UNIVERSIDADE D
COIMBRA

Samuel António Ferreira Temporão Alves

Particle-Filter based 3D Mapping, Localization and SLAM for Indoor Mobile Robot Navigation

Dissertation submitted to the Department of Electrical and Computer Engineering
of the Faculty of Science and Technology of the University of Coimbra
in partial fulfillment of the requirements for the Degree of Master of Science.

September 2019

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

Samuel António Ferreira Temporão Alves

Particle-Filter based 3D Mapping, Localization and SLAM for Indoor Mobile Robot Navigation

Supervisor:

Prof. Dr. Urbano José Carreira Nunes

Co-Supervisor:

Master Luís Carlos Artur da Silva Garrote

Jury:

Prof. Dr. Urbano José Carreira Nunes

Prof. Dr. António Paulo Mendes Breda Dias Coimbra

Prof. Dr. David Bina Siassipour Portugal

A dissertation submitted in partial satisfaction of the requirements for the degree of Master of Science in Electrical and Computer Engineering

Coimbra, 2019

Acknowledgements

This dissertation could not have been completed without the orientation and assistance of many people to whom I would like to express my appreciation and gratitude. I would like to thank my advisor, Professor Doutor Urbano Nunes, for allowing me the opportunity to work on a subject of my interest, for providing all the material, knowledge and suggestions essential during the completion of this work.

A special mention to Luis Garrote, for always providing me with great advice and allowing me to take advantage of his knowledge in this field, always being available to debate any issue.

I am grateful to the researchers from the projects UID/EEA/00048/2019, AGVPOSYS (CENTRO-01-0247-FEDER-003503) and MATIS (CENTRO-01-0145-FEDER-000014), with FEDER funding, programs PT2020 and CENTRO2020, for all the assistance they gave me to complete my dissertation, and also to Institute of Systems and Robotics-University of Coimbra (ISR-UC) for providing me with the conditions and resources to make this work possible.

I am very grateful to my girlfriend for all the support and for always believing in me. To all my friends a thank you for all the moments that we had during the academic journey and that I will keep those memories always with me.

Finally, I am deeply grateful to my family for their support, but especially to my parents for always providing everything necessary for my graduation and to my brother for being a faithful course-mate.

Abstract

Over the years, there has been a massive development in robotic systems. One of the main features of autonomy is the ability of the mobile robot to move without limitations in the environment, without risking collisions. The autonomous behaviour of a mobile robot can be described by the connection between three modules: mapping, localization and path planning. Mapping builds a map of the environment, localization consists of estimating the robot pose in a map, and path planning computes safe paths. The localization and mapping can be used together, resulting in a Simultaneous Localization and Mapping (SLAM) technique. In this dissertation, the focus was on the mapping and localization modules.

The objective of this dissertation was to develop a localization and mapping approach in indoor environments for the platform "InterBot-Social Robot". This approach included the fusion of 2D and 3D sensory data applied to a particle filter to estimate the position of the robot and the construction/update of a 3D map using a 3D point cloud. The development of this approach resulted in two more approaches (2.5D Mapping and Localization, and 3D Simultaneous Localization and Mapping).

Experimental tests were conducted to evaluate the performance of the developed approaches. The tests consisted in verifying the influence of the number of particles on the filter: on pose estimation, the generated map and the localization score. Test results were analyzed, with the final outcomes meeting the expectations set for the approaches.

Keywords: Mobile Robot, Mapping, Localization, SLAM, Particle filter.

Resumo

Ao longo dos anos, tem havido um enorme desenvolvimento em sistemas robóticos. Uma das principais características da autonomia é a capacidade do robô se mover sem limitações no meio ambiente, sem risco de colisão. O comportamento autónomo de um robô móvel pode ser descrito pela relação entre três módulos: mapeamento, localização e planeador de caminhos. O mapeamento constrói um mapa do ambiente, a localização consiste em estimar a pose do robô no mapa e o planeador para determinar caminhos seguros. A localização e o mapeamento quando usados de forma conjunta, resulta numa técnica de Simultaneous Localization and Mapping (SLAM). Nesta dissertação, o foco foram os módulos de mapeamento e localização.

O objectivo desta dissertação foi desenvolver uma abordagem de localização e mapeamento em ambientes *indoor* para a plataforma "InterBot-Social Robot". Esta abordagem incluiu a fusão de informação sensorial 2D e 3D aplicados a um filtro de partículas para estimar a posição do robô e a construção/actualização de um mapa 3D usando uma nuvem de pontos 3D. O desenvolvimento desta abordagem levou a que mais duas abordagens fossem desenvolvidas (Mapeamento e Localização 2.5D, e 3D Simultaneous Localization and Mapping).

Foram realizados testes no sentido de avaliar o desempenho das abordagens propostas. Os testes consistiram em verificar a influência do número máximo de partículas no filtro: na estimação da pose, no mapa gerado e numa pontuação de localização. Os resultados dos testes foram analisados, sendo que os resultados finais atenderam às expectativas estabelecidas para as abordagens.

Palavras Chave: Robô movel, Mapeamento, Localização, SLAM, Filtro de partículas.

"Persistence is the shortest path to success."

Charles Chaplin

Contents

List of Acronyms

AMCL Adaptive Monte Carlo Localization

EKF Extended Kalman Filter

KLD Kullback–Leibler Distance

MCL Monte Carlo Localization

MR Multinomial Resampling

PF Particle Filter

ROS Robot Operating System

SLAM Simultaneous Localization and Mapping

UKF Unscented Kalman Filter

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and motivation

For a mobile robot to move autonomously, it is important to have the knowledge of the environment map and have the ability to locate itself on that map. To date, several approaches have been developed for the environment representation, in which we can differentiate two types of representation: topological maps and metric maps (grids) [1]. The map can represent the environment in 2D, 2.5D, and 3D. One of the first implementations of 2.5D representations, proposed by Herbert *et al.* [2], was designed for a mission to the planet Mars, where a vehicle could collect samples of materials as well as represent the environment. Another approach was proposed by Premebida *et al.* [3], based on a 2.5D polar grid and Kriging to generate denser representations. The information on 2D and 2.5D maps does not provide a good overview of the environment and to overcome it, 3D maps have been developed. Recently an approach to a 3D representation was presented by Garrote *et al.* [4], which combines features from 2D and 3D representations. Localization of mobile robots is one of the most important factors to perform autonomous behavior, where the robot must know its position while moving on a map. Several approaches have been proposed and the PF-based approach is appealing because of its simple implementation. Recently, the Particle Filter has been applied in multi-sensor localization applications [5, 6, 7]. Another approach is described in [8], where data from multiple sensors is fused into a Particle Filter to estimate the robot pose. In scenarios that mapping and localization are solved simultaneously, this is known as Simultaneous Localization and Mapping (SLAM). There are several SLAM applications, including indoor [9, 10, 11], outdoor [12, 13] and sub-sea [14, 15].

1.2 Main objective

The main objective of this dissertation was to develop and test a system for a mobile robot that allows localization and mapping in indoor environments using 2D and 3D maps. Figure 1.1 presents the proposed system.

