of an invalid seed could lead to an ill formed tree due to the sparse nature of the $2.5D$ representation, as shown in Fig.3.2. This means that not all of the cells provided by the $2.5D$ map, $M_{2,5D}$, will contain information and besides that, a seed far from a valid cell may be unable to expand.

During the process of finding a valid seed, the robot's pose provides an initial guess and every direction is explored to find a valid candidate. This is done because no assumptions are made on the sensor's configuration (i.e., transformation between sensor and robot). Although it is important to find a valid candidate to initialize the exploration, the final solution may discard the initial seed point, meaning that it is not guaranteed that the final solution will contain the initial point. This description is based on the assumption that the $2.5D$ map is computed with the sensor referenced to the robot's frame and that exists a relation between the robot and the environment, being this relationship given by transforming the robot coordinates (x, y, z) in the coordinates (x, y, z) of the world for a local map, where a mobile robot is centered in $p_{xyz} = (0; 0; 0)^T$, or for a representation in the referential of the sensor the constraints are similar. In this particular case the concept of valid seed implies a valid local plane, computed by the **fitPlane** function (see algorithm 3). After finding a valid seed and a correspondent local plane, a new tree is generated in **initializeTree**, where the seed, which corresponds to the center of the plane, and the plane normal define the base unit (or node) of the RRT tree.

The algorithm then enters an iterative process to expand the RRT until it reaches K iterations, being that the first step for each interaction involves sampling (**sampleRandomDirection**) a point x_{rand} in a search space window. For a given sample node x_{rand}, the nearest node x_{near} already present in the tree is retrieved (**nearestNode**) and a new node $x_{expansion}$ is created based on the direction between x_{near} and x_{rand} (provided that they do not overlap). The direction is given by 3.1 (**angleBetween**),

$$\theta = \arctan(\frac{x_{rand}(y) - x_{near}(y)}{x_{rand}(x) - x_{near}(x)}) \tag{3.1}$$

and the (x, y) component given by 3.2,

$$x_{expansion}(x, y) = \begin{cases} x_{near}(x) + d_n \cos(\theta) \\ x_{near}(y) + d_n \sin(\theta) \end{cases} \tag{3.2}$$

where dn denotes the node expansion distance. For the new candidate node $x_{expansion}$, the z component is given by the function **elevation**. As we said before, given the sparse nature of the $2.5D$ grid, not every $x_{expansion}$ will correspond to a valid cell when projected to $2.5D$.

To provide a valid z component, a spatial interpolation method was applied on the node neighborhood (a region of interest with dimensions (sx,sy) centered in $x_{expansion}$) and if an invalid z value is retrieved, i.e., no valid neighbors were found, the current iteration is ended. With a valid $x_{expansion}$, and applying a similar spatial interpolation method, a set of neighborhood points P is extracted, where empty points missing the z component are interpolated and discarded if the neighborhood does not contain valid elements. If the neighborhood P contains at least three non-collinear points, a least squares regression is performed to find the best planar fit to the points (**fitPlane**) of the form $ax+by+cz+d=0$. Given a valid plane $plane_{local}$, the inner product of the normal stored in x_{near} and the planes' normal (**normal(**$plane_{local}$**)**) is computed and if it is less than a given threshold (Nth) the plane is considered to be at least similar in orientation given the node x_{near}. In order to analyze whether the new plane can connect or not with the nearest node, the variation in height is also checked using the elevation threshold (Eth). If the normals are not similar, the nodes near a radius dn from $x_{expansion}$ are retrieved (**nearestNodes**) and the same similarity thresholds are applied to validate further node connections. If a pair plane-node is conformant, i.e., is valid for each threshold, a new node is added to the tree, containing the parent connection (x_{near}), the new center point (computed plane midpoint) and the correspondent plane normal. Other tree expansion constraints such as elevation gradient (also evaluated in this dissertation) can be employed.

The last step after adding a node to the tree is to update the search space window (**updateSearchSpace**). The search space window starts centered on the seed node but with each added node and each iteration, the search window grows and shifts towards the average value (geometric center) of the explored nodes, slightly biasing the search process to areas with similar properties where the expansion is more prominent, but without leaving out unexplored areas. After K iterations, the center points of each node on the RRT tree are extracted (a Kd-tree is used at the algorithm's core to store each discovered node) and a new plane fitting is performed. In this final step, we apply the random sample consensus (RANSAC) algorithm [11] due to its robust estimation even in the presence of outliers. The RRT-GPD algorithm is summarized in the pseudocode of **Algorithm 3**.

Algorithm 3: Rapidly exploring random tree based ground-plane detection (RRT-GPD) algorithm [13].

```
Input: Robot pose ($\mathbf{p}_{xyz}$),Plane normal threshold ($N_{th}$)
       Elevation threshold ($E_{th}$), 2.5D Map ($M_{2.5D}$)
       Maximum number of iterations ($K$), Node expansion distance ($d_n$)
1  Initialisation:
2  $\mathbf{p}_0 \leftarrow$ nearestValidSeed($\mathbf{p}_{xyz}$, $M_{2.5D}$);
3  $G \leftarrow$ initializeTree($\mathbf{p}_0$);
4  for $k$=1 to $K$ do
5      $x_{rand} \leftarrow$ sampleRandomDirection();
6      $x_{near} \leftarrow$ nearestNode($G$,$x_{rand}$);
7      $\theta \leftarrow$ angleBetween($x_{near}$,$x_{rand}$);
8      $x_{expansion}(x) \leftarrow x_{near}(x) + d_n \cos(\theta)$;
9      $x_{expansion}(y) \leftarrow x_{near}(y) + d_n \sin(\theta)$;
10     $x_{expansion}(z) \leftarrow$ elevation($M_{2.5D}$, $x_{expansion}$) ;
11     $P \leftarrow$ interpolation($M_{2.5D}$, $x_{expansion}$);
12     $plane_{local} \leftarrow$ fitPlane($P$);
13     if $\mid$ normal($plane_{local}$) $\cdot$ normal($x_{near}$) $\mid \leq N_{th}$ then
14         if $\mid plane_{local}(z)$ - $x_{near}(z) \mid \leq E_{th}$ then
15             $G \leftarrow G \bigcup \{ x_{near}, x_{expansion}$, normal($plane_{local}$) $\}$;
16             updateSearchSpace($x_{expansion}$);
17     else
18         $x_{neighbours} \leftarrow$ nearestNodes($G$,$x_{expansion}$,$d_n$);
19         foreach node in $x_{neighbours}$ do
20             if $\mid$ normal($plane_{local}$) $\cdot$ normal(node) $\mid \leq N_{th}$ then
21                 if $\mid plane_{local}(z)$ - node$(z) \mid \leq E_{th}$ then
22                     $G \leftarrow G \bigcup \{$ node,$x_{expansion}$, normal($plane_{local}$) $\}$;
23                     updateSearchSpace($x_{expansion}$);
24 $g_{plane} \leftarrow RANSAC(Points(G))$
Output: $g_{plane}$
```

3.1.3 2.5D to 2D convertion

The conversion from $2.5D$ to $2D$ follows the same principles introduced with the integration of sensor readings for occupancy grid mapping [24], [32]. In this case, our observations are the elevation voxels present in the $2.5D$ grid-map, turning the representation into a virtual sensor. The probability that a cell c is occupied given the observations $z_{1:t}$ is given in log odds by:

$$l(c \mid z_{1:t}) = \log \frac{p(c \mid z_t)}{1 - p(c \mid z_t)} - \frac{p(c)}{1 - p(c)} + \frac{p(c \mid z_{1:t-1})}{1 - p(c \mid z_{1:t-1})} \tag{3.3}$$

with $p\left(c\right)$ the prior probability, $p\left(c|z_{1:t-1}\right)$ the previous estimate and $p\left(c|z_t\right)$ denotes the probability that cell c be occupied given the measurement z and it is computed using an ISM. The log odds representation is used here due to its numerical stability. To solve the $2.5D$ to $2D$ conversion problem we propose an ISM that converts an observation in the form of an elevation voxel c_v to the probability that given the actual observation, the cell from the $2D$ grid is occupied. Each elevation voxel can be defined as being in a valid state if it contains more than one measurement ($N_z \geq 1$). In order to determine the influence of each voxel we rely on the concept of voxel density explored in [8]. An elevation voxel c_v in the $2.5D$ map (see Fig. 2) occupies the volume given by $V_{voxel} = hA$, with h is the height of the voxel ($h = \Delta z = z^+ - z^-$) and A the base area of the voxel (i.e., based on the cell resolution). The voxel density is given by $\rho_{voxel} = \frac{m}{V_{voxel}}$, where m is the voxel mass. The mass of a voxel in this context can be defined as the amount of data the voxel contains and can be extrapolated using the number of samples Nz of an elevation voxel. To represent a normalized voxel density, the following sigmoidal function is proposed,

$$\rho_{voxel}(c_v) = \frac{K_m}{1 + e^{-\left(\frac{K_n(c_v(N_z) - d_{min})}{V_{voxel}}\right)}} \tag{3.4}$$

where K_m denotes an amplitude gain, K_n a sample normalization factor, and d_{min} the minimum number of points. The voxel is composed by 3 explicit parameters and an implicit one related to the distance $|c_v|$ defined by the voxel position in relation to the base frame (e.g., sensor frame, robot frame or local frame). In the previous subsection we presented the RRT-GPD method to extract the ground plane that from now on is denoted by g_{plane}.

The proposed ISM takes into account the voxel distance to the base frame, decreasing the elevation voxel occupancy probability as voxels move away from the base frame. Knowing a valid ground-plane allows for the definition of $free$ or *occupied* values in the sense that a specific cell contains a high or low probability of being occupied, for instance, if a voxel is near the detected ground plane it may be considered as part of the ground and thus contribute to decrease the cell's probability. On the other hand, if an elevation voxel is above the ground, it may be considered to be an obstacle and thus contributes to increase the cell's probability. The ISM is expressed by,

- If $|c| \ \epsilon \ [\ 0,\ |c_v|\ ]$:

$$p(c|z_t) = \begin{cases} \max(\rho_{voxel}, 0.5)\ e^{-\frac{(|c|-|c_v|)^2}{2\sigma^2}} & , if\ d > d_{pth} \\ K_g + \frac{0.5 - K_g}{1 + e^{-(|c|-|c_v|)}} & , if\ d \leq d_{pth} \end{cases} \tag{3.5}$$

- If $|c| \ \epsilon \ ] \ |c_v|, \ |c_v|^{max} \ ]$:

$$p(c|z_t) = \begin{cases} \max(\rho_{voxel} \ e^{-\frac{(|c|-|c_v|)^2}{2\sigma^2}}, 0.5) & , if \ \ d > d_{pth} \\ 0.5 & , if \ \ d \leq d_{pth} \end{cases} \tag{3.6}$$

where σ^2 denotes the Gaussian variance, K_g an amplitude and bias gain with $0 \leq Kg \leq 0.5$, d_{pth} a distance threshold and d the distance between the plane g_{plane} and the voxel c_v. The distance between the voxel cv and the plane is computed using the maximum plane distance to the points $p^+ = (x; y; z^+)^T$ and $p^- = (x; y; z^-)^T$ where x and y represent the voxel position. The $2D$ grid-map structure is identical to the $2.5D$ counterpart introduced in III-A with each cell c_{2D} composed only by an occupancy value.

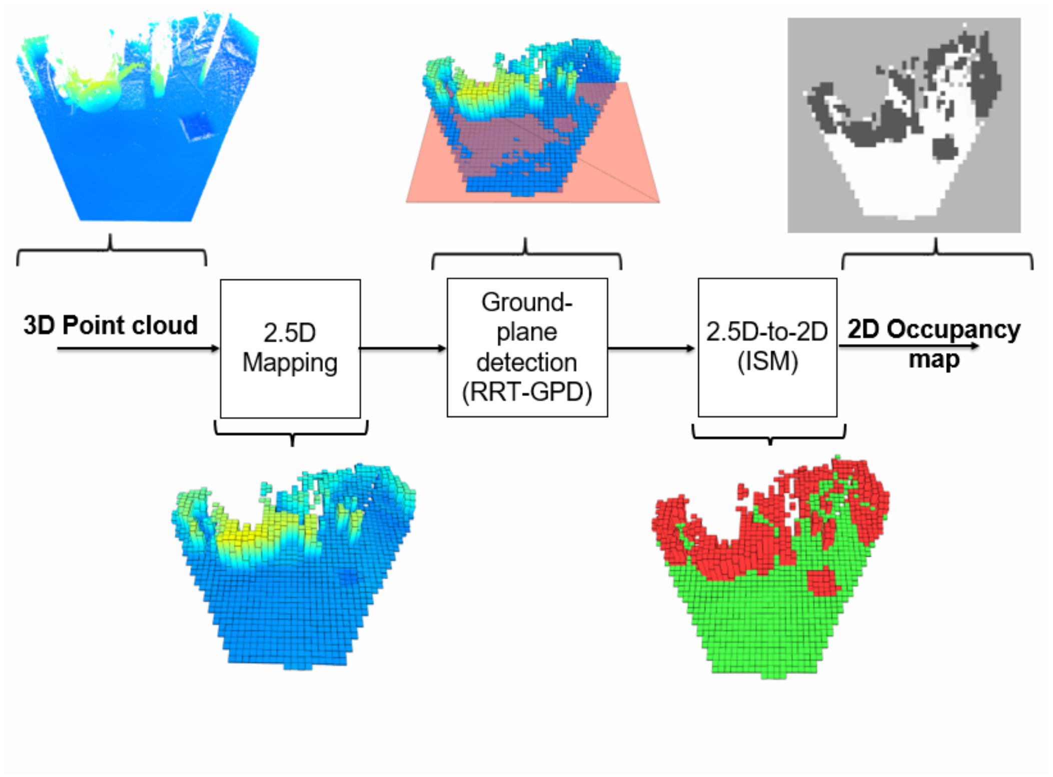


Figure 3.3: The algorithm developed for local perception consists of three blocks: 1) Conversion of the 3D point cloud to a 2.5D elevation map; 2) Ground plane detection; 3) Conversion of the 2.5D elevation map to a 2D occupancy grid map using the inverse sensor model.

Finally, we were able to obtain a 2D occupancy grid map capable of modeling obstacles with irregular shapes and heights that will be used in the following section for the robot's motion planning. An overview of the output from each step of the algorithm is represented in Fig. 3.3, representing at the last stage the resulting 2D occupancy grid map.