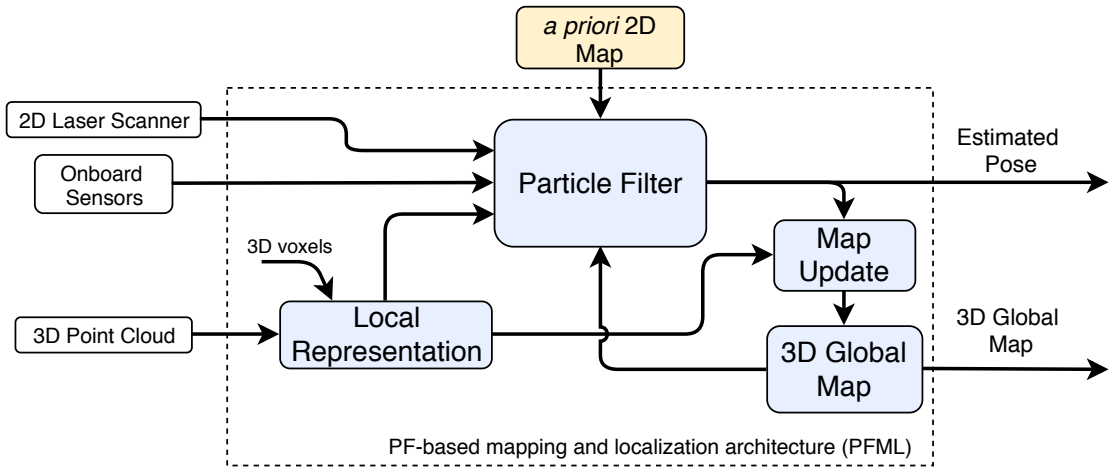


Figure 1.1: Block diagram of the proposed system.

Two main data processing modules compose the proposed PFML architecture:

- **Map Update:** Generates an updated 3D map of the environment. The update is based on the local representation obtained from the 3D point cloud and the estimated pose by the PF.
- **Particle Filter:** The implemented filter estimates the robot pose (x, y, θ) integrating 2D/3D sensory data and 2D/3D maps.

1.3 Main contributions

The goal was to study more complex representations than 2D to provide more faithful representations of the environment, taking advantage of 3D representations and 3D sensors. The development of the approach led initially to the implementation of different approaches, all in Robot Operating System (ROS) middleware. As a final result, three approaches were implemented:

1) 2.5D Mapping and Localization

- Development of the ROS package "PFML1", in which the particle filter uses 2.5D maps to estimate the robot pose. A global map of the environment is updated using the estimated pose and a local map built from a 3D point cloud. The 2D

map given *a priori* is transformed into a 2.5D map which along with the data from the local and global representations are used in the particles' weight computation (update step of PF).

2) 3D Localization and Mapping (SLAM)

- Development of the ROS package "PFML2" allowing a 3D map to be built from a 3D point cloud and from the estimated pose by the PF, and locate itself on that map simultaneously. As in the previous approach, the particle's weights are adjusted based on data from the local and global representations.
- An optimization step has been implemented to minimize the pose error by considering the overlap of the local and the global representations. This stage is based on gradient descent optimization.

3) 3D Mapping and 2D/3D Localization

- Implementation of a ROS package "PFML3", where the PF merges 2D/3D sensory data and 2D/3D maps to estimate the robot pose.

1.4 Dissertation Outline

This Master Dissertation has seven chapters structured as follows:

- **Chapter 2:** Introduction of the existing methods of environment representation, localization and SLAM approaches.
- **Chapter 3:** Provides an overview of some essential algorithms for the proposed approaches in this work.
- **Chapter 4:** Presents in detail the solutions proposed in this dissertation and the changes that are made in the PF according to the proposed approach.
- **Chapter 5:** Gives an overview of the hardware and software used and developed to accomplish the proposed objective.
- **Chapter 6:** Presents and discusses the experimental results of the implemented approaches.
- **Chapter 7:** Contains the conclusions of this work and provides suggestions for future work.

Chapter 2

State of the art

In this chapter, an overview of the existing techniques for mapping will be presented, giving strengths and weaknesses for each case. Most used approaches are also presented in mobile robot localization systems, with the main features being highlighted. Finally, some solutions for the SLAM problem, are presented as well.

2.1 Mapping

Topological approaches produce maps that represent places (nodes) in the environment and how these places are connected (arcs). Regarding the metric approaches, in the most common ones, the environment is represented by a grid. The 2D occupancy grid maps have been introduced by Moravec and Elfes [16], where the map is divided into cells, in which each cell contains an occupancy probability. This representation cannot represent 3D objects, and to overcome this problem the 2.5D and 3D representations have been proposed. The 2.5D representations are known as elevation maps, where each cell contains "height" information about the highest object [2]. The main shortcoming of 2.5D maps is the representation of vertical overlapping objects (e.g. the free space between the ground and the deck of a bridge will be considered as an obstacle). The 3D representations have been proposed to provide a more detailed environment. A popular approach to 3D representation was proposed by Roth-Tabak and Jain [17], in which the occupancy grid is composed of equal size cubic volumes designated *voxels*, to discretize the mapped area. Work with Voxel grids has been presented in [18, 19] where 2D grid maps store a list of voxels in each cell. Another approach to 3D representation is described in [4], called Height-Voxel Map (HMAP), a 2D grid-map with several Height-Voxels aligned vertically, where free and occupied space is shaped by these voxels. A major shortcoming of fixed size voxels grids is their large memory requirement in large-scale outdoor environments or when there is the need for high resolutions. The

OctoMap approach has been proposed by Hournung *et al.* [20] where the environment representation is based on *octrees*. OctoMap supports multi-resolution map queries and the probabilistic representation of occupancy also includes free and undiscovered areas [20]. However, the tree structure of the *octrees* makes access to data more complex than other 3D representations listed above. This approach is widely used in robotics projects and is available as open-source in ROS.

Figure 2.1 shows the mapping approaches obtained from a 3D point cloud. Table 2.1 illustrates the strengths and weaknesses of three mapping approaches.

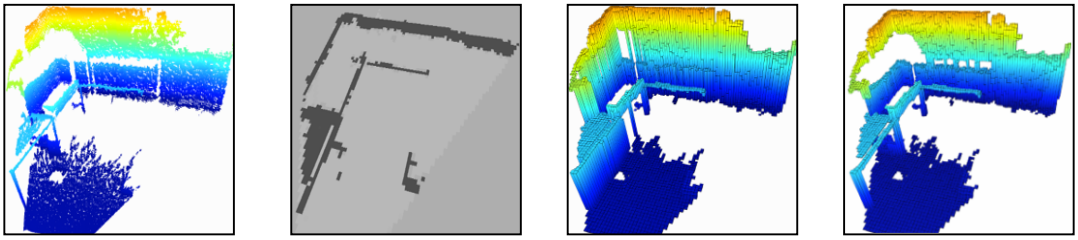


Figure 2.1: Representation of a corner with a table from a 3D point cloud: (a) 3D point cloud; (b) 2D occupancy grid map; (c) Elevation maps (2.5D) and (d) 3D voxel grid map.

Table 2.1: Advantages and disadvantages of mapping approaches (adapted from [20, 21]).

Approaches	**Advantages**	**Disadvantages**
2D Occupancy Map	- Simpler and faster representations - Memory efficient - Constant time access	- Impossible to represent 3D objects, only planar environments
2.5D Map	- Simpler and faster representations - Memory efficient - Constant time access	- Non-probabilistic - No distinction between free and unknown space
3D Occupancy Map	- Volumetric representation - Tree data structures allow multi- resolution - More detailed representation	- Memory requirement - Discretization errors

2.2 Localization

The mobile robot should have the ability to determine its position on the environment map. Given this map, the mobile robot analyses the data from its sensors and estimates its position (see Fig. 2.2).

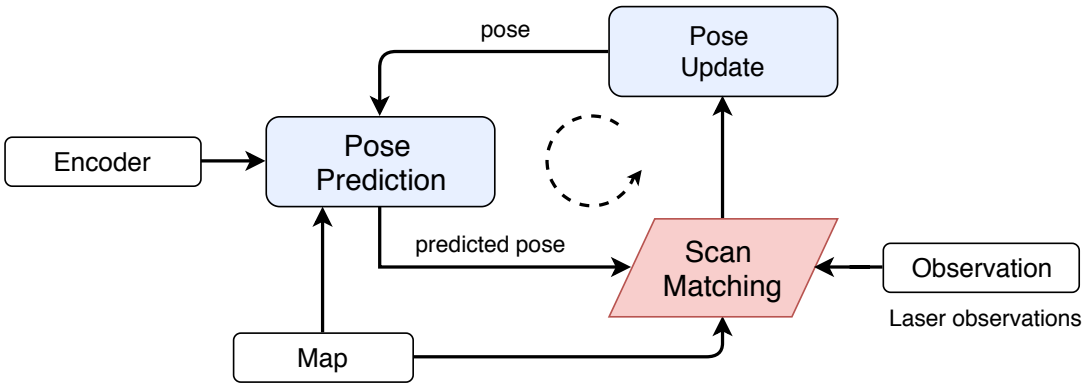


Figure 2.2: Diagram for mobile robot localization.

The concept of Scan Matching for localization, proposed by Feng Lu and Milios [22], is described as a geometric approach, where through lasers scans a rigid transformation between consecutive *matching scans* is obtained. The *Scan Matching* aims to obtain the translation and rotation between the current position and a reference robot position. Methods based on *scan matching* present high speeds and satisfactory accuracies. However, they can fail and may not recover.

Localization has been subject to intense research over the years and different methods have been proposed, based on Extended Kalman Filter (EKF)[21], Unscented Kalman Filter (UKF)[23] and Particle Filter (PF)[24]. The EKF and UKF are variants of Kalman Filter (KF), which can handle nonlinear functions. In the PF method, a set of particles is used to represent the belief of the robot pose. If a region is denser (higher concentration of particles), then the probability of the robot being in that pose is high. PF-based localization is one of the most interesting methods due to its low computational complexity and it does not require much memory or resources [24]. In mobile robot localization, this method is more recognized as Monte Carlo Localization (MCL) and can represent multi-modal distributions and globally localize a robot [24]. A variant of MCL, Adaptive Monte Carlo Localization (AMCL) or also known as *KLD-sampling* is described in [25], where an adaptive approach to MCL is implemented. This method adjusts the number of particles in each iteration, thus increasing the PF efficiency.

2.3 Simultaneous localization and mapping (SLAM)

SLAM is one of the main challenges of robotics. Without prior knowledge of the surrounding space, SLAM allows the robot to navigate in unknown environments, building its map and determining its position on the map.

A pioneer approach of SLAM is based on EKF, known as EKF-SLAM [26]. This solution is computationally heavy, and the space required increases squarely with the number of landmarks [27]. There are several open-source SLAM approaches in ROS. Reviews of the most commonly used approaches was done in [28, 29], in particular GMapping [30], Hector SLAM [31], Google Cartographer [32], Real-Time Appearance-Based Mapping (RTAB-Map) [33], and Laser Odometry and Mapping (LOAM) [34]. Google Cartographer is a LiDAR graph-based SLAM approach, that generate a 2D occupancy grid and supports 3D LiDARs. This approach uses sub-maps and has a graph optimization module for loop closure detection, that minimizes the pose error. Loop closure detection occurs when the robot recognizes a place already visited, allowing it to update a map and reduce the error of the pose estimate. RTAB-Map is a Graph-Based SLAM that also implements loop closure detection. This approach can be used with multiple sensors such as a stereo camera, RGB-D, Kinect, 3D LiDAR and 2D Laser Scan, and generate 3D point clouds of the environment to build a 2D or 3D occupancy grid maps. LOAM is an approach that uses data from a 3D LiDAR and runs two algorithms in parallel with different frame rates. The lowest frame rate is responsible for the odometry to estimate the velocity of the LiDAR and fixes the blurring in the 3D point cloud, and the second algorithm updates a map with the registration of the 3D point cloud using scan-matching [34]. This approach does not include loop closure detection.

Table 2.2 presents a categorization of the above mentioned SLAM approaches, stating the inputs, the methods employed and the outputs.

Table 2.2: Summary of popular ROS-SLAM approaches with their supported inputs, methods and outputs (adapted from [29]).

	Inputs			**Methods**	**Outputs**			
	Odom	LiDAR 2D	LiDAR 3D		Point Cloud	Pose	Ocuppancy 2D	Ocuppancy 3D
GMapping [30]	*	*		Particle Filter		*	*	
Hector SLAM [31]		*		Gauss-Newton		*	*	
Cartographer [32]	*	*	*	Loop closure detection and submaps	*	*	*	
RTAB-Map [33]	*	*	*	Loop closure detection and graph optimization	*	*		*
LOAM [34]	*		*	Two threads for odometry and mapping	*	*		*

Chapter 3

Background material

3.1 Environment representation

The localization of a mobile robot can only be achieved if an environment map is available. Multiple sensors provide data to build or update the map. As mentioned in Section 2.1, there are two types of maps being our focus on the grid-based (metric) maps, in which the environment can be represented by 2D, 2.5D and 3D grids.

3.1.1 Occupancy grid map

The most common 2D grid-based map is the occupancy grid map, in which each cell contains the probability of that cell being occupied. For each sensor measurement (sensor beam), is computed an occupied cell and a set of cells is updated based on Bresenham's algorithm [35], which determines the cells that are crossed by a line (see Fig. 3.1).

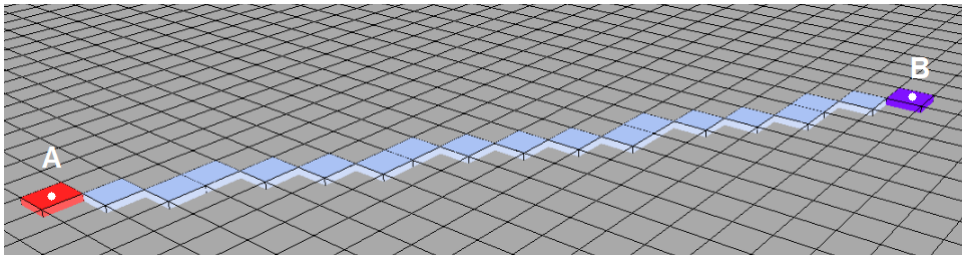


Figure 3.1: Illustration of the 2D line tracing from the robot pose (A) to the occupied cell (B) (grid coordinates). The robot cell is in red, the computed cells are in light blue, and the occupied cell is in purple.