4 Robot Assisted Navigation

Navigation in complex and unknown environments is a major challenge for people with walking or visual disabilities. This includes path planning and obstacle avoidance, which has been investigated since the beginning of robotics. The difficulty to solve this problem increases when the detection and avoidance of dynamic elements in the environment becomes one of the main requirements. When considering the case where the robot is a physical support aid, such as in gait rehabilitation scenarios, the safety of all agents is even more crucial. The robot must ensure that it follows all the users instructions accordingly, while obeying safety rules taking into account the knowledge of the surrounding environment. The mobile robot planning problem differs from the specific scenarios involving most assistive robots such as walkers or wheelchairs as it is also important to incorporate the user's intent in the decision process.

In this chapter we will talk about the navigation strategies used in this work to safely avoid obstacles.

4.1 Dynamic Window Approach

The dynamic window approach [12] is a local path planning algorithm that provides optimal local solutions to collision avoidance. This approach reduces the search space to the translational and rotational velocities which are reachable considering the kinematic and dynamic constrains of the robot. In addition to this restriction, this model only considers velocities that can be reached without putting the robot on a collision trajectory with an obstacle, choosing the rotational and translational velocities that maximize the efficiency of the model.

The DWA approach can react very quickly to sudden changes in the surrounding scenario due to its low computational complexity, providing an excellent choice for the navigation of robotic platforms (including but not limited to assistive robotic walkers).

4.1.1 Search space

With the dwa method there is a need to reduce the velocity search space to a set of translational and rotational velocities (v, w), that are achievable by the robot in a short period of time, that guaranties that the robot will not hit any obstacles. Restricting the velocity search space can be done using the following three steps:

Circular trajectories (V_s)

The generation of a trajectory to a given goal point for the next n time intervals can be approximate by a sequence of circular arcs trajectories. For this purpose the velocities (v_i, w_i) must be determinate for each n time intervals between $t0$ and tn, considering that the resulting trajectory does not intersect any obstacle. This results in a two-dimensional velocity search space.

Admissible Velocities (V_a)

Obstacles in the vicinity of the robot impose restrictions on the rotational and translational velocities. For example, the translational speed decreases according to the decrease of the distance of the robot to an existing obstacle in its pathway, on the other hand the rotational velocity will increase to avoid the collision to the obstacle and if there are any alternative paths the robot will stop and move back to find a valid trajectory to follow. This means that a pair of velocities (v, w) is considered admissible if the robot is able to stop before it reaches the obstacle.

Considering that the term dist(v,w) represents the euclidean distance to the closest obstacle in the robot's trajectory and $\dot{v_b}$ and $\dot{w_b}$ are maximal translational and rotational breakage decelerations, then the set of admissible velocities Va is defined as

$$V_a = (v, w)|v \leq \sqrt{2dist(v, w)\dot{v_b}} \wedge w \leq \sqrt{2dist(v, w)\dot{w_b}} \tag{4.1}$$

Dynamic Window (V_d)

Taking into account the limited accelerations that are possible to perform by the motors, the overall search space is reduced to the dynamic window that contains the set of velocities that are possible to archive within the next time interval t. Let $\dot{v}$ and $\dot{w}$ be the accelerations applied and (v_a, w_a) the actual velocity during the time interval t, then the dynamic window

V_d is defined as

$$V_d = (v, w)|v \in [v_a - \dot{v}.t, v_a + \dot{v}.t] \wedge w \in [w_a - \dot{w}.t, w_a + \dot{w}.t] \tag{4.2}$$

Resulting Search Space

Combining the given restrictions imposed on the search space, result in the following intersection of restricted areas, V_r, used to compute the objective function.

$$V_r = V_s \cap V_a \cap V_d \tag{4.3}$$

4.1.2 Proposed Method

The DWA approach applied in this dissertation, presented in **Algorithm 4**, begins by transforming the local map, obtained from the perception package and explained in the previous section, in a cost map. This needs to be done because the probability value of the local map is given in logarithmic values and in terms of navigation it makes more sense to have values from 0 to 100 to represent the occupancy of a given cell, being that for navigation purposes we consider that values below 25 represent a free cell, values higher than 75 a occupied cell And the values in between are considered unknown space.

The proposed algorithm of the DWA approach starts by computing the feasible velocity space. Considering v_{max} and w_{max} the linear and angular maximum speeds, a_{max} and a_{wmax} the respective maximum accelerations, v_l and v_w, the robot actual linear and angular speeds and Δt the simulation time step, then the linear and angular maximum velocities, Vl_{max} and Vw_{max} of the feasible velocity space are given by:

$$Vl_{max} = min(v_{max}, v_l + a_{max} \times \Delta t); \tag{4.4}$$

$$Vw_{max} = min(w_{max}, v_w + aw_{max} \times \Delta t); \tag{4.5}$$

Considering then the linear and angular minimum velocity, v_{min} and w_{min}, the minimum linear and angular velocities, Vl_{min} and Vw_{min}, of the feasible velocity space are given by:

$$Vl_{min} = max(v_{min}, v_l - a_{max} \times \Delta t); \tag{4.6}$$

$$Vw_{min} = max(w_{min}, v_w - aw_{max} \times \Delta t); \tag{4.7}$$

This approach continues by simulating a pre-defined number of trajectories, n_{traj}, obtained from the previous velocities, by incrementing the acceleration of linear and angular velocities, in (4.6) and (4.7), during a period of time Δt. This increment is given by:

$$dv_{linear} = \frac{Vl_{max} - Vl_{min}}{n_{traj}} \tag{4.8}$$

for the linear increment and by

$$dv_{angular} = \frac{Vw_{max} - Vw_{min}}{n_{traj}} \tag{4.9}$$

for the angular increment. The number of trajectories, n_{traj}, chosen was obtained by trial and error, being that the ideal number of trajectories that best suited the navigation.

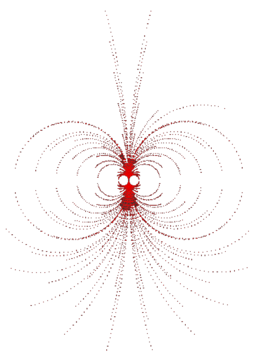


Figure 4.1: Trajectories created by the linear and angular velocities.

The algorithm then checks for collisions and verifies if any of this trajectories will go off the map, updating costs along the way accordingly with the number of steps that the robot must take along it's trajectory to guarantee the safety of the user. Finally, the cost of a trajectory is computed and compared with every trajectory cost created, applying to the walker the respective linear and angular velocities from the trajectory that has the smallest cost. The cost, C, of a given trajectory can be expressed as follows:

$$C = |Vl - Vl_u, Vw - Vw_u| + C_{occ} * k_s + \frac{k_0}{C_{obs}} + k_1 + k_2 * \frac{1}{1 + d_p} \tag{4.10}$$

Where Vl, Vl_u, Vw, Vw_u are the linear and angular velocities of the robot and the user, respectively, C_{occ} is the occupational cost, C_{obs}, the cost of the nearest obstacle, d_p is the cost of the pose distance, k_s, k_0, k_2, the scaling factors and k_1 the scaling factor that changes if the direction of the trajectory sample is different from the desired direction of the user.

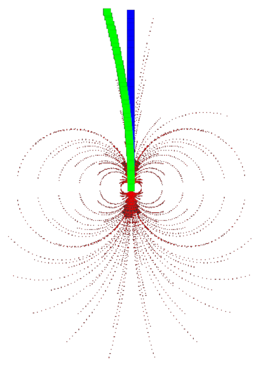


Figure 4.2: Complete velocity setup: 1) Desired trajectory of the user in blue; 2) Trajectory chosen by the DWA algorithm in green; 3) Complete set of trajectories in red.

A graphical environment was then developed with the purpose of helping to understand the trajectory that the algorithm had chosen and, in Fig. 4.1, it is possible to see in red the 24 trajectories considered, 12 forward and 12 backwards and in Fig. 4.2 the desired trajectory of the user in blue and the trajectory chosen and applied to the walker in green. This was an important tool for the validation of the method, as we will demonstrate in the next section.

Algorithm 4: Dynamic Window approach algorithm

```
   Input: Occ map, $(v, w)_{user}$, $(v, w)_{robot}$, Odom
1  Initialisation:
2  while $Odom = true$ do
3      CreateCostMap(rows, columns, cellsize, base, freeValue, occupiedValue);
4      ComputeVelocities($(v, w)_{robot}$, $v_{max}$, $w_{max}$, $v_{min}$, $w_{min}$, dt);
5      for $i$=1 to $i < v_{samples}$ do
6          $v_{sample} = min_v + (i * dv]$;
7          for $j$=1 to $j < w_{samples}$ do
8              $w_{sample} = min_w + (i * dw]$;
9              ComputeTrajectories(CostMap, $v_{sample}$, $w_{sample}$, dt, $(v, w)_{user}$);
10             if $cost > 0$ then
11                 SelectBestTrajectory($best_{traj}$, $comp_{traj}$);
   Output: $BestTrajectory(v, w)$
```

5 Experimental Results

This chapter describes the experimental setup and the test scenarios developed to validate the proposed methods. Validation was carried out in two sets of experiments, first the validation of the local environment perception and finally the navigation of the robotic platform (ISR-AIWALKER). The first one was made using a series of datasets recorded in places where this type of robot can be found on a daily basis. The experimental results of the second part was also divided into two parts, testing the navigation of the robot using static and dynamic obstacles. The validation of the navigation also incorporates the local environment perception because it requires a environment representation (2D occupancy map) for the navigation of the robot, being that in this part the two methods are combined and validated in different environments and conditions.

The experiments were carried out in a mid-range laptop with Kinect and Leddar data acquired at the frequency of 10Hz, the average time per frame for the proposed method was less than $30ms$, and the ground-plane detection less than $3ms$. The configuration parameters used for the robot assisted navigation package, represented in Table 5.1, were suited for the ISR-AIWALKER in order to guarantee a smooth navigation for the user.

Table 5.1: Configuration Parameters

Maximum Speed		Maximum Acceleration		Forward safe distance (m)	Simulation time (s)	Simulation time step (s)	Time constant δt (s)
Linear (m/s)	Rotational (rad/s)	Linear (m/s^2)	Rotational (rad/s^2)				
0.3	0.75	0.5	1.0	0.325	7.0	0.1	0.5

5.1 Experimental Setup

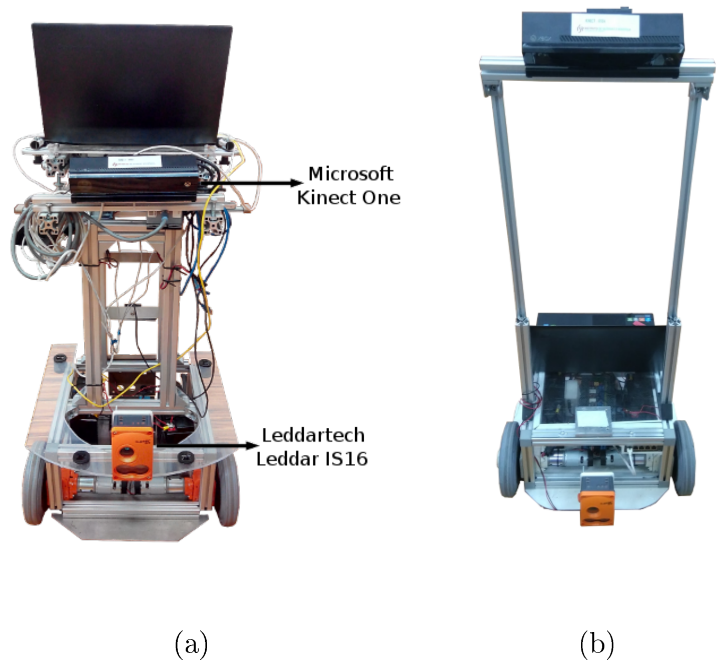


Figure 5.1: Robotic platforms: (a) ISR-AIWALKER used for validation of the perception module; (b) ISRobot2 used for validation of the navigation module.

The experimental evaluation of the proposed methods was carried out in two platforms;the ISR-AIWALKER and the ISRobot2 (see Fig.5.1). The ISR-AIWALKER [18] (walker platform) is composed by a differential mobile robot at the base, two grips interfaced with Leap Motion sensors for the command velocities of the user and a gait perception module aided by a $3D$ sensor. The ISR-AIWALKER was used for the validation of the local environment perception module presented in Section 5.3. The ISRobot2 platform has the same dimensions as the ISR-AIWALKER, is composed by a differential mobile robot at the base and a joystick was used to simulate user speed commands. The ISRobot2 was used for the validation of the robot assisted navigation presented in Section 5.3. Both platforms were equipped with two sensors, the Microsoft Kinect One and the Leddartech Leddar IS16, for environment perception, assessment of hazardous situations and safety purposes. The Kinect One outputs a 512×424 point cloud and the Leddar IS16 delivers a 16-channel distance array, but the first one cannot output distances less than 50cm while the second can, ensuring in this way that the platform will not collide with obstacles that may appear in the way, for example, during a rotation.

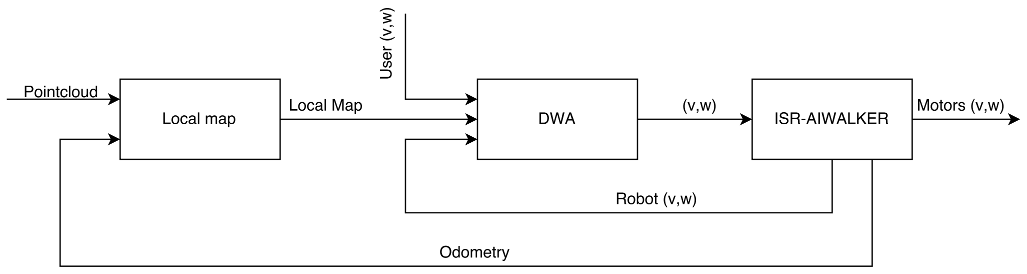


Figure 5.2: Software architecture representing all of the inputs and outputs of the perception and navigation modules.

The perception, navigation and visualization modules used on both platforms run in ROS nodes, and all experiments reported in this section were carried out in the same environment with $C++$ implementations. The following diagram shows how the system software is connected.

5.2 Software Overview

This work has been developed in ROS mainly because it supports $C++$ implementations and there is a great community that provides useful and updated open-source tools and documentation for the most of the robotic applications. There are some concepts such as package node, topic, message, subscriber and publisher, defined in [14], that are important to understand since all ROS packages work with these elements. A package can contain one or several nodes (processes) defined as publishers, subscribers or both. For example, one node controls a Kinect One, one node controls the wheel motors, other node performs the navigation and so one. A publisher node publishes topics containing messages, for example the Kinect One node publishes a pointcloud, and a subscriber node subscribes topics from publishers, processing this information, for example the Local Map node receives the pointcloud, processes the information and then publishes the result to other node. Other important topic is the relationship between coordinate frames of the packages used in a robotic system. This relationship gives us the location and orientation of the different components of the robot so that it is possible to process the sensory data according to the location of the sensors on the robot, for example the height and orientation of the Kinect on the platform

5.2.1 Physical Layer

The physical layer module is responsible for linking the hardware and software, being in charge of receiving the information of the used sensors and delivering commands, in our case velocity commands, to the platform. This linking is accomplished using drivers to connect the hardware devices, such as Leddar IS16 and Kinect One, to the software modules that process the received information and generate velocity commands to move the platform. The *iai_kinect2*, *freenect_lauch* and *leddartech* are drivers provided by the ROS community responsible for publish topics containing pointcloud messages from the Kinect One and Kinect 360 and scan messages from Leddar IS16. The *walker_driver* receive encoder mesurements from the powerdriver and using the kinematic model of the walker the odometry data is computed. This driver also receives velocity commands that are sent to the low-level part of the platform. This driver is based on the driver developed for the ISR intelligent wheelchair Robchair [14] and, although the robchair driver was made for a different platform, it can be adapted to other platforms with low-level architecture not much different from the wheelchair.

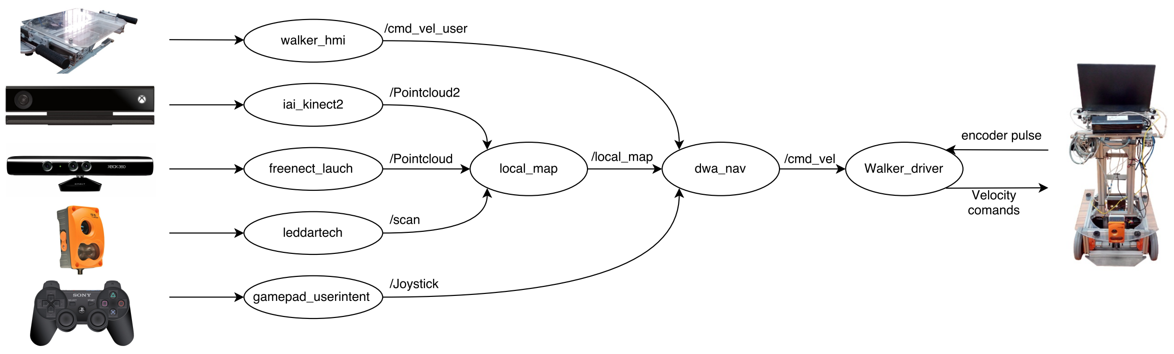


Figure 5.3: Physical layer architechture.

5.2.2 User interface

Our system was created to help the navigation of the user and there is a need for the user to interact with the system. With this in mind the *gamepad_userintent* package was used to receive velocity commands from the user and publish this information that will be used by the robot assisted navigation package to plan the best trajectory. A similar package, *walker_userintent*, was also used to receive user commands using the handles of the ISR-AIWALKER. This package is connected by TCP / IP to the HMI module proposed in [25] and provides the user intent on the ROS platform. We use the Rviz package, included in the ROS platform, to be able to see the available topics like the occupancy grid map, the trajectories generated and the user intent.

5.2.3 Perception Node

The type of mobile robot used in this dissertation as the need to perceive the surrounding area and find a way to localize itself in the environment. For this propose it was developed the *local_map* package that receives data from the sensors and odometry information from the *walker_driver* and then transform this information into an occupational map. The package also receive information about the system transforms in order to compute the information accordingly with its location source. This package was adapted from the original *local_map* package available in the ROS community that takes an input from a laser scan message and output a local map as occupancy grid. An overview of the perception module is represented in the following figure.

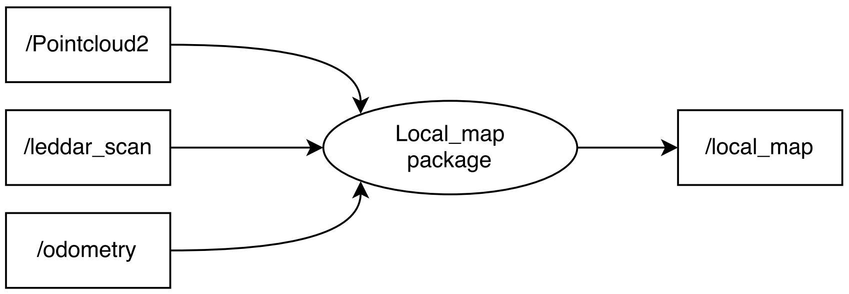


Figure 5.4: Perception module representation.

5.2.4 Navigation Node

The navigation package *dwa_nav* provides an implementation of the Dynamic Window Approach to local navigation. This package receives Odometry information from the *Robchair_driver*, velocity information from the *gamepad_userintent* or *walker_userintent* and a map created in the *local_map* package and then computes trajectories. The package then outputs the command velocities that lead to the best trajectory and the visualization messages of the computed trajectories, the best trajectory and the trajectory chosen by the user. The inputs and outputs of the navigation module are represented in the following figure.

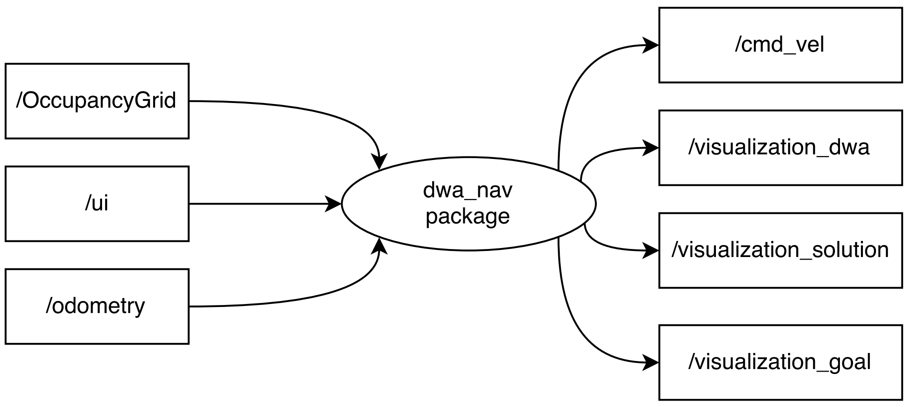


Figure 5.5: Navigation module representation.

5.3 Local Environment Perception

Five experiments were carried out, using the ISR-AIWALKER presented in the previous section, with the propose of validating the local environment perception module of this dissertation. The experiments were carried out in environments where a user may manifest some difficulties, such as ramps, and in environments where navigation is not permitted, such as stairs or obstacles that the user may not be able to overcome. All of these experiments were then compared with an available solution based on standard ROS package solution, presented in figure 5.6 in a local map framework. This available solution converts the point cloud to a laser-like scan and then projects the result to a local occupational map.

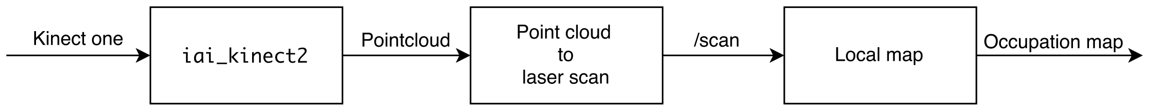


Figure 5.6: Standard ROS package solution overview.

The results were also compared with a PCL-based algorithm, inspired on the LIDAR sensor model presented in [1]. The PCL-based algorithm divides the point cloud into two stages: mapping of obstacles with height (z) between 0.2 and 1.5 meters and mapping of elements near the ground plane, considering for this heights smaller than 0.2 meters. The first one considers the coordinates (x, y, z) of the point, p, of the point cloud, assigning a probability of occupancy to points with height, z, between 0.2 and 1.5 meters. Using a ray tracing approach (Bresenham's line algorithm), it was possible to define a free path from the coordinate (S_x, S_y) of the sensor source to the coordinates (p_x, p_y) of the detected point. The second stage uses the points the point cloud for heights smaller than 0.2 meters to define the ground plane used as a reference for motion planning. For this, it uses the region growing algorithm to segment the points of the chosen region. These segments serve as inputs to the RANCAC algorithm where for each segment the correspondent plane is extracted. The ground plane is then selected taking into account the closest plane to the robot. Using the selected ground plane, as threshold is used to map the points ($z < 0.2$). In this way we can compute the occupation map by marking the points as free near the ground plane, and the reaming points with heights above or below the ground threshold are marked as occupied.

Qualitative results are shown in the respective images of each scenario being that the sequence of images describing the procedures (from image (a) to (e)) is the same for all of the scenarios. This sequence is given from left to rigth with real image (a), input pointcloud (b), occupational grid map with normal RTT Evaluator (c), occupational grid map with normal RTT Gradient Evaluator (d), occupational grid map with pcl method (e).

5.3.1 Test Scenario 1

The first scenario considered for the validation of the local environment perception module was a common amphitheater, where the user is faced with an access ramp from the entrance door, through the benches and ending on the stage. Two cases were then considered in this scenario, where in the first the user intends to descend the ramp and in the second where he intends to climb the ramp.

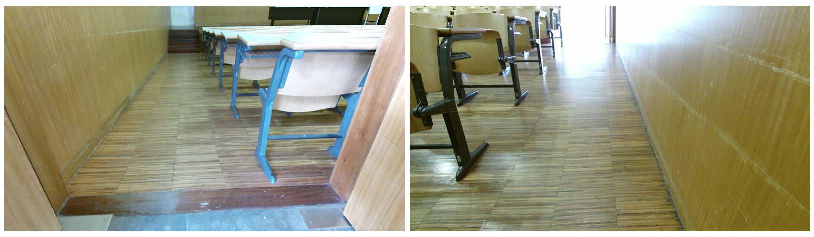

(a) (b)

Figure 5.7: Considered scenarios for the first test. (a) Amphitheater ramp down, (b) Amphitheater ramp up.

Each test lasted approximately five seconds and the results were analyzed separately. The results obtained for each of the methods considered are represented in the following figure.

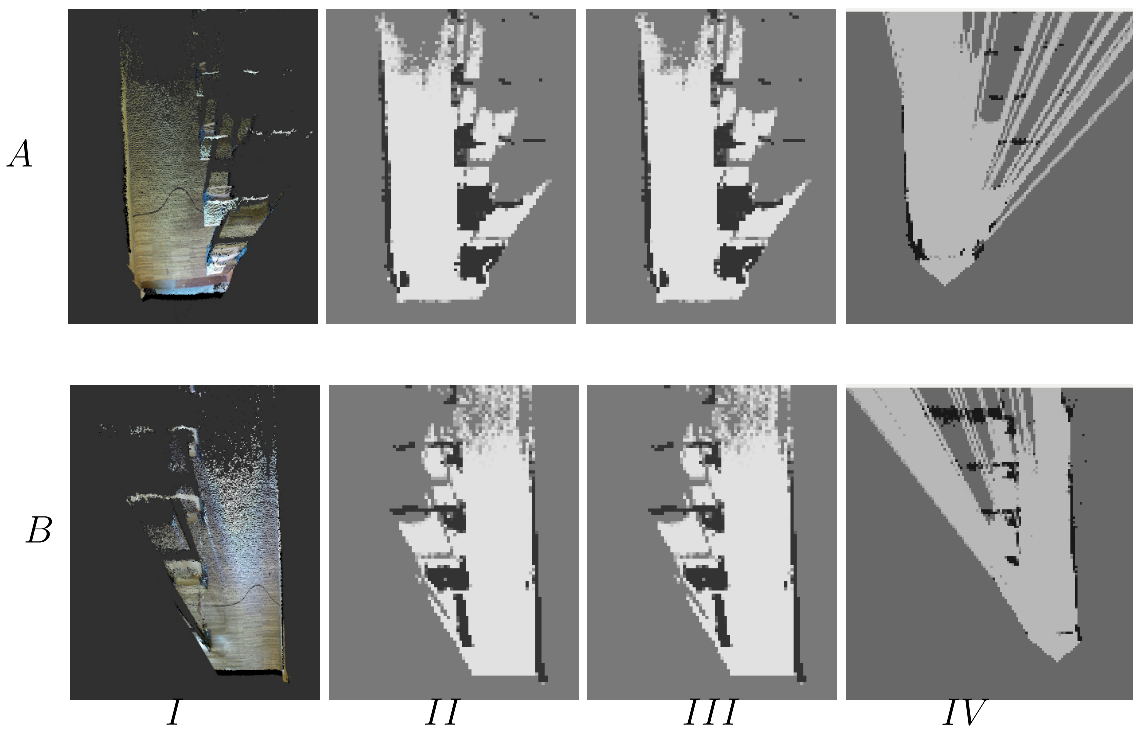


Figure 5.8: Results for the first scenario.

Results from 5.8 proves that the local map from Kinect One, using the RRT-GPD, Fig.$A-II$ and Fig. $B-II$, and the RRT-GPD with Gradient constraints, Fig. $A-III$ and Fig. $B-III$, provided by the proposed method, generate very similar results, being that the difference between them is reduced to some pixels that will not influence the navigation of the platform. The proposed method provided a reliable environment model by detecting the general scenario outline and the considered hazards as it correctly mapped the access ramp as well as the benches and the space between them. It is possible to verify that the result generated form the pcl method, Fig. $A-IV$ and Fig. $B-IV$, does not contain the same amount of occupancy information as the proposed method. This method correctly detects the wall and part of the tables, Fig. $A-I$ and Fig. $B-I$, but cannot properly detect the amphitheater chairs as well as the space between them. The biggest difference between the methods is the time each takes to generate results. The proposed method takes about 30 milliseconds to return the occupation map, contrary to the results obtained by the pcl method that took about a second to process the occupation map.

Then, the result of the proposed method for each of the sensors used was compared with the method provided by the ROS community using a laser scan obtained through the kinect One sensor. These results are shown in Fig. 5.9b for the descent of the amphitheater,

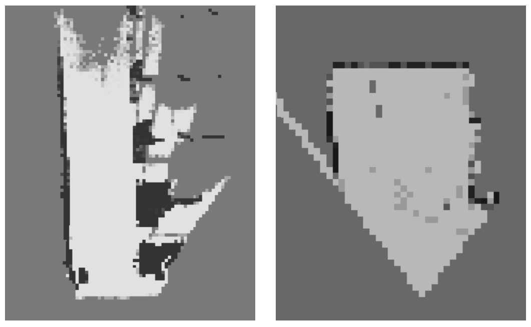

(a) (b)

Figure 5.9: Comparison between the proposed method (a) and the available ROS method (b) for the descent of the amphitheater.

and in Fig. 5.10 for the downward ramp of the amphitheater.