Once the line is defined by two cells, one cell from the robot pose (A) and the other from the occupied cell (B) (both coordinates are transformed into grid coordinates), all cells on the line are set empty and the hit cell as occupied. All cells beyond the hit cell have no

influence. The cell probability is updated by the equation described in [21]:

$$l(x|z_{1:t}) = log\frac{p(x|z_{1:t-1})}{1-p(x|z_{1:t-1})} + log\frac{p(x|z_t)}{1-p(x|z_t)} - \underbrace{log\frac{p(x)}{1-p(x)}}_{=0,\ if\ p(x)=0.5} \tag{3.1}$$

with $p(x)$ the prior probability, $p(x|z_{1:t-1})$ the previous estimate and $p(x|z_t)$ denotes the probability that cell x be occupied given the measurement z_t. The conversion of *log odds* to probabilities of a cell is given by:

$$p(x|z_{1:t}) = 1 - \frac{1}{1+exp(l(x|z_{1:t}))} \tag{3.2}$$

3.1.2 Incremental Map

A different mapping approach for 2D maps is presented in [36], designated the reflection maps. In such maps, each cell contains two counters of *hits* and *misses*, and based on a ratio between them defines the reflection probability of the cell [36]. The hits correspond to the number of cases a sensor beam was reflected in the cell (endpoint) and the misses represent the number of cases a sensor beam crossed through the cell. The reflection probability in the cell (x, y) is given by:

$$p_{ref}(x,y) = \frac{hits_{x,y}}{hits_{x,y} + misses_{x,y}} \tag{3.3}$$

The cells between the sensor and the measured point are selected by using the Bresenham's algorithm [35].

3.2 Particle filter

The particle filter represents the belief of the system state by a set of random particles. Initially, the particles are randomly distributed over the environment, and at each iteration, they converge to the real system state. The set of random particles is denote by:

$$X_t = [\mathrm{x}_t^{[1]}, \mathrm{x}_t^{[2]}, ..., \mathrm{x}_t^{[N]}] \tag{3.4}$$

where each particle $\mathrm{x}_t^{[n]}$ (with $1 \leq n \leq N$) is associated with a hypothesis of the state of the system at the instant t, with their respective weight ($w_t^{[n]}$). N denotes the number of particles in the set X_t, which is usually a fixed value but can be adaptive when implementing the KLD-sampling method [25]. This filter has as input a set of particles that represents the belief of the system state at the instant $t-1$, X_{t-1}, the control u_t and the measurements z_t at the instant t. PF consists in 3 steps: prediction, update and resampling (see Fig. 3.2).

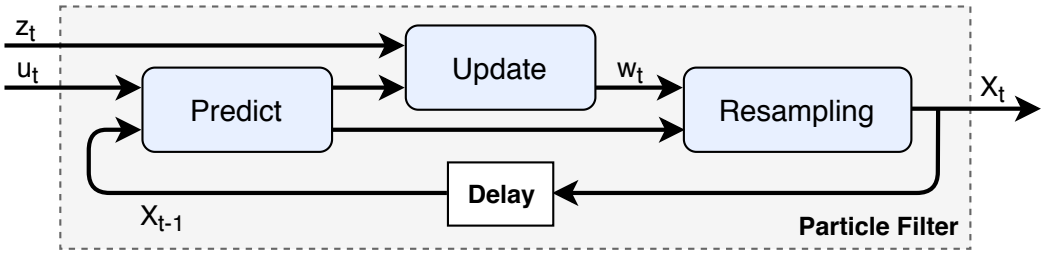


Figure 3.2: Particle filter diagram with: z_t the observation; u_t the control; w_t denotes the particles weight; X_t and X_{t-1} being the actual and the previous particle sets.

In the *Prediction* step, the next state of the particles is predicted using the motion model of the system. In the next step *Update*, the observation model adjusts the particles weights. Finally, the *Resampling* step has the purpose of generating a set of new particles with more particles around the previous ones with higher weights.

The initial belief $bel(x_0)$ is defined by a set of N particles randomly generated according to *a priori* distribution $p(x_0)$, where each particle is assigned an importance factor (weight) $\frac{1}{N}$ [21]. Each step mentioned above is detailed below.

3.2.1 Prediction

The PF predicts a possible state for the platform, $\mathrm{x}_t^{[n]}$, based on the previous particle $\mathrm{x}_{t-1}^{[n]}$ and the motion model u_t. The platform motion over time is calculated based on odometry, i.e., integrating wheel encoders information. The platform moves from a pose $\mathrm{x}_{t-1} = [x_{t-1}\ y_{t-1}\ \theta_{t-1}]^T$ to a pose $\mathrm{x}_t = [x_t\ y_t\ \theta_t]^T$, with $u_t = [\bar{\mathrm{x}}_{t-1}\ \bar{\mathrm{x}}_t]^T$ obtained from the robot odometry. As it is presented in [21] this movement is transformed into a sequence of steps: a rotation (δ_{rot1}), followed by a translation (δ_{trans}) and a second rotation (δ_{rot2}).

Algorithm 1: Sample motion model adapted from [21].

Data: Particles $\mathrm{x}_{t-1}^{[n]} = [x_{t-1}^{[n]}\ y_{t-1}^{[n]}\ \theta_{t-1}^{[n]}]^T$ and $u_t = [\bar{\mathrm{x}}_{t-1}\ \bar{\mathrm{x}}_t]^T$

1 $\delta_{rot1} \leftarrow atan2(\bar{y}_t - \bar{y}_{t-1}, \bar{x}_t - \bar{x}_{t-1}) - \bar{\theta}_{t-1}$;

2 $\delta_{trans} \leftarrow \sqrt{(\bar{x}_t - \bar{x}_{t-1})^2 + (\bar{y}_t - \bar{y}_{t-1})^2}$;

3 $\delta_{rot2} \leftarrow \bar{\theta}_t - \bar{\theta}_{t-1} - \delta_{rot1}$;

4

5 $\hat{\delta}_{rot1} \leftarrow \delta_{rot1} - \mathbf{sample}(\alpha_1\delta_{rot1}^2 + \alpha_2\delta_{trans}^2)$;

6 $\hat{\delta}_{trans} \leftarrow \delta_{trans} - \mathbf{sample}(\alpha_3\delta_{trans}^2 + \alpha_4\delta_{rot1}^2 + \alpha_4\delta_{rot2}^2)$;

7 $\hat{\delta}_{rot2} \leftarrow \delta_{rot2} - \mathbf{sample}(\alpha_1\delta_{rot1}^2 + \alpha_2\delta_{trans}^2)$;

8

9 $x_t^{[n]} \leftarrow x_{t-1}^{[n]} + \hat{\delta}_{trans}\cos(\theta_{t-1}^{[n]} + \hat{\delta}_{rot1})$;

10 $y_t^{[n]} \leftarrow y_{t-1}^{[n]} + \hat{\delta}_{trans}\sin(\theta_{t-1}^{[n]} + \hat{\delta}_{rot1})$;

11 $\theta_t^{[n]} \leftarrow \theta_{t-1}^{[n]} + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$;

12 **return** $\mathrm{x}_t^{[n]} \leftarrow [x_t^{[n]}\ y_t^{[n]}\ \theta_t^{[n]}]^T$

In Algorithm 1, the relative motion parameters are calculated in lines 1 to 3 ($\delta_{rot1}, \delta_{trans}, \delta_{rot2}$).