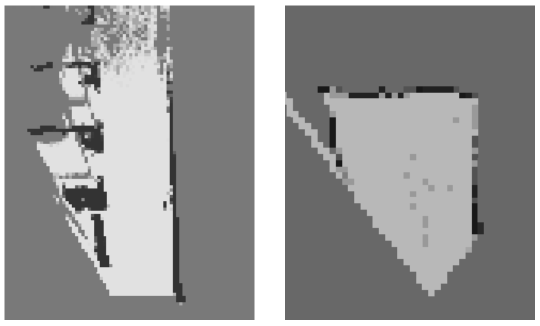

(a) (b)

Figure 5.10: Comparison between the proposed method (a) and the available ROS method (b) for the upward ramp of the amphitheater.

As expected the proposed method using Microsoft Kinect One provides a detailed map, which is the closest to reality, correctly detecting the wall and the chairs of the amphitheater, allowing safe navigation for the user of the walker.

The standard ROS solution provided only an outline of the scenario since the point cloud was converted to a laser-like scan and important points relevant to the detection of near ground objects were discarded because the laser scan does not provide information of obstacle heights, which in this case could lead to the collision of the walker with the tables.

5.3.2 Test Scenario 2

The second scenario considers an office environment that includes several tables and chairs, as well as a gutter on the floor. This was an important scenario precisely because it contains a gutter, which represents a dangerous obstacle because the user may not realize its existence, since it takes the walker in front of him, and therefore can get stuck or fall because of this obstacle, while the incorrect mapping of the remaining obstacles could not have such harmful effects as a fall.

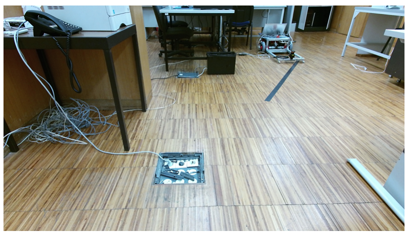


Figure 5.11: Test Scenario for the second test.

Through the analysis of the occupation map presented in 5.12 generated by each of the methods, it is possible to conclude that the proposed method, using RRT-GPD and RRT-GPD with Gradient constraints, in Fig. $A-II$ and Fig.$A-III$, generate very similar results. Again, the results obtained through the PCL method in Fig. $A-IV$ were unsatisfactory because the method, besides taking about a second to process the map, is not able to correctly map the gutter nor the tables at the bottom of the scene, unlike the maps generated in Fig. $A-II$ and Fig. $A-III$ where is possible to have a clear view of the obstacles and the free space between them.

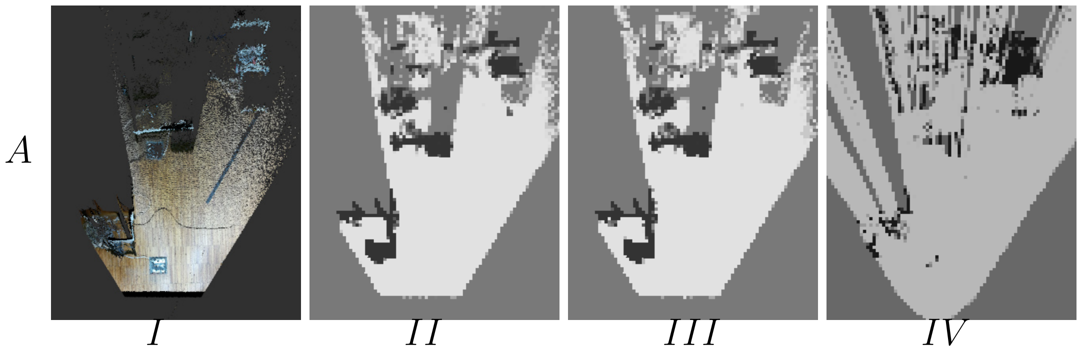


Figure 5.12: Results for the second scenario.

The results shown in 5.13 proves, once again, that the local map provided by our method presents the best result as it correctly detects the shape of the obstacles and the correctly mapped the floor outlet, providing a safer navigation for the user. The ROS approach correctly detects the table but can not detect the gutter, and as it is a laser scan does not provide as detailed information as the pointcloud since it can not detect the most distant obstacles, making it more difficult to find a valid trajectory for navigation propouses.

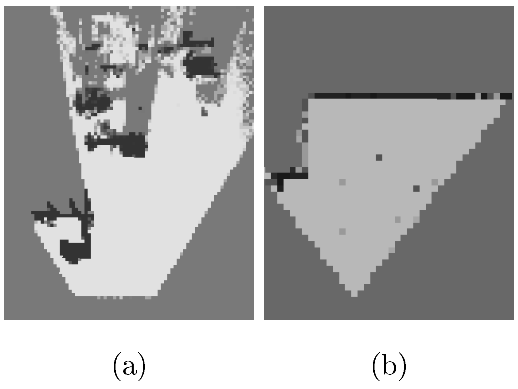


Figure 5.13: Comparison between the proposed method (a) and the available ROS method (b) in a office environment

5.3.3 Test Scenario 3

The third scenario, showed in Fig. 5.16a, considers a partially open door being that this scenario is one of the most difficult for most of the mobile robots, since the type of materials used in the floor may be different in each of the sides of the door, what makes the light reflection is different what It may lead to wrong calculation of the map of occupation if a sensor is used that collects data through the reflection of the light, which is the case of Kinect One used in this dissertation.

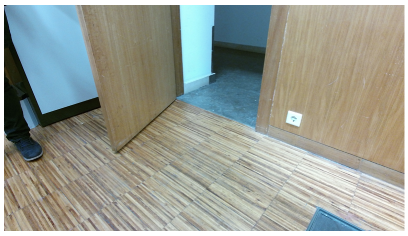


Figure 5.14: Test Scenario for the third test.

This test was not conclusive since there were times when the method worked correctly, especially in the cases tested during the night, and failed in cases where the difference of light between the two sides of the door, i.e. during the day, leading to an inconsistent mapping, which causes the robot to not go through the door every time the test was performed.

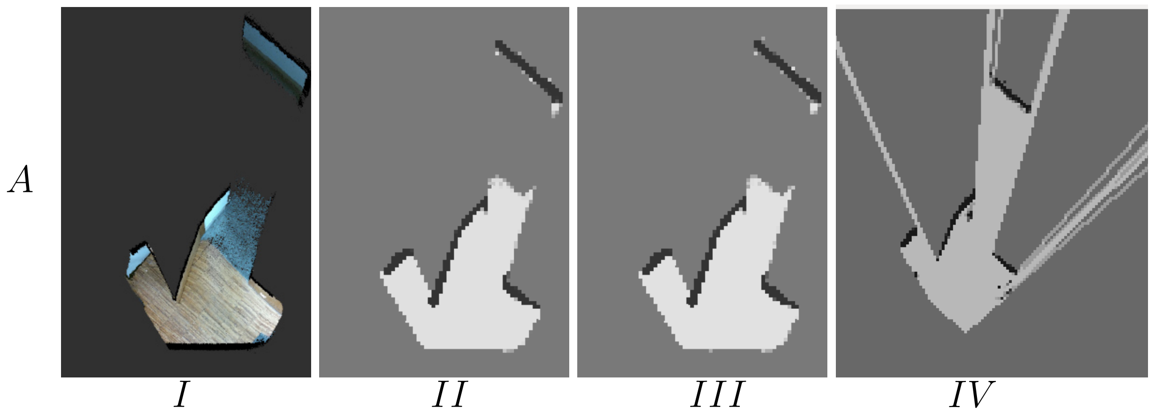


Figure 5.15: Results for the third scenario.

Considering the case where the method correctly mapped the occupied space, represented in 5.15, it was verified that the results obtained in the different methods are quite similar, since we have obstacles with very simple forms having been correctly detected the door and the space between it and the wall, as well as the foot of another user that opened the door for the user of the walker to pass.

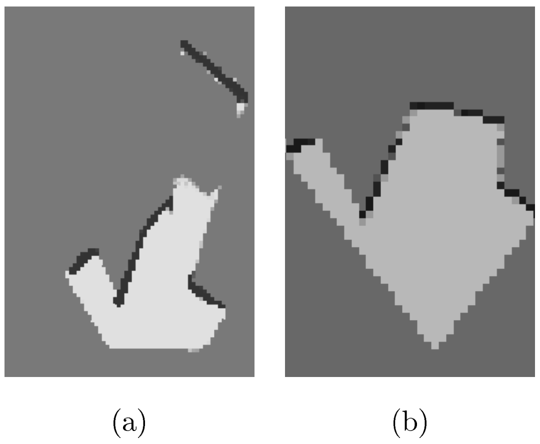


Figure 5.16: Comparison between the proposed method (a) and the available ROS method (b) for a partially open door.

The same result is verified when we use the method available in ROS, and in this case we find the same reflection problems because this method uses the point cloud of kinect One, which causes it to have the same problems as the method proposed using this sensor.

5.3.4 Test Scenario 4

The following scenario was the one that led to the development of this method because it is a very important scenario regarding the safety of navigation when rehabilitation devices are used. The scenario, divided in two, represents a descending and ascending staircase, in a scenario where it is important to ensure that the user does not fall from the stairs in the case of descent, and does not collide with the stairs to climb.

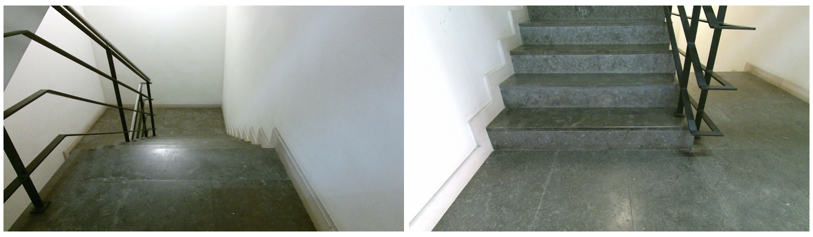

(a) (b)

Figure 5.17: Considered scenarios for the fourt test. (a) Staircase down, (b) Staircase up .

It is possible to verify through the analysis of figure 5.19 that the developed method, in Fig. $A - II$ and Fig. $A - III$, correctly represents the downward stairs, the railing and the walls, thus creating a occupation map where it is clearly possible to visualize the mentioned obstacles, which guarantees that there will be no doubt during navigation that there is a staircase and that this represents an obstacle that the walker will not be able to transpose. As for the ROS method, Fig. $A - IV$, correctly represents the stairs and a little of the walls but fails in the handrail, marking this space as free. This approach prevents the user from falling down the stairs but may lead to a colision with the rails. As for the second scenario, both methods, Fig. $B - II$ and Fig. $B - III$, can correctly represent the staircase, which would be expected since it is a scenario where it is possible to detect the vertical plane of the first step. Even though the map created by the proposed method is more perceptible than the map generated by the PLC method, Fig. $B - IV$, since it clearly detects that the stairs and the handrail are different obstacles, while the PLC method detects a plane between them, which leads to incorrect mapping of these two obstacles and, at the limit, this representation could lead the walker to a local minimum.

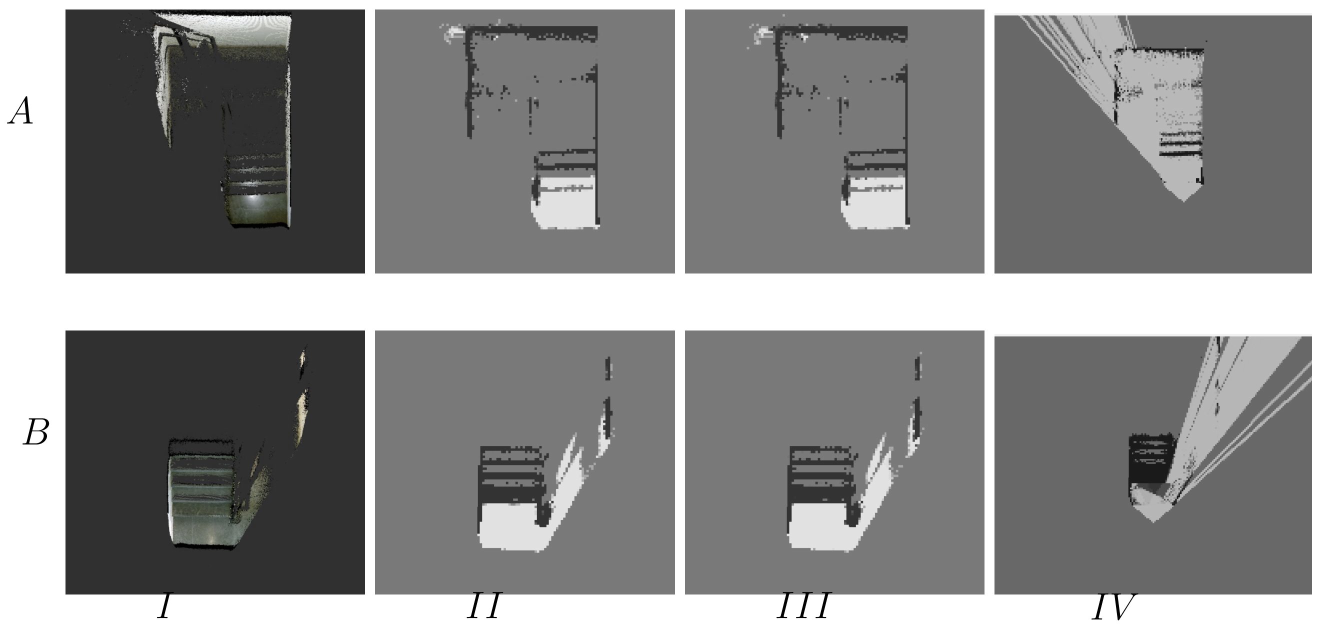


Figure 5.18: Results for the fourth scenario.

We can also verify from Figure 5.19 and 5.20 that the navigation using the available ROS method cannot be allowed in this type of scenario since none of the methods can map stairways because these methods uses 2D sensors and this type of sensors do not provide z-axis information of the surrounding environment.

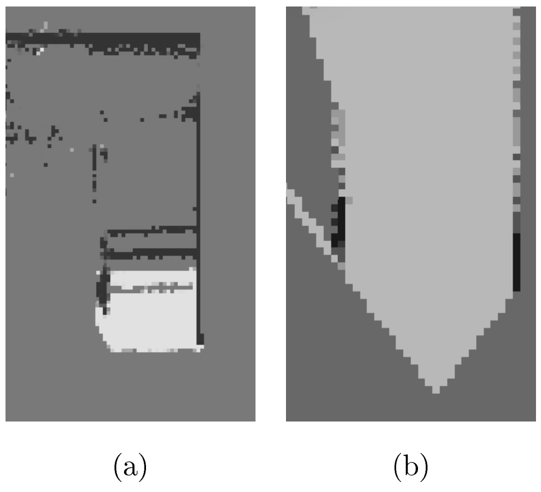


Figure 5.19: Comparison between the proposed method (a) and the available ROS method (b) for downward stairs.

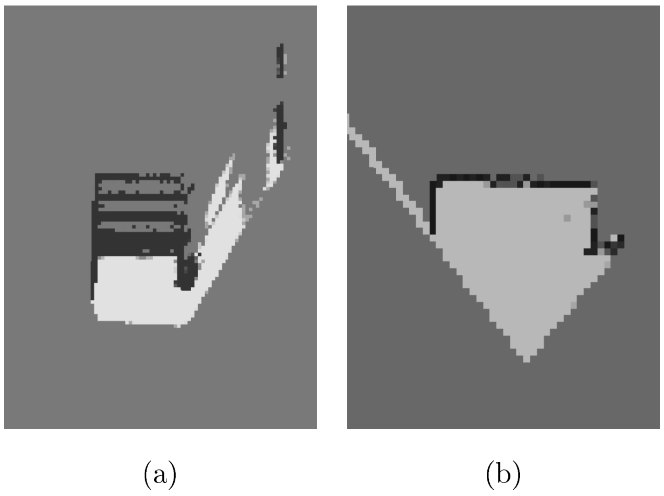


Figure 5.20: Comparison between the proposed method (a) and the available ROS method (b) for upward stairs .

We conclude that in the first scenario only 3D sensors can be used in conjunction with the proposed method since the information collected through the sensors is not discarded, unlike the ROS approach that can only represent the walls correctly. This means that the navigation system will not allow the navigation of the walker in the direction of the stairs, guaranteeing the safety of the user in one of the most dangerous indoor scenarios for the navigation of a mobile robot.

5.3.5 Test Scenario 5

In order to compare the proposed method using the two considered sensors and the solution provided by the ROS community, a last test was conducted in a scenario that includes three floor outlets, represented at the top of the scenario, and some chairs and tables at the bottom of the scenario.

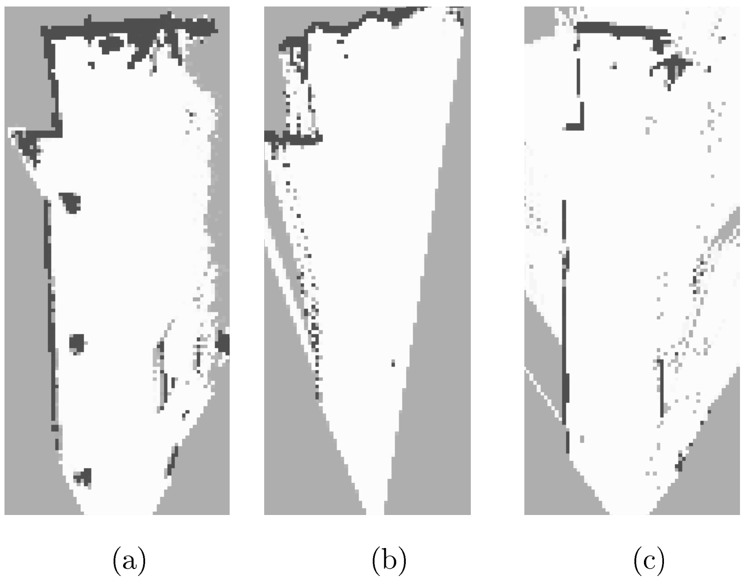


Figure 5.21: Results obtained from experiment 5 with Microsoft Kinect One and the proposed method (a), Leddartech Leddar IS16 (b) and standard ROS package solution using Microsoft Kinect One (c).

The results, represented in 5.21, show that the proposed method using the Kinect One provides a reliable environment representation that is able to detect the general outline of the scenario as well as the three floor outlets, resulting in a valid occupational map. The results using the Leddar IS16 sensor, along with the proposed method, provided a rough outline representation of the scenario, being unable to represent the floor outlets. This was an expected result once the Leddar IS16 is a 2D sensor and for that reason can only detect obstacles that are in the direction of the light emitted beam. However this type of sensor is important in a safety-oriented scenario because it is able to provide a reliable local free space map

5.4 Robot Assisted Navigation

Several experiments were carried out, using the ISRobot2 platform, in order to validate the robot assisted navigation module proposed in this dissertation. The validation of this module is in fact the validation of the complete system since it uses the perception and the navigation module to assist the user to navigate through the chosen scenarios. Static and dynamic scenarios were used to perform these tests, since users of this type of platform are faced with these scenarios on a regular basis. It is important to refer that the user can stop the platform at any time by simply stop giving commands to the platform.

5.4.1 Static environment

Test Scenario 1

In the first scenario considered, the platform was place positioned in the middle of the hall and it was placed an obstacle close to the right wall. The objective of this test was to prove that the robot can avoid the obstacle without colliding with the other side of the hall. The experimental scenario for this second test is showed in the Fig.5.22.

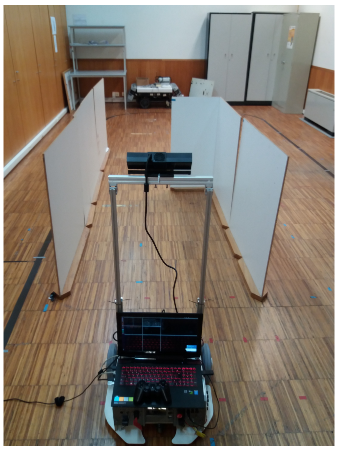


Figure 5.22: Test Scenario for the first test.

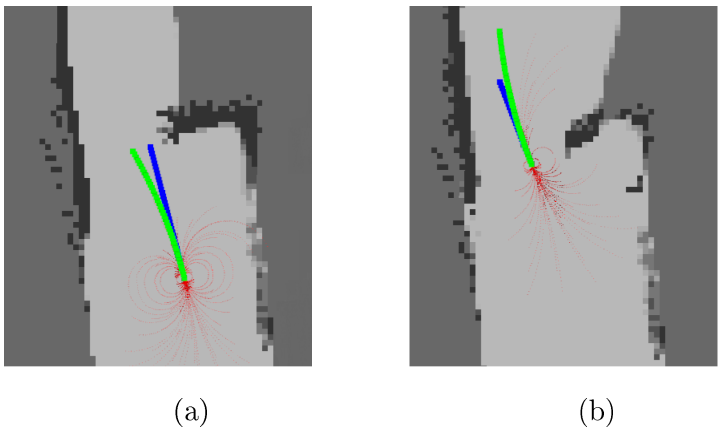


Figure 5.23: Key phases of the first scenario.

From the first image it's possible to understand that the user only gave command to walk straight forward and when the obstacle is detected the robot shows the chosen trajectory to avoid colliding with the obstacle, and then starts to move to this direction. The platform then detects the left wall and readjusts again the trajectory to avoid the collision with the wall. From the second image it is possible to perceive that the user continues to give commands to the platform to move forward and the avoidance was performed by the proposed dwa algorithm, proving in this case that the method successfully avoids collisions.

Test Scenario 2

For the second scenario, the platform was placed in the same place as the previous tests, and one more obstacle was placed. This obstacle was placed on the left side of the scenario, as shown in Fig.5.24, to test the collision avoidance considering two obstacles in opposite sides one after the other and, once again, the user only gave commands to move forward.

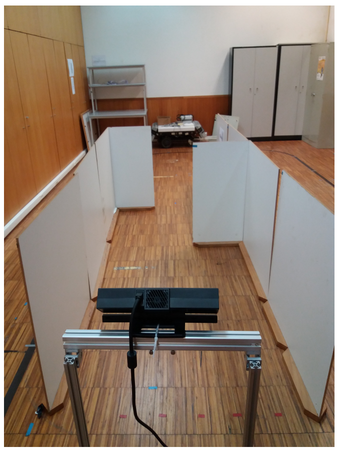


Figure 5.24: Test Scenario for the second test.

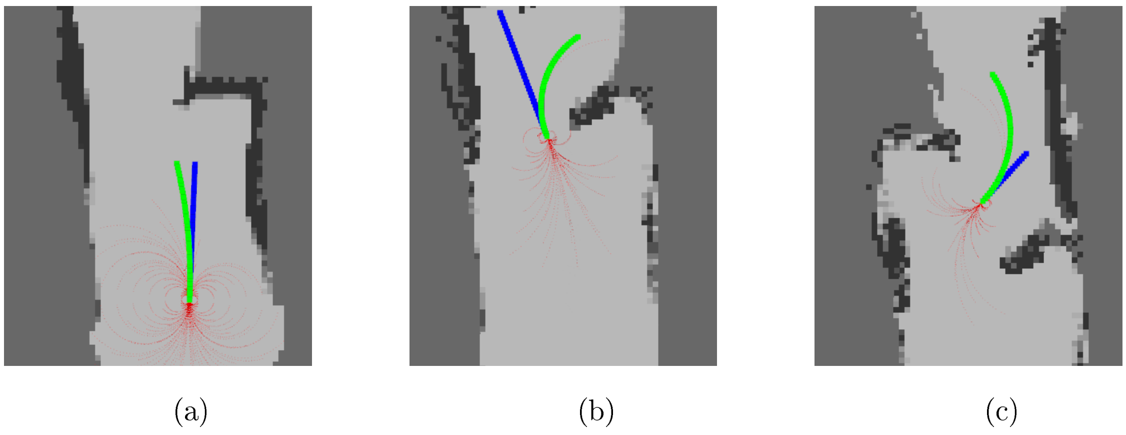
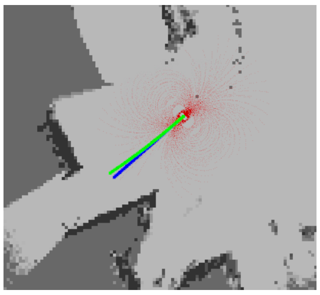


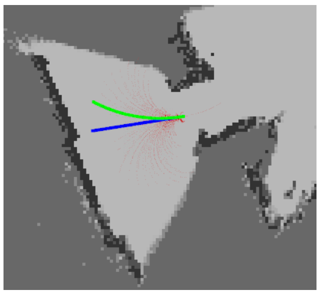
Figure 5.25: Key phases of the second scenario.

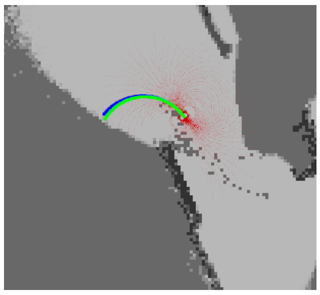
It is possible to understand, form the first image, that the robot detects the first obstacle and chooses the trajectory that avoids the collision, being that as soon as the second obstacle is detected the trajectory changes and the platform successfully avoids the two obstacles, as it can be seen in the second image. The third image shows the moment that the platform detects the right wall and once again recalculates the trajectory avoiding the wall, successfully completing this second scenario.

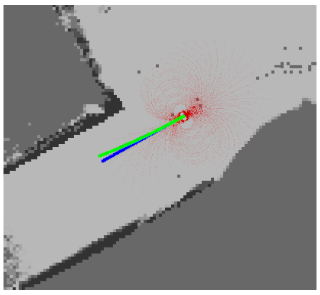
Test Scenario 3

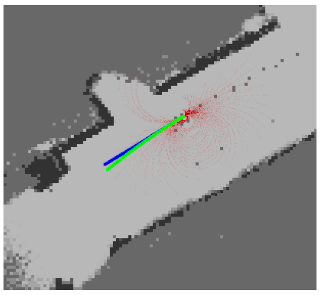
The last scenario was the most complete of them all because this test was conducted on a real scenario where regular obstacles, like garbage cans and potted plants can be found, as well as some windows and different surfaces that can confuse the system. This test was performed on the first floor of the ISR where a path was defined that includes obstacles and 90 degrees curves like the corridor corners and to enter or leave rooms.

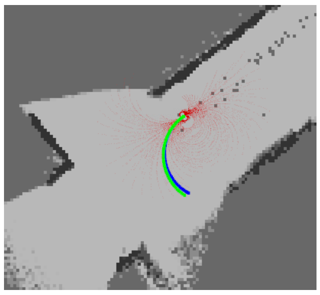

(a)


(b)


(c)


(d)


(e)


(f)

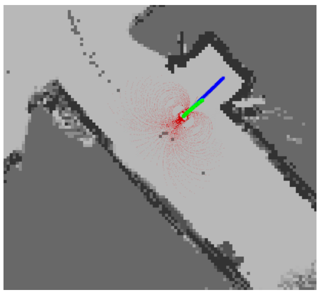

(g)

Figure 5.26: Third scenario map

The test started on the Automation LAB where the platform needs to pass through a doorway followed by a right curve, as shown in Fig. 5.26a and Fig. 5.26b. Right after this curve the user commanded the platform to move to the left, entering in the main corridor of this test. We can see in Fig. 5.26c that the platform doesn't choose a different trajectory from the user because the user chooses one that doesn't have any obstacles, and because of this, the system performed what the user required without any constrains. After this curve the platform was positioned by the user in a trajectory that contained one potted plant and when the system detects the obstacle it corrects the trajectory correctly and avoids the collision, Fig. 5.26e, returning then to the direction that the user intended. The platform then moves until it reaches an intersection, where the user choose, once again, a trajectory that do not contain any obstacles and then stopped in front of a closed door where the platform did not allow the user to move forward because it would lead to a collision.

This test proves that the system was well designed and allows the user to move safely in an everyday environment, correcting trajectories that lead to collisions while allowing free navigation in the remaining cases.

5.4.2 Dynamic environment

The following tests were conducted to assess the system's behaviour in a dynamic environment, considering two cases.

Test Scenario 1

The first scenario consist in a hall with one user moving straight forward on the left side of the scenario and the platform moving on the opposite direction on the right side.

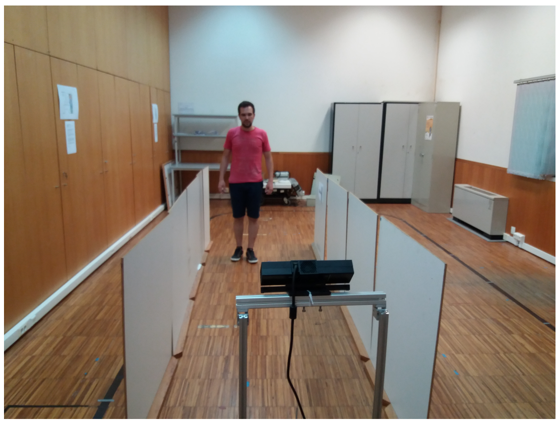


Figure 5.27: Test Scenario for the first test.