The corresponding relative motion parameters $(\hat{\delta}_{rot1}, \hat{\delta}_{trans}, \hat{\delta}_{rot2})$ are calculated in lines 5 to 7 by subtracting sampled error to each motion parameter, with $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ being the error parameters. Finally, in lines 9 to 11 the relative motion parameters are added to the previous states obtaining the new states $(x_t^{[n]}, y_t^{[n]}, \theta_t^{[n]})$.

3.2.2 Update

In the update stage, the particle weights are updated based on the observation model. The measurement model is described as a conditional probability distribution $p(z_t|\mathrm{x}_t, M)$, with z_t the measurement at instant t, x_t a particle, and M the map of the environment. Algorithm 2 illustrates an example that can be implemented in the upgrade step using the range beam model [21].

Algorithm 2: Range beam model algorithm adapted from [21].

Data: Particle $\mathrm{x}_t = \{x_t, y_t, \theta_t\}$, beam range measurements $z_t = \{z_1, ..., z_K\}$ and occupancy grid map M

1 $w \leftarrow 0$; // Initialize
2 **for** *all* K **do**
3 $z_k^* \leftarrow$ bresenham_algorithm(x_t, z_k, M) ; // Find the first occupied cell
4 $w \leftarrow w + \mathcal{N}_z(0, z_k^* - z_k)$;
5 **end**
6 **return** w

In Algorithm 2, for each distance measured by the sensor, a distance (z_k^*) to the first occupied cell is obtained by applying the Bresenham's algorithm [35]. This line drawing algorithm is illustrated in Fig. 3.1, and these obtained cells are analyzed to determine the first occupied cell (see Fig. 3.3). In line 4, the weight of the particle is calculated by making the difference between z_k^* and z_k (measured distance).

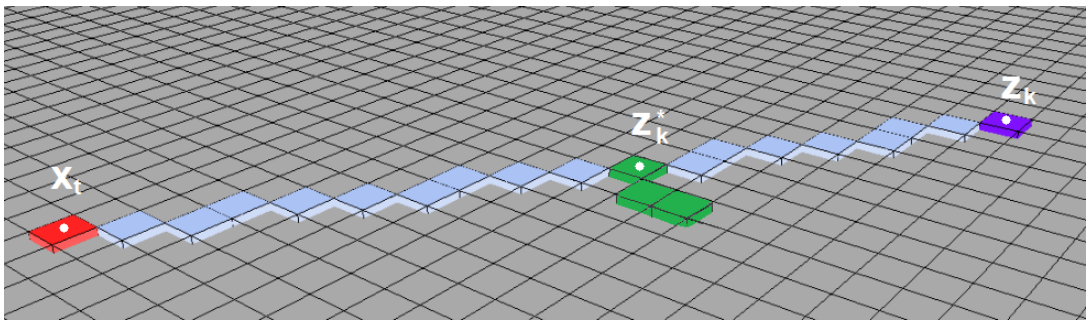


Figure 3.3: Illustration of the line tracing algorithm and the search for the first occupied cell. The robot state x_t is in red, the computed cells crossed by a line are in light blue, the measurement z_k is in purple, and the occupied cell is in green, with z_k^* corresponding to the first detected occupied cell.

3.2.3 Resampling

In the resampling step, a new set of particles ($\hat{X}_t$) is generated from the initial set (X_t) based on the particles' weight. The result of the resampling depends on the resampling method applied, but the idea of this step is to give more emphasis to particles with larger weights than particles with small weights [37, 38]. One method for resampling is the Multinomial Resampling (MR) [37], in which the new set of particles can contain replicated particles from the initial set once the sample is arbitrary and the same particle can be chosen several times. The MR generates N ordered random numbers from the uniform distribution

$$a_i \sim \mathcal{U}[0,1), \quad i = 1, 2, ..., N \ , \tag{3.5}$$

and a particle $\mathrm{x}_t^{[i]}$ is chosen based on these numbers as follows:

$$\hat{\mathrm{x}}_t^{[i]} = \mathrm{x}_t^{[j]}, \quad j = F^{-1}(a_i) \ and \ a_i \in \left[\sum_{s=1}^{j-1} w_t^{[s]}, \sum_{s=1}^{j} w_t^{[s]} \right) \tag{3.6}$$

with $\mathrm{x}_t^{[j]}$ the particle from the set X_t and F^{-1} the inverse cumulative probability distribution of the normalized particle weight.

3.2.4 KLD-sampling

Kullback-Leibler Distance (KLD) sampling is a variant of PF that stands out for adjusting the number of particles over time [25]. The KLD-sampling is illustrated in Algorithm 3. Each particle goes through three steps: resampling (line 4), prediction (line 5), and update (lines 6). After each particle sample, the number of desired samples n_χ is updated based on current k occupied bins. On the grid map, each cell contains an angle histogram with b sections (bins). The computation of k is done by the verification of the particle if it falls into an empty bin or not (line 9). If the particle falls into an occupied bin nothing happens, but if it falls into an empty bin, the number k is incremented and bin b takes the occupied state. In line 12, the number n_χ is calculated, with $z_{1-\delta}$ the upper $1-\delta$ quantile of the standard normal distribution [25]. The variable n increases for each new sample and when n is equal to n_χ, the particles are not sampled anymore in this iteration of the filter (line 15). The difference between this method and the common particle filter (Section 3.2) is in lines 9 to 13.

Algorithm 3: KLD-sampling algorithm adapted from [25].

Data: Posterior particle set $X_{t-1} = \{(\mathrm{x}_{t-1}^{[n]}, w_{t-1}^{[n]}) |\ n = 1, ..., N\}$, representing belief $bel(\mathrm{x}_{t-1})$, control u_t, observation z_t, bounds ε and δ, and minimum number of samples n_{min}

X_t=$\emptyset$, $n \leftarrow 1$, $n_\chi \leftarrow 0$, $k \leftarrow 0$, $\alpha \leftarrow 0$; ; // Initialize

$X_{t-1} \leftarrow \mathrm{Resample}(X_{t-1})$; ; // Rearranges the previous particle set

do

  $\mathrm{x}_{t-1}^{[j]} \leftarrow \mathrm{Draw}(X_{t-1})$; // Samples a particle with index j from X_{t-1}

  $\mathrm{x}_t^{[n]} \leftarrow \mathrm{Predict}(\mathrm{x}_{t-1}^{[j]}, u_{t-1})$; // Predicts next state

  $w_t^{[n]} \leftarrow \mathrm{Update}(z_t,\ \mathrm{x}_t^{[n]})$; // Computes Importance weight

  $\alpha \leftarrow \alpha + w_t^{[n]}$; // Updates normalization factor

  $X_t \leftarrow X_t \cup \{\mathrm{x}_t^{[n]}, w_t^{[n]}\}$; // Inserts new particle in the new particle set

  if $\mathrm{x}_t^{[n]}$ *falls into empty bin* b **then**

    $k \leftarrow k + 1$; // Updates number of non-empty bins

    $b \leftarrow$ occupied; // Marks bin as non-empty

    $n_\chi \leftarrow \frac{k-1}{2\varepsilon}\left(1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}} z_{1-\delta}\right)^3$; // Updates bin size

  end

  $n \leftarrow n + 1$;

while *($n < n_\chi$ and $n < n_{min}$)*;

for $i = 1, ..., n$ **do**

  $w_t^{[i]} \leftarrow w_t^{[i]} / \alpha$; // Normalizes the weight of each particle

end

return X_t

Chapter 4

Developed work

This chapter describes in detail the mapping and localization approaches implemented in the dissertation. The PFML system was developed in response to the proposed approaches, receiving as input odometry data from the wheel encoders, a set of measurements obtained from scanning by the Hokuyo Laser Scanner, and a 3D point cloud obtained by the Velodyne LiDAR. The result of processing this data is the estimation of the pose and an updated map of the environment. Fig. 4.1 shows the developed PFML system.