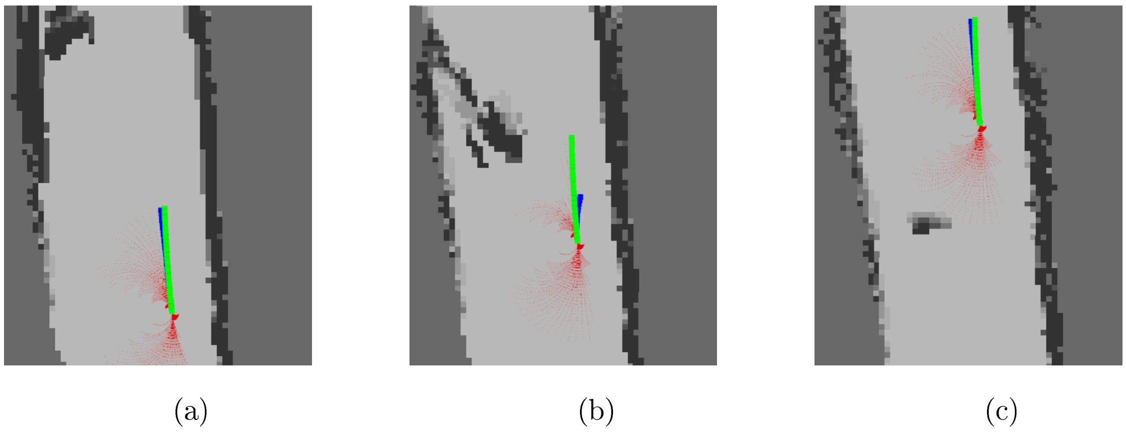


Figure 5.28: Dynamic test scenario map for the first test.

As it can be seen in Fig. 5.28b, the robot start moving and when the person was detected the user pushes the platform to the wall to move away from the user. The system then computes the trajectory and concludes that the current trajectory will not collide with the person or the wall, assuming the control until the person passes though the platform and then return to the command of the user, completing with success the test.

Test Scenario 2

In the second scenario the person moves as in the previous test but then passes to the front of the platform and then stops. As we can see in the Fig. 5.29b the system detects the change in position of the obstacle and recalculates a new trajectory to avoid the collision with the person and then with the left wall, completing this test sucessfully.

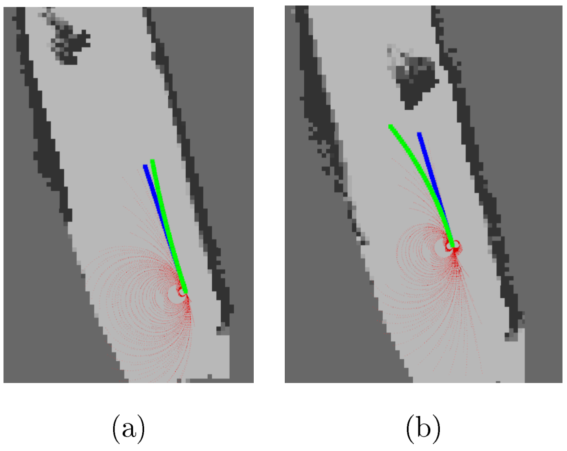


Figure 5.29: Dynamic test scenario map for the second test.

It is important to note that the second scenario is not a usual situation because normally a person will not try to disturb someone with walking disabilities, being that the test will probably fail if a higher speed was used by the person. In this situation the system would not take the time to apply a trajectory that would avoid collision without using high speeds, which is not recommended in this type of platforms. In this case the platform will stop because it is the only way to avoid the collision with the other user, ensuring the safety of all involved parts.

6 Conclusion

6.1 Conclusion

This work was focused on designing modules for the local environment perception and robot-assisted navigation to be applied to the ISR-AIWALKER in order to ensure that it follows all of the user's instructions, while obeying to safety rules, taking into account the knowledge of the surrounding environment. This was a challenge because the type of approach chosen for this dissertation was little explored in terms of assistive robotics, which makes this work a good starting point for future real applications, thus contributing to a future where a patient rehabilitation process will be aided by assistive robots and ensure, at the same time, their safety throughout the process. This assistive system showed good results despite the fact that sometimes some collisions were verified due to the sampling period chosen for the navigation approach. Other drawback on the navigation side is the existence of local minimum, despite of the fact that the user can readjust the walker and continue its navigation. It was verified (see Chapter 5) that the architecture developed for the local environment perception, when compared to other methods, can provide a richer representation and a solution to map non-trivial obstacles with more realistic contour information, contributing to the resolution of some key pitfalls on the handling of walkers. To accomplish the main goal, several sub-goals were attained:

- Study of different methods for the representation of local environments and navigation strategies;
- Research and Development of a local map solution and a moddified dynamic window approach in ROS environment;
- Adaptation of the ISR-AIWALKER for the sensors used during the implementation of this work and conversion of the platform to the ROS environment (driver packages);

- The local environment perception and the robot-assisted navigation were successfully tested proving that the method is robust.

6.2 Future work

The navigation method can be improved to guarantee that collisions are minimized. For this, the complete setup must be rearranged to ensure that there are no loose parts, such as cables. A multisensory approach, by integrating new sensors, can lead to the enhancement of the perception module. The perception module can be enhanced to detect if the user is falling, thus increasing the degree of safety provided by the robotic walker, and finally the overall system should be validated with users suffering from reduced motor skills.

7 Bibliography

[1] Juan David Adarve, Mathias Perrollaz, Alexandros Makris, and Christian Laugier. Computing occupancy grids from multiple sensors using linear opinion pools. *IEEE ICRA*, 2012.

[2] Joydeep Biswas and Manuela M. Veloso. Depth camera based indoor mobile robot localization and navigation. In *ICRA*, pages 1697–1702. IEEE, 2012.

[3] Joydeep Biswas and Manuela M. Veloso. Planar polygon extraction and merging from depth images. In *IROS*, pages 3859–3864. IEEE, 2012.

[4] Johann Borenstein and Yoram Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE transactions on robotics and automation*, 7(3):278–288, 1991.

[5] James Bruce and Manuela Veloso. Real-time randomized path planning for robot navigation. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2383–2388. IEEE, 2002.

[6] David M Cole and Paul M Newman. Using laser range data for 3D slam in outdoor environments. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1556–1563, 2006.

[7] Konstantinos G Derpanis. Overview of the ransac algorithm version 1.2. May 2010.

[8] Ivan Dryanovski, William Morris, and Jizhong Xiao. Multi-volume occupancy grids: An efficient probabilistic 3D mapping model for micro aerial vehicles. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1553–1559. IEEE, 2010.

[9] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, June 1989.

[10] Alberto Elfes. Sonar-based real-world mapping and navigation. *Robotics and Automation, IEEE Journal of*, 3(3):249–265, 1987.

[11] 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), June 1981.

[12] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[13] Luís Garrote, José Rosa, João Paulo, Cristiano Premebida, Paulo Peixoto, and Urbano J Nunes. 3d point cloud downsampling for 2d indoor scene modelling in mobile robotics. In *Autonomous Robot Systems and Competitions (ICARSC), 2017 IEEE International Conference on*, pages 228–233. IEEE, 2017.

[14] Diogo Manuel da Silva Gonçalves. Robchair 2.0: Simultaneous localization and mapping and hardware/software frameworks. Master's thesis, DEEC-FCTUC 2013.

[15] Raia Hadsell, J. Andrew Bagnell, Daniel F. Huber, and Martial Hebert. Accurate rough terrain estimation with space-carving kernels. In *Robotics: Science and Systems*, 2009.

[16] M. Herbert, C. Caillas, E. Krotkov, I. S. Kweon, and T. Kanade. Terrain mapping for a roving planetary explorer. In *Proceedings, 1989 International Conference on Robotics and Automation*, May 1989.

[17] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013.

[18] Urbano J. Nunes João Paulo, Paulo Peixoto. Isr-aiwalker: Robotic walker for intuitive and safe mobility assistance and gait analysis. In *IEEE Transactions on Human-Machine Systems*, 2017.

[19] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[20] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986.

[21] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[22] J.J. Leonard and H.F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *IEEE/RSJ International Workshop on Intelligent Robots and Systems IROS'91.*, pages 1442–1447, 1991.

[23] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Auton. Robots*, 4(4):333–349, 1997.

[24] H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, March 1985.

[25] João Paulo, Paulo Peixoto, and Urbano Nunes. A novel vision-based human-machine interface for a robotic walker framework. In *IEEE RO-MAN, Japan*, 2015.

[26] Cristiano Premebida, Joao Sousa, Luis Garrote, and Urbano Nunes. Polar-grid representation and kriging-based 2.5D interpolation for urban environment modelling. In *IEEE ITSC, Spain*, 2015.

[27] Tahir Rabbani, Frank Van Den Heuvel, and George Vosselmann. Segmentation of point clouds using smoothness constraint. *International archives of photogrammetry, remote sensing and spatial information sciences*, 36(5):248–253, 2006.

[28] Tiana Rakotovao, Julien Mottin, Diego Puschini, and Christian Laugier. Multi-sensor fusion of occupancy grids based on integer arithmetic. *IEEE ICRA*, 2016.

[29] Carolina Raposo, Michel Antunes, and Joao P Barreto. Piecewise-planar stereoscan: structure and motion from plane primitives. In *European Conference on Computer Vision*, pages 48–63. Springer, 2014.

[30] Sebastian Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21 – 71, 1998.

[31] Sebastian Thrun. Learning occupancy grid maps with forward sensor models. *Auton. Robots*, 15(2):111–127, September 2003.

[32] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. Probabilistic robotics, 2005.

[33] R. Triebel, P. Pfaff, and W. Burgard. Multi-level surface maps for outdoor terrain mapping and loop closing. In *IEEE/RSJ IROS*, Oct 2006.

Appendix A

in "IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), Coimbra, 2017".

3D Point Cloud Downsampling for 2D Indoor Scene Modelling in Mobile Robotics

Luís Garrote, José Rosa, João Paulo, Cristiano Premebida, Paulo Peixoto and Urbano Nunes

Abstract—**Sensory perception and environment modelling are important for autonomous navigation in mobile robotics. 2D discrete grid representations such as the classic 2D occupancy grid maps are a widely used technique in scene representation because of the inherent simplicity and compact representation. In recent years, many 2.5D and 3D grid based methods have been proposed however, as for the 2D case, a compromise between keeping a low computational bound and reliable sensor interpretation must be kept in order to perform real-world tasks. Assuming the input data in the form of a 3D point-cloud, in this paper we propose a 2D scene modelling approach which converts the 3D data to a 2.5D representation and then to a 2D grid map in an efficient and meaningful manner. The proposed approach incorporates a new rapidly exploring random tree inspired ground-plane detection (RRT-GPD), and an inverse sensor model (ISM) to correctly map 3D to 2.5D and then to 2D grid cells. Experiments were conducted in indoor scenarios with a robotic walker platform equipped with a Microsoft's Kinect One and a LeddarTech's Leddar IS16 sensor. Reported results show an improvement on the representation of non-trivial obstacles (stairs, floor outlets) over the classical occupancy grid map, when applied to a 3D point cloud input.**

I. Introduction

The development of sensor-based perception systems is one of the main focuses of research in robotics. Considering the increasing introduction of robots in human populated environments, there is an heightened concern to ensure the safety of all agents who may interact with the robot. In particular, when considering the case where the robot is a physical support aid, such as a robotic walker used in gait rehabilitation scenarios [1], the safety of all agents is crucial. The robot must ensure that it follows all the user's instructions accordingly, while obeying safety rules taking into account the knowledge of the surrounding environment.

Despite the recent advances in the area, the creation of robust models for representation of the surrounding environment of the robot is still an open and challenging problem, mostly because of the uncertainty processing of sensory measurements and their correct representation in unpredictable environments. Usually such representation follows an approach based on occupancy maps, which makes it possible to determine, based on the probabilistic models of the used sensors, specific information about the existence of free and occupied space in the surrounding environment. In this work, we review the 2D occupancy grid-map framework, and propose a new approach to the problem of mapping collision-able non-trivial obstacles from 3D point clouds into a 2.5D and then to a 2D environment representation for autonomous navigation. Such obstacles may have can have disastrous consequences for the user if not detected properly (e.g., stairs, gutters or floor outlets). The main contributions are: 1) an inverse sensor model (ISM) for 2.5D to 2D mapping, from 3D data input, incorporating the ground-plane and the concept of voxel density; 2) a new rapidly exploring random tree inspired ground-plane detection (RRT-GPD) algorithm; 3) real time execution ($\approx$ 30 frames per second).

An overview of the related work in environment representation is provided in section II. In Section III, the proposed method is explained. Experiments are carried out in section IV, using a differential mobile robot equipped with a Kinect and a Leddar IS16 sensors, followed by discussions of the reported results. Final conclusions are pointed in section V.

II. Related Work

Environment representation and modelling have been the focus of many research works in mobile robotics [2], [3], with many representations being proposed in the last 30 years. We can categorize scene representations in 3 main classes: direct, topological or grid-based. The approach presented in this paper follows a grid-based representation in which the environment is subdivided into a set of smaller units that form a grid. Based on the unit structure, the grid-based representation can be further extended into 2D cells, 2.5D cells (or voxels) and 3D voxels. In the literature, the 2D grid-map is a well-known representation, providing a probabilistic framework [4], with fast and constant-time access while being usually only applicable in planar environments. In this type of representation each cell contains the probability of occupancy, where values near zero correspond to free cells, values near one occupied cells and a middle value corresponds to an unexplored region. These probabilistic values depend on the successive readings from sensors, increasing or decreasing according to sensor observation models. On the other hand, representations such as 2.5D grid-maps provide a framework to represent elevations and irregular terrain instead of just occupancy, having properties similar to 2D maps but with information regarding

Authors are with Institute of Systems and Robotics, Electrical and Computer Engineering Department, University of Coimbra, Coimbra, Portugal.

This work was supported by the Portuguese Foundation for Science and Technology (FCT) under the PhD grants SFRH/BD/88672/2012 and SFRH/BD/88459/2012 with funds from QREN – POPH and the European Social Fund from the European Union. It was also partially supported by the FCT projects UID048 and AMS-HM12: RECI/EEI-AUT/0181/2012.

Email: {garrote, joserosa, jpaulo, cpremebida, peixoto, urbano}@isr.uc.pt

978-1-5090-6234-8/17/$31.00 ©2017 IEEE

the height of each cell, and a computationally lighter model than 3D grid-maps. However, a 2.5D representation can not fully represent vertical overlapping. Solutions to this problem have been proposed in the form of multi-layer maps. These grid-map approaches are easy to build, represent, maintain and facilitate computation of shortest paths but, on the other hand, they present downsides such as being memory consuming for large environments and presenting poor interfaces with most classic planning algorithms. More recently, 3D grid-maps, and in particular solutions such as octomaps [3], provided a reliable 3D representation at the expense of variable access time, increased computational complexity and increased planning complexity for ground robots. In spite of the constant evolution, 2D grid-map is still an usefull and somehow efficient representation for indoor and outdoor scenarios but, due to its own limitations, fails to incorporate vertical elements that are in the robot's pathway, even if they can be trivially spotted by humans. A solution to this problem could be the use of a ground reference and/or a mapping strategy in order to distinguish between obstacles and drivable space. The ground reference could be provided by elevation tresholds or by a ground-plane detection algorithm. In [5], [6] 2D grid-maps are created from 3D point clouds and stereo images where a fusion model incorporates vertical penalization based on elevation thresholds. In [7] a 3D voxel representation is created from stereo images and converted to 2D grid-maps using a voxel observation model. Some solutions available in the Robot Operating System (ROS) provide a medium of converting 3D point clouds into 2D laser scan messages using elevation thresholds and can be trivially converted into 2D grid-maps. In [8], 2D grid-maps are computed from stereo sequences using a intermediary 2.5D representation to generate elevation thresholds. Solutions for multiple plane detection [9] have been proposed in indoor scenarios from depth images, which can be adapted for ground-plane detection. For ground-plane detection in point clouds, region growing methods [10] can provide good results, but their computational cost becomes prohibitive in denser point clouds.

In this paper, we propose an approach for 2D scene representation and modelling, from a 3D point-cloud input, to allow autonomous navigation in real-world (indoor) conditions. The approach takes advantage of 2.5D representation to detect non-trivial obstacles. Moreover, a ground detection solution, using a RRT algorithm, is also addressed.

III. PROPOSED METHOD

In this section we detail the proposed method and its functional modules. The key motivation here is to develop a reliable and computationally efficient method for mapping 3D point clouds into a 2D grid-map and consequently, the correct mapping of collision-able non-trivial obstacles. Such obstacles, are defined in the context of this work as obstacles that are in a given robot's pathway but are but are neither trivially detected nor efficiently mapped by common 2D mapping approaches (e.g., stairs, small boxes or electric wiring). The proposed method depends on a 2.5D representation, followed by the

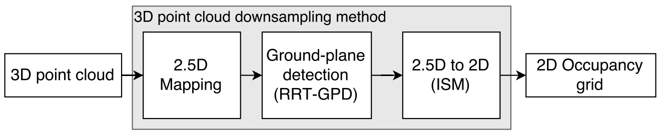

3D point cloud downsampling method
3D point cloud
2.5D Mapping
Ground-plane detection (RRT-GPD)
2.5D to 2D (ISM)
2D Occupancy grid

Fig. 1. Depiction of the proposed method including the modules: 2.5D mapping; ground-plane detection; 2.5D to 2D conversion.

ground-plane detection and then the final (enhanced) 2D grid-map is obtained using a 2.5-to-2D ISM.

A general overview of the proposed method is presented in Fig. 1 which includes 2.5D mapping, ground-plane detection and 2.5D to 2D conversion.

A. 2.5D Mapping

The first module builds on the construction of a 2.5D environment representation from a 3D point cloud [11]. The 2.5D representation discards the concept of occupancy and provides an elevation measure for each cell. The proposed 2.5D grid is composed by m_x x m_y cells with constant resolution d_r. Each cell $c_{2.5D}$ is addressed by i and j indexes and has the elements $c_{2.5D} \leftarrow \{z^-, \ z^+, N_z\}$ where z^- is the minimum elevation value, z^+ the maximum elevation value and N_z the number of samples that contribute with information to the cell. Given a 3D point cloud composed by a set of 3D Cartesian points ($\mathbf{p}_k = (x_k, y_k, z_k)^T$, $k = 1, 2, ..., n$) the values present in each cell are obtained by projecting the x and y components on the grid cell with consequent update of the maximum and minimum elevation (z^+ and z^-) and the number of samples ($\|z\|$) projected to the given cell. The projection consists on the conversion of the x and y component from $\mathbf{p}_k$ into grid indexes i, j with $i \leq m_x$, $j \leq m_y$, $m_x, m_y > 0$. The 3D to 2.5D maps conversion process is illustrated by an example in Fig. 2.

B. Rapidly exploring random tree based ground-plane detection (RRT-GPD)

One problem when converting an elevation map into an occupancy grid is the definition of free and occupied 2.5D cells. Given a 2.5D grid, the definition of what is an obstacle or what is navigable space increases in difficulty because there is no reference to the location of the ground plane. For a given frame, we can assume that a robot moves locally in a 2D plane, and based on this constraint we propose a new approach to the problem of ground-plane detection that operates with a 2.5D map and is inspired by the rapidly-exploring random tree (RRT) algorithm, which has been widely applied in motion planning [12] and region growing [10], in the sense that it implements a rapid flood-like expansion towards similar normals.

The inputs of the proposed RRT-GPD method include, the robot's pose, the plane's normal threshold, elevation threshold, the 2.5D environment model, the maximum number of iterations to find a suitable solution, and a node expansion distance. The RRT-GPD (presented in Algorithm 1) starts by finding a valid seed to form the root of the RRT (**nearestValidSeed**).

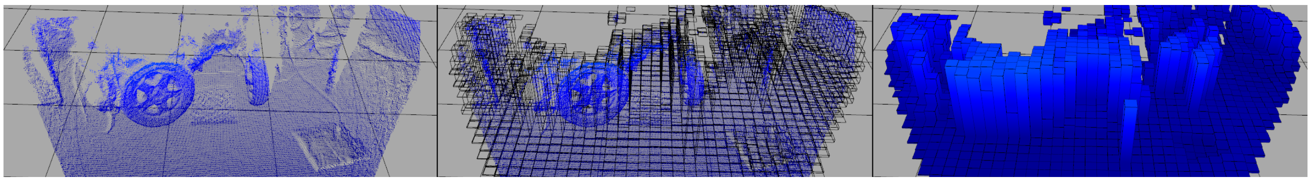


Fig. 2. Representation of the conversion from 3D point clouds to 2.5D environment representation. From left to right, the input 3D point cloud (given by a Microsoft's Kinect One), the projection step where each point in the point cloud is projected into the corresponding cell, and the final result where each cell resembles a voxel defined by the maximum and minimum elements projected onto the cell.

The use of an invalid seed may create a ill formed tree. This is due to the sparse nature of the 2.5D representation (see Fig. 2) meaning that not all cells of the provided 2.5D map $M_{2.5D}$ will contain information. Besides, a seed far from a valid cell may be unable to expand, a problem which can be solved using a variable node expansion distance. In the process of finding a valid seed, the robot's pose provides an initial guess, and since no assumptions are made on the sensors' configuration (i.e., transformation between sensor and robot) every direction is explored to find a valid candidate. Although it is important to find a valid candidate to initialize exploration, the final solution may discard the initial seed point (i.e., it is not guaranteed that the final solution will contain the initial point). This description is based on the assumption that the 2.5D map is computed with the sensor referenced on the robot's frame and that exists a relation between the robot and the environment (transformation between robot and world frame) but for local maps (mobile robot centered in $\mathbf{p}_{xyz} = (0,0,0)^T$) or for a representation in the sensor's referential the constraints are similar. The concept of valid seed implies in this particular case, a valid local plane computed using **fitPlane**. After finding a valid seed and a correspondent local plane, a new tree is generated in **initialiseTree** where the seed, which corresponds to the center of the plane, and the plane normal define the base unit (or node) of the RRT tree. If a valid node is created, the algorithm enters an iterative process to expand the RRT until it reaches K iterations. For each iteration the first step involves sampling (**sampleRandomDirection**) a point x_{rand} in a search space window. For a given sample node x_{rand}, the nearest node x_{near} already present in the tree is retrieved (**nearestNode**) and a new node $x_{expansion}$ is created based on the direction between x_{near} and x_{rand} (provided that they do not overlap). The direction is given by (**angleBetween**),

$$\theta = \arctan\left(\frac{x_{rand}(y) - x_{near}(y)}{x_{rand}(x) - x_{near}(x)}\right) \tag{1}$$

and the (x, y) component is given by,

$$x_{expansion}(x,y) = \begin{cases} x_{near}(x) + d_n \cos(\theta) \\ x_{near}(y) + d_n \sin(\theta) \end{cases} \tag{2}$$

where d_n denotes the node expansion distance. For the new candidate node $x_{expansion}$, the z component is given by the procedure **elevation**. As stated earlier, given the sparse nature of the 2.5D grid, not every $x_{expansion}$ will correspond to a valid cell when projected on 2.5D. To provide a valid z component we employ a spatial interpolation method on the node neighborhood (a region of interest with dimensions (s_x,s_y) centered in $x_{expansion}$) and if an invalid z value is retrieved (no valid neighbors) the current iteration is ended. With a valid $x_{expansion}$, and applying a similar spatial interpolation method, a set of neighborhood points P is extracted (**interpolation**) where empty points missing the z component are interpolated and discarded if the neighborhood does not contain valid elements. If the neighborhood P contains at least three non-collinear points a least squares regression is performed to find the best planar fit to the points (**fitPlane**) of the form $ax+by+cz+d=0$. Given a valid plane $plane_{local}$, the inner product of the normal stored in x_{near} and the planes' normal (**normal**($plane_{local}$)) is computed and if it is less than a given threshold (N_{th}) the plane is considered to be at least similar in orientation given the node x_{near}. In order to analyze whether the new plane can connect or not with the nearest node, the variation in height is also checked (using the elevation threshold (E_{th})). If the normals are not similar, the nodes near a radius d_n from $x_{expansion}$ are retrieved (**nearestNodes**) and the same similarity thresholds are applied to validate further node connections. If a pair plane-node is conformant (i.e., is valid for each threshold) a new node is added to the tree, containing the parent connection (x_{near}), the new center point (computed plane midpoint) and the correspondent plane normal. The last step after adding a node to the tree is to update the search space window (**updateSearchSpace**). The search space window starts centered on the seed node but with each added node and each iteration, the search window grows and shifts towards the average value (geometric center) of the explored nodes, slightly biasing the search process to areas with similar properties where the expansion is more prominent, but without leaving out unexplored areas. After K iterations, the center points of each node on the RRT tree are extracted (a Kd-tree is used at the algorithm's core to store each discovered node) and a new plane fitting is performed. In this final step, we apply the random sample consensus (RANSAC) algorithm [13] due to its robust estimation even in the presence of outliers. The RRT-GPD algorithm is summarized in the pseudocode of Algorithm 1.

C. 2.5D to 2D conversion

The conversion from 2.5D to 2D follows the same principles introduced with the integration of sensor readings for occupancy grid mapping [14], [4]. In this case, our observations

Algorithm 1: Rapidly exploring random tree based ground-plane detection (RRT-GPD) algorithm.

```
Input: Robot pose ($\mathbf{p}_{xyz}$), Plane normal threshold ($N_{th}$) ,
       Elevation threshold ($E_{th}$), 2.5D Map ($M_{2.5D}$) ,
       Maximum number of iterations ($K$) , Node expansion
       distance ($d_n$)
1  Initialization:
2  $\mathbf{p}_0 \leftarrow$ nearestValidSeed($\mathbf{p}_{xyz}$, $M_{2.5D}$);
3  $G \leftarrow$ initializeTree($\mathbf{p}_0$);
4  for $k$=1 to $K$ do
5      $x_{rand} \leftarrow$ sampleRandomDirection();
6      $x_{near} \leftarrow$ nearestNode($G$,$x_{rand}$);
7      $\theta \leftarrow$ angleBetween($x_{near}$,$x_{rand}$);
8      $x_{expansion}(x) \leftarrow x_{near}(x) + d_n \cos(\theta)$;
9      $x_{expansion}(y) \leftarrow x_{near}(y) + d_n \sin(\theta)$;
10     $x_{expansion}(z) \leftarrow$ elevation($M_{2.5D}$, $x_{expansion}$) ;
11     $P \leftarrow$ interpolation($M_{2.5D}$, $x_{expansion}$);
12     $plane_{local} \leftarrow$ fitPlane($P$);
13     if | normal($plane_{local}$) · normal($x_{near}$) | $\leq N_{th}$ then
14         if | $plane_{local}(z)$ - $x_{near}(z)$ | $\leq E_{th}$ then
15             $G \leftarrow G \bigcup$ { $x_{near}$,$x_{expansion}$,
                 normal($plane_{local}$) };
16             updateSearchSpace($x_{expansion}$);
17     else
18         $x_{neighbours} \leftarrow$ nearestNodes($G$,$x_{expansion}$,$d_n$);
19         foreach node in $x_{neighbours}$ do
20             if | normal($plane_{local}$) · normal(node) | $\leq N_{th}$
               then
21                 if | $plane_{local}(z)$ - node($z$) | $\leq E_{th}$ then
22                     $G \leftarrow G \bigcup$ { node,$x_{expansion}$,
                         normal($plane_{local}$) };
23                     updateSearchSpace($x_{expansion}$);
24 $g_{plane} \leftarrow RANSAC(Points(G))$
   Output: $g_{plane}$
```

are the elevation voxels present in the 2.5D grid-map, turning the representation into a virtual sensor. The probability that a cell c is occupied given the observations $z_{1:t}$ is given in log odds by:

$$l(c|z_{1:t}) = \log \frac{p(c|z_t)}{1-p(c|z_t)} \underbrace{-\log \frac{p(c)}{1-p(c)}}_{=0, \text{ if } p(c)\,=\,0.5} +\log \frac{p(c|z_{1:t-1})}{1-p(c|z_{1:t-1})} \tag{3}$$

with $p(c)$ the prior probability, $p(c|z_{1:t-1})$ the previous estimate and $p(c|z_t)$ denotes the probability that cell c be occupied given the measurement z and it is computed using an ISM. The log odds representation is used here due to its numerical stability.

To solve the 2.5D to 2D conversion problem we propose an ISM that converts an observation in the form of an elevation voxel c_v to the probability that given the actual observation, the cell from the 2D grid is occupied $p(c|z_t)$. Each elevation voxel can be defined as being in a valid state if it contains more than one measurement ($N_z \geq 1$). In order to determine the influence of each voxel we rely on the concept of voxel density explored in [15]. An elevation voxel c_v in the 2.5D map (see Fig. 2) occupies the volume given by $V_{voxel} = hA$, with h is the height of the voxel ($h = \Delta z = z^+ - z^-$) and A the base area of the voxel (i.e., based on the cell resolution). The voxel density is given by $\rho_{voxel} = \frac{m}{V_{voxel}}$, where m is the voxel mass. The mass of a voxel in this context can be defined as the amount of data the voxel contains and can be extrapolated using the number of samples N_z of an elevation voxel. To represent a normalized voxel density, the following sigmoidal function is proposed,

$$\rho_{voxel}(c_v) = \frac{K_m}{1+e^{-(\frac{K_n(c_v(N_z)-d_{min})}{V_{voxel}})}} \tag{4}$$

where K_m denotes an amplitude gain, K_n a sample normalization factor, and d_{min} the minimum number of points.