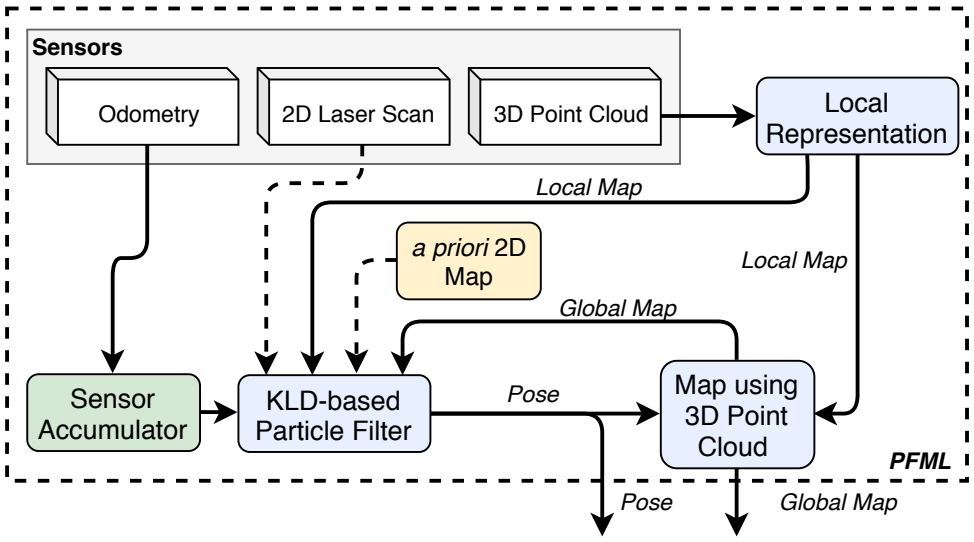


Figure 4.1: Pipeline of the PFML system, including the main steps.

4.1 Sensor Accumulator

In order to guarantee the stability of the PFML approach, an iteration of the filter is subject to thresholds applied to the local angular ($\Delta\theta$) and linear (Δd) displacements of the pose data. The platform motion is calculated based on the current (u_t) and previous (u_{t-1}) odometry measurements that are transformed into two rotations (δ_{rot1} and δ_{rot2}) and one

translation (δ_{trans}) using lines 1 to 3 from Algorithm 1. As long as condition $\delta_{rot1}+\delta_{rot2} > \Delta\theta$ or $\delta_{trans} > \Delta d$ is not verified, the transformed data is accumulated each a new measurement is received. When one of these displacements is reached, the sensor data is sent to the PF, and the accumulator data is reset (see Fig. 4.2).

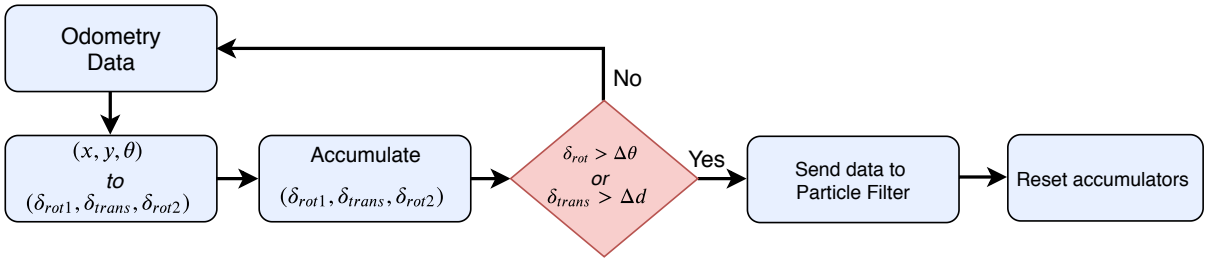


Figure 4.2: Diagram of sensor accumulator, where δ_{rot}=$\delta_{rot1} + \delta_{rot2}$.

4.2 Mapping using 3D Point Clouds

The Velodyne LiDAR provides a 3D point cloud and this data is processed to obtain an environment representation. The block diagram, displaying the data flow from the input to the final representation is illustrated in Fig. 4.3.

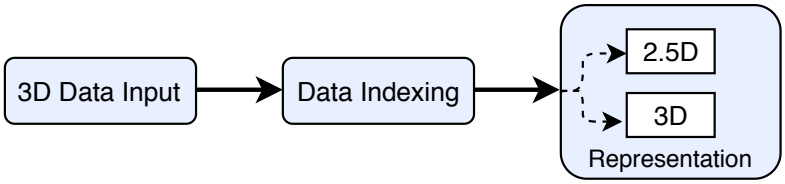


Figure 4.3: Data flow diagram from the point cloud to the desired representation.

The data indexing is done differently for each representation (2.5D and 3D) and is described below for each representation.

4.2.1 2.5D Representation

This representation is based on a 2D grid of square cells (C_{ij}, $i,j \in \mathbb{N}$). Each point in the 3D point cloud is converted into the robot coordinates system and projected into the XY-plane, overlapping with a grid cell C_{ij}. Considering a local grid map of $N_{rows} \times N_{columns}$, with a resolution of s_{cell} per cell and coordinates of the center of the map given by $p^o_m = (x^o_m, y^o_m)$. Converting between Cartesian coordinates (x, y) and grid coordinates (i, j) is given by:

$$(i, j) \leftarrow \left(\frac{(x - x^o_m + \frac{s_{cell}}{2})}{s_{cell}} + \frac{N_{cols}}{2}, \ \frac{(y - y^o_m + \frac{s_{cell}}{2})}{s_{cell}} + \frac{N_{rows}}{2} \right) \tag{4.1}$$

Converting grid coordinates (i, j) to Cartesian coordinates (x, y) is given by:

$$(x, y) \leftarrow \left(s_{cell}(i - \frac{N_{cols}}{2}) + x_m^o,\ s_{cell}(j - \frac{N_{rows}}{2}) + y_m^o \right) \tag{4.2}$$

Each cell in the grid contains two possible values: empty if no point is in that cell or with the lowest and highest z coordinate of the points belonging to that cell. A *height-voxel* is obtained based on the cell height values and an example is illustrated in Fig. 4.4.