The voxel is composed by 3 explicit parameters and an implicit one related to the distance $|c_v|$ defined by the voxel position in relation to the base frame (e.g., sensor frame, robot frame or local frame). In the previous subsection we presented the RRT-GPD method to extract the ground plane that from now on is denoted by g_{plane}.

1) Inverse sensor model: The proposed ISM takes into account the voxel distance to the base frame, decreasing the elevation voxel occupancy probability as voxels move away from the base frame. Knowing a valid ground-plane allows for the definition of "free" or "occupied" values in the sense that a specific cell contains a high or low probability of being occupied, for instance, if a voxel is near the detected ground-plane it may be considered as part of the ground and thus contribute to decrease the cell's probability. On the other hand, if an elevation voxel is above the ground, it may be considered to be an obstacle and thus contributes to increase the cell's probability. The ISM is mathematically expressed by,

- If $|c| \;\epsilon\; [\ 0,\ |c_v|\ ]$:

$$p(c|z_t) = \begin{cases} \max(\rho_{voxel}, 0.5)\; e^{-\frac{(|c|-|c_v|)^2}{2\sigma^2}} & ,if\ d > d_{pth} \\ K_g + \frac{0.5-K_g}{1+e^{-(|c|-|c_v|)}} & ,if\ d \leq d_{pth} \end{cases} \tag{5}$$

- If $|c| \;\epsilon\;]\ |c_v|,\ |c_v|^{max}\ ]$:

$$p(c|z_t) = \begin{cases} \max(\rho_{voxel}\; e^{-\frac{(|c|-|c_v|)^2}{2\sigma^2}}, 0.5) & ,if\ d > d_{pth} \\ 0.5 & ,if\ d \leq d_{pth} \end{cases} \tag{6}$$

where σ^2 denotes the Gaussian variance, K_g an amplitude and bias gain with $0 \leq K_g \leq 0.5$, d_{pth} a distance threshold and d the distance between the plane g_{plane} and the voxel c_v. The distance between the voxel c_v and the plane is computed using the maximum plane distance to the points $\mathbf{p}^+ = (x, y, z^+)^T$ and $\mathbf{p}^- = (x, y, z^-)^T$ where x and y represent the voxel position. The 2D grid-map structure is identical to the 2.5D counterpart introduced in III-A with each cell c_{2D} composed only by an occupancy value.

IV. EXPERIMENTAL RESULTS

Two experiments, using the mobile robot shown in Fig. 3, were carried out with the purpose of validating the proposed method. The first experiment consisted on the qualitative

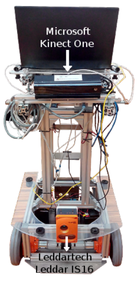


Fig. 3. The ISR-AIWALKER experimental platform.

TABLE I
LIST OF PARAMETERS

Variable	Value	Variable	Value
K	600	d_{pth}	0.03
N_{th}	0.2	K_g	0.3
E_{th}	0.05	σ^2	0.02
s_x,s_y	3	K_n	0.0005
m_x,m_y	200	d_{min}	3
d_r	0.05	K_m	0.8
d_n	0.4		

comparison of the generated occupancy map on two static indoor scenarios with a Microsoft's Kinect One sensor mounted onboard the robot. The second experiment consisted on the comparison of the output from the proposed method in the Kinect One sensor, a local-map approach using the Leddar IS16 and a solution based on standard ROS packages in a local map framework, using the Kinect One sensor.

A. Experimental Setup and Implementation details

Our experimental setup is a walker platform (ISR-AIWALKER, see Fig. 3). The walker's base is a differential drive mobile robot and contains two grips interfaced with Leap Motion sensors and a gait perception module aided by a 3D sensor [1]. The robot platform is equipped with two sensors (a Kinect One and a Leddar IS16) for environment perception, assessment of hazardous situations and safety purposes. The Kinect One outputs a 512 x 424 point cloud and the Leddar IS16 delivers a 16-channel distance array. Throughout the experimental evaluation of the method, the detection and removal of outliers was an important step (as illustrated in Fig. 4). The detection and removal of outlier points belonging to the point cloud were performed at the 2.5D elevation voxel map. For a given voxel, the 3σ method is applied to the set of projected elevations and then all voxel elements are recomputed. The spatial interpolation approach applied in this work on the 2.5D map, for candidate plane computation, is the inverse distance weighting (IDW). Also the IDW algorithm is applied to interpolate the 2.5D representation in order to obtain a more dense representation to qualitatively compare the obtained results. The IDW is applied separately in the absence of points (invalid elevation voxel) to z^- and z^+ and the sample count for those generated nodes is the number of neighbour voxels which contributed to the IDW computation. The ISR-AIWALKER perception modules run in ROS nodes, and all experiments reported in this section were carried out in the same environment with C++ implementations. Also, visualization of results were provided by a in-house QT/C++ application. The experiments were carried out in a mid-range laptop with Kinect and Leddar data acquisition frequencies of 10Hz, the average time per frame for the proposed method was less than 30 ms, and the ground-plane detection less than 3 ms. The correspondent parameter values defined in this work are presented in Table I.

B. Results and Discussion

Figure 4 shows two indoor scenarios, an uncovered floor outlet and downward stairs, that were considered in the first experiment. For each scenario, qualitative results are shown in Fig.4 where column I gives the input raw data and column II shows the 2.5D. The 2.5D grid-map was generated without any preprocessing or outlier removal. As it is noticeable on the first scenario, only a reduced number of light-blue elevation voxels correspond to erroneously generated voxels due to noise. The 2.5D to 2D map conversion was performed and the ground-plane was correctly detected, as well as the floor gutter, but the correspondent outlier voxels were mapped (column III) and produced a non-traversable map (i.e. on the context of motion planning and considering only this map, the robot would be unable to move). Applying the 3σ method to the 2.5D representation (column IV) yields a cleaner 2D representation with only a small portion of noise at the end of the traversable path. The following result (column V) was generated directly from the input point cloud for a z-axis layer (0.2 to 1.5 m) using the sensor model described in [5]. It is noticeable that the generated 2D representation does not contain the same amount of occupancy information. Finally, in the result shown in column VI, the 3σ and IDW were applied to the 2.5D representation which led to a dense representation, but with inflated cells. By comparing the columns III, IV, and VI, for both scenarios, the output generated with outlier removal achieves the most satisfactory results, as it correctly maps the floor outlet and detects the portion of floor before the stairs.

The second experiment, with corresponding results shown in Fig. 5, presents three 600 x 600 2D local maps: the proposed method with outlier removal, the classic sensor beam model using the Leddar scan as input and a standard ROS package solution (Kinect One acquisition $\rightarrow$ point cloud to laser scan[1] $\rightarrow$ local map framework[2]). The scenario used for this experiment included three floor outlets (on the left side of the scenario), office chairs and tables (on the right side of the scenario). Results show that the local map from Kinect One, along with the proposed method, provided a reliable environment model by detecting the general scenario outline and the considered hazards (correctly mapped three floor outlets). As expected, the standard ROS solution provided only an outline of the scenario since the point cloud was converted to a laser-like scan and important points relevant to the detection of near ground objects were discarded. The Leddar local map provided a rough outline representation of the scenario. A comparison between the methods applied to both sensors becomes somehow unfair since the sensors have different fields of view and the quantity/quality of information is very distinct. However, in a safety-oriented scenario, the Microsoft's Kinect One, when compared to the Leddar, has

[1]pointcloud_to_laserscan Package – http://wiki.ros.org/pointcloud_to_laserscan
[2]local_map Package – http://wiki.ros.org/local_map

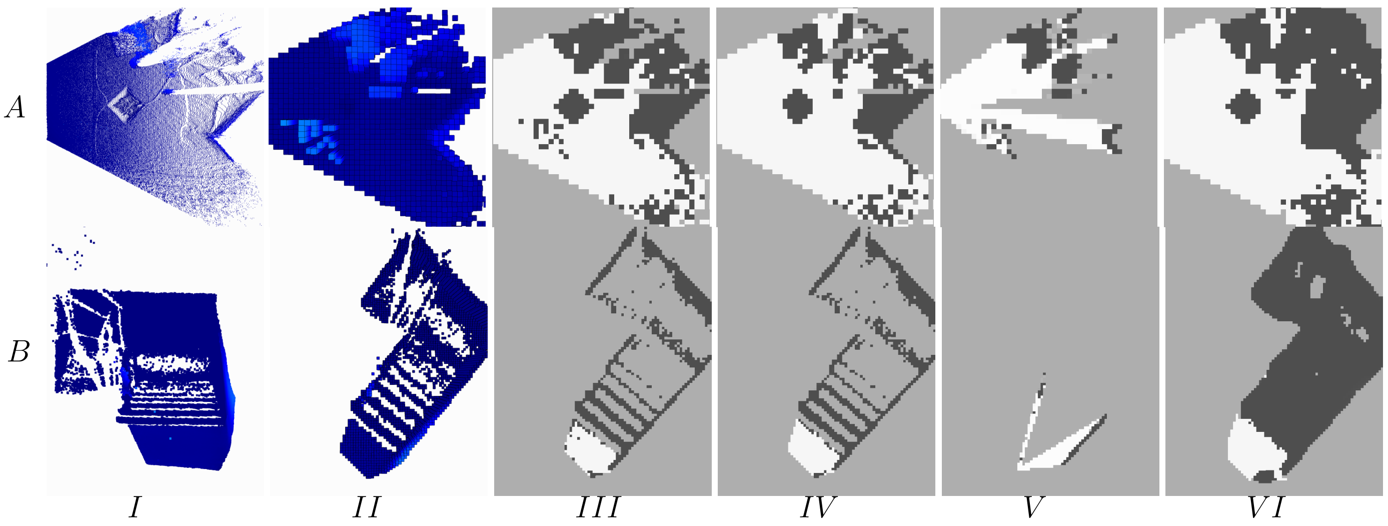


Fig. 4. Results obtained from experiment 1: first row corresponds to a scenario with a uncovered floor outlet (A), second row to a scenario with downward stairs (B). From left to right - point cloud (I), 2.5 representation with outliers (II), 2D representation with outliers (III), 2D representation without outliers (IV), 2D representation from mid-level point cloud (V) and 2D representation interpolated without outliers (VI).

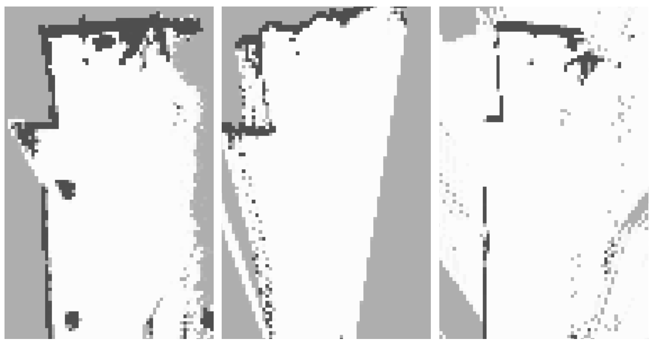


Fig. 5. Results obtained from experiment 2 (from left to right): office traversal with Kinect One using the proposed method, Leddar IS16 using local map and standard ROS package solution using Kinect.

a minimum depth distance of 0.5 m which can become problematic for close obstacles, while on the other hand, the Leddar, while lacking much of the precision of the Kinect One, provides a reliable local free space mapping.

V. Conclusion

In this paper, a novel approach for 2D environment modelling, that maps 3D input data into an enhanced 2.5D-to-2D grid map, is proposed. The approach encompasses a new RRT-based ground-plane detection algorithm (RRT-GPD) and a new way to model the concept of elevation voxel density. We demonstrated in our work that the proposed method, when compared to the classic method adapted to point clouds, can provide a richer representation and a solution to map non-trivial obstacles with more realistic contour information. We also applied the proposed algorithm in a walker-assisted scenario, successfully validating the proposed algorithm in scenarios that are key pitfalls on the handling of walkers. For future work, we plan to research on a multisensory approach for local mapping, incorporating the Microsoft's Kinect One and the Leddartech Leddar sensor, validating the framework in terms of safe navigation in mobile robotics applications.

References

[1] J. Paulo, P. Peixoto, and U. Nunes, "A novel vision-based human-machine interface for a robotic walker framework," in *IEEE RO-MAN*, 2015.

[2] R. Triebel, P. Pfaff, and W. Burgard, "Multi-level surface maps for outdoor terrain mapping and loop closing," in *IEEE/RSJ IROS*, Oct 2006.

[3] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013.

[4] S. Thrun, W. Burgard, and D. Fox, "Probabilistic robotics," 2005.

[5] J. D. Adarve, M. Perrollaz, A. Makris, and C. Laugier, "Computing occupancy grids from multiple sensors using linear opinion pools," *IEEE ICRA*, 2012.

[6] T. Rakotovao, J. Mottin, D. Puschini, and C. Laugier, "Multi-sensor fusion of occupancy grids based on integer arithmetic," *IEEE ICRA*, 2016.

[7] H. Ghazouani, M. Tagina, and R. Zapata, "Robot Navigation Map Building Using Stereo Vision Based 3D Occupancy Grid," *Journal of Artificial Intelligence: Theory and Application*, vol. 1, no. 3, 2011.

[8] H. Lategahn, W. Derendarz, T. Graf, B. Kitt, and J. Effertz, "Occupancy grid computation from dense stereo and sparse structure and motion points for automotive applications," in *IEEE IV*, 2010.

[9] J. Biswas and M. Veloso, "Planar polygon extraction and merging from depth images," in *IEEE/RSJ IROS*, Oct 2012.

[10] T. Rabbani, F. A. van den Heuvel, and G. Vosselmann, "Segmentation of point clouds using smoothness constraint," in *IEVM06*, 2006.

[11] C. Premebida, J. Sousa, L. Garrote, and U. Nunes, "Polar-grid representation and kriging-based 2.5d interpolation for urban environment modelling," in *IEEE ITSC*, 2015.

[12] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Iowa State University, Tech. Rep. 98-11, Oct 1998.

[13] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, Jun. 1981.

[14] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *IEEE ICRA*, 1985.

[15] I. Dryanovski, W. Morris, and J. Xiao, "Multi-volume occupancy grids: An efficient probabilistic 3d mapping model for micro aerial vehicles," in *IEEE/RSJ IROS*, Oct 2010.