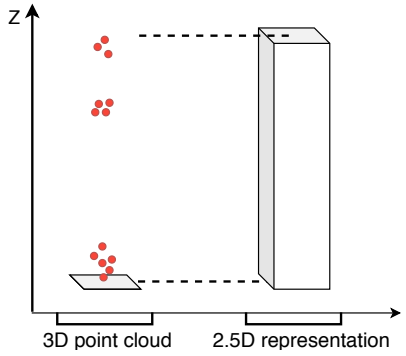


Figure 4.4: Illustration of the 2.5D representation.

In order to get the coordinates of the local map in the world coordinates a rigid transformation is applied.

For this representation, *a priori* 2D global map is provided and being converted into a 2.5D map, assigning a minimum and maximum height to each cell. This map is updated by the height-voxels obtained from the local map. A 3D line tracing algorithm for *height-voxels* was developed to compute the free-space between the sensor and each *height-voxel*. Sensor and each *height-voxel* Cartesian coordinates are projected into a 2D grid map plane. The cells on the line 2D from the 3D Sensor to the *height-voxel* are computed using the Bresenham's line algorithm [35]. For each height-voxel, the minimum and maximum height values are known, and two-line models are defined from the sensor providing a free *height-voxel* for each cell (see Fig. 4.5).

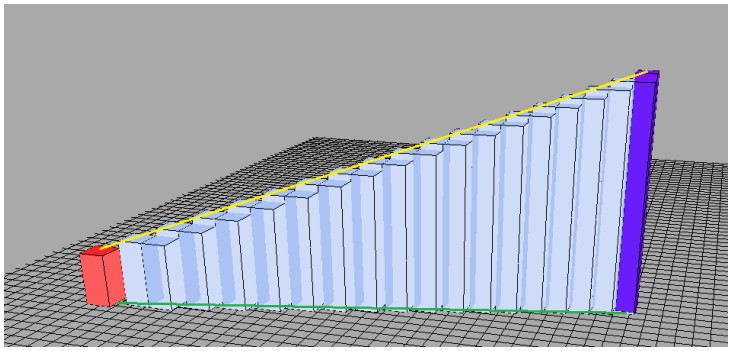


Figure 4.5: Illustration of the line tracing for one height-voxel measured. The sensor is in red, the free height-voxels are in light blue, the measured height-voxel is in purple, and the lines connecting the minimum and maximum values of z are in green and yellow respectively.

The representation generated from the 3D point cloud provides an update of the map given *a priori*, changing the minimum and maximum height values of each cell through the mentioned algorithm. All cells identified by the Bresenham's algorithm that contain data are overlapped with the corresponding free height voxel. Considering the minimum and maximum height values of the height-voxel (h_{min}, h_{max}) and the free height-voxel (fh_{min},fh_{max}) the following observations will occur according to the rules listed below:

(a) If the free-height voxel overlap at the bottom of the height-voxel, it will update the minimum height value to fh_{max} (Fig. 4.6a).

(b) If the free-height voxel overlaps at the top of the height-voxel, it will update the maximum height value to fh_{min} (Fig. 4.6b).

(c) If no free-height voxel overlap height-voxel then, no changes will be made (Fig. 4.6c).

(d) If the free height-voxel completely overlaps the height-voxel, the information is removed (Fig. 4.6d).

(e) If the free height-voxel is part of the height-voxel the Jaccard index is used, also known as Intersection over Union (IoU). The score is defined as the area of overlap divided by the area of union. If the score is greater than 0.5 then it removes the data, otherwise nothing is applied (Fig. 4.6e).

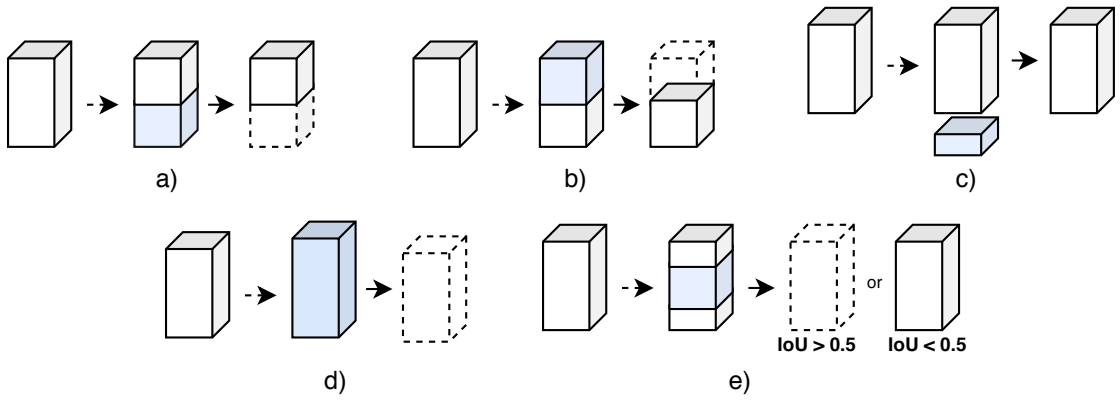


Figure 4.6: Illustration of the overlap method for 2.5D map update. The *height-voxel* in white and the free *height-voxel* in light blue.

4.2.2 3D Representation

The 3D representation is based on a 3D voxel grid map and composed by a set of voxels V_{ijk} with row i, column j and layer k. Starting from the formulation in (4.1) and (4.2) is

possible to obtain the equations for three-dimensional space. Converting between Cartesian coordinates (x, y, z) and grid coordinates (i, j, k) is given by:

$$(i, j, k) \leftarrow \left(\frac{(x - x_m^o + \frac{s_{cell}}{2})}{s_{cell}} + \frac{N_{cols}}{2}, \ \frac{(y - y_m^o + \frac{s_{cell}}{2})}{s_{cell}} + \frac{N_{rows}}{2}, \ \frac{(z - z_m^o + \frac{s_{cell}}{2})}{s_{cell}} + \frac{N_{layers}}{2} \right) \tag{4.3}$$

Converting grid coordinates (i, j, k) to Cartesian coordinates (x, y, z) is given by:

$$(x, y, z) \leftarrow \left(s_{cell}(i - \frac{N_{cols}}{2}) + x_m^o, \ s_{cell}(j - \frac{N_{rows}}{2}) + y_m^o, \ s_{cell}(k - \frac{N_{layers}}{2}) + z_m^o \right) \tag{4.4}$$

The map is built based on the reflective maps but only uses one counter per cell (voxel) to estimate the occupation value. The counter is incremented each time the voxel is observed by the sensor. The voxel is considered occupied if its counter is greater than a defined threshold. Figure 4.7 shows an example of how the voxels are extracted through the 3D point cloud.

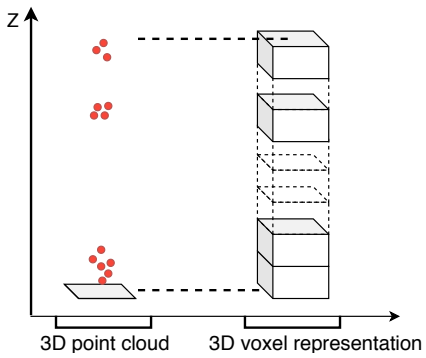


Figure 4.7: Illustration of the 3D voxel representation.

As in 2.5D data indexing, all voxels from the local map are converted in world coordinates using a rigid transformation.

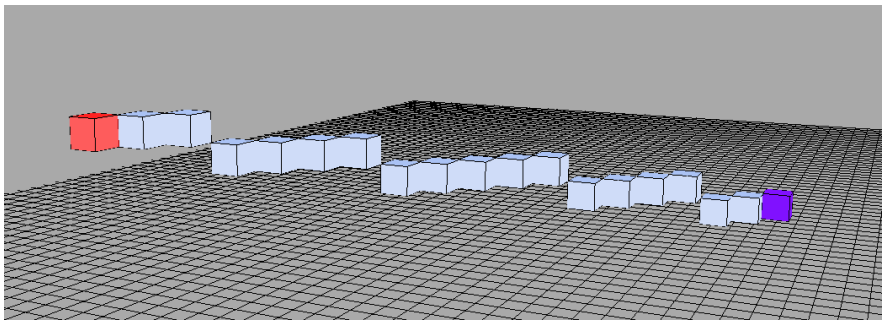


Figure 4.8: Illustration of the 3D line tracing algorithm from the sensor to measured voxel. The sensor is in red, the free computed voxels are in light blue, and the measured voxel is in purple.

Similar to the OctoMap approach [20], a 3D variant of the Bresenham's algorithm is used to compute the voxels that are updated along a beam (see Fig. 4.8). In the measured voxel, the counter is incremented and in the voxels that are crossed by the beam, the counter is

decreased. The algorithm developed to determine which voxels are updated is presented in the Algorithm 10. The *voxelUpdate* procedure updates the counter for a given voxel, adding *val* value (line 9, 21 and 33). The 3D map update is presented in Algorithm 4.

Algorithm 4: Generation of 3D map grid.

Data: Robot position (p^w), set of voxels from the local map (V) and 3D grid Map (M).

1
2 $V^w \leftarrow$ transform(p^w, V); // Transform the coordinates of voxels into world coordinates.
3 $v \leftarrow$ convertToMapIndex(V^w) ; // Convert the world coordinates into grid coordinates
4 $p \leftarrow$ convertToMapIndex(p^w); // Convert pose into grid coordinates
5 **for** *all* v **do**
6 $M \leftarrow$ voxelUpdate$(v_i,\ v_j,\ v_k,\ count)$; // Increments the counter in that grid coordinates
7 $M \leftarrow$ 3DLineTracing$(p_i,\ p_j,\ p_k,\ v_i,\ v_j,\ v_k,\ -count)$; // Tracing algorithm
8 **end**
9 **return** M

Algorithm 4 receives as input the robot pose, a vector with the voxels coordinates from the local map and a 3D global map. Initially, the voxels coordinates correspondent to the local map (robot coordinates system) are transformed into world coordinates (line 2). In lines 3 and 4, the robot position (p_x, p_y, p_z) and the vector of voxels are converted into grid coordinates. In lines 5 to 8, the map update is done by using the *voxelUpdate* and the *3DLineTracing* procedures.

4.3 KLD-based particle filter algorithm

The PF used in this dissertation was based on the methods presented in [21, 25], which follows the KLD-sampling method. This algorithm differs from the one described in Section 3.2, the first step is the resampling, followed by the prediction step and the last step is the update. During the initialization process of the filter, the particles are generated in an initial pose set by the user instead of being randomly distributed over the environment. The initial number of particles is equal to the maximum number of particles (N_{max}), and the weight given to each particle is $\frac{1}{N_{max}}$. At each iteration of the filter the number of desired particles is computed between a defined minimum and maximum. The KLD-sampling algorithm developed is shown in Algorithm 5.

Algorithm 5: KLD-based particle filter algorithm with sensor fusion.

Data: Posterior particle set $X_{t-1} = \{(\mathrm{x}_{t-1}^{[n]}, w_{t-1}^{[n]})|\ n = 1, ..., N\}$, representing belief $bel(\mathrm{x}_{t-1})$, control measurement u_t, observation z_t , bounds ε and δ, bin size, minimum and maximum number of samples N_{min},N_{max}

1 $X_t \leftarrow \emptyset$, $k \leftarrow 0$, $n \leftarrow 1$, $n_\chi \leftarrow 0$, $\alpha \leftarrow 0$; // Initialize
2 $X_{t-1}^* \leftarrow \mathrm{sort}(X_{t-1})$; // Orders the particles from the lowest to the biggest weight
3 $X_{t-1}^* \leftarrow \mathrm{prepare}(X_{t-1}^*)$; // Gives the particle set the weight counter
4 **do**
5 $\quad \mathrm{x}_{t-1}^{[n]} \leftarrow \mathrm{pick}(X_{t-1}^*)$; // Samples a particle from the previous set
6 $\quad \mathrm{x}_t^{[n]} \leftarrow \mathrm{Prediction}(\mathrm{x}_{t-1}^{[n]},\ u_t)$; // Predicts next state
7 $\quad w_t^{[n]} \leftarrow \mathrm{Update}(\mathrm{x}_t^{[n]},\ z_t,\ M)$; // Computes importance weight
8 $\quad \alpha \leftarrow \alpha + w_t^{[n]}$; // Updates normalization factor
9 $\quad$ **if** $(inEmptyBin(\mathrm{x}_t^{[n]}))$ **then**
10 $\quad\quad k \leftarrow k + 1$; // Updates number of non-empty bins
11 $\quad\quad \mathrm{setBin}(\mathrm{x}_t^{[n]})$; // Marks bin as non-empty
12 $\quad\quad n_\chi \leftarrow \frac{k-1}{2\epsilon}(1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}}z_{1-\delta})^3$; // Updates number of samples
13 $\quad$ **end**
14 $\quad n \leftarrow n + 1$;
15 $\quad$ **if** $(n >= N_{max})$ **then**
16 $\quad\quad$ break;
17 $\quad$ **end**
18 **while** $(n < n_\chi \ and \ n < N_{min})$;
19 **for** $n := 1, ..., N$ **do**
20 $\quad w_t^{[n]} \leftarrow w_t^{[n]}/\alpha$; // Normalizes the weight of each particle
21 **end**
22 **return** $X_t \leftarrow \{(\mathrm{x}_t^{[n]}, w_t^{[n]})|\ n = 1, ..., N\}$

4.3.1 Resampling

The approach used is based on the multinomial resampling, described in Subsection 3.2.3. Initially, the particles are sorted from the lowest to the highest weight. In line 2 from Algorithm 5, a new ordered particle set (X_{t-1}^*) is obtained from the previous particle set (X_{t-1}). In line 3, each ordered particles receives a number containing the sum of all the weights up to the respective particle. The attribution of this number is made by the following equation:

$$f(\mathrm{x}_{t-1}^{*[n]}) = \sum_{i=1}^{n} w_{t-1}^{[i]}, \quad with \ \ 0 < f(\mathrm{x}_{t-1}^{*[n]}) \leq 1 \tag{4.5}$$

In this step, a new point mass distribution is created. In line 5, the algorithm finds the first number in the point mass distribution with the value greater than the sampled number obtained from the uniform distribution ($a \sim \mathcal{U}(0, 1)$). The particle is chosen based on the correspondent number and is prepared to advance through the prediction and update steps until the number of desired samples was reached. After all the process, the particle's weights are normalized ($\sum_{n=1}^{N} w^{[n]} = 1$).

Figure 4.9 shows graphically an example of the resampling step.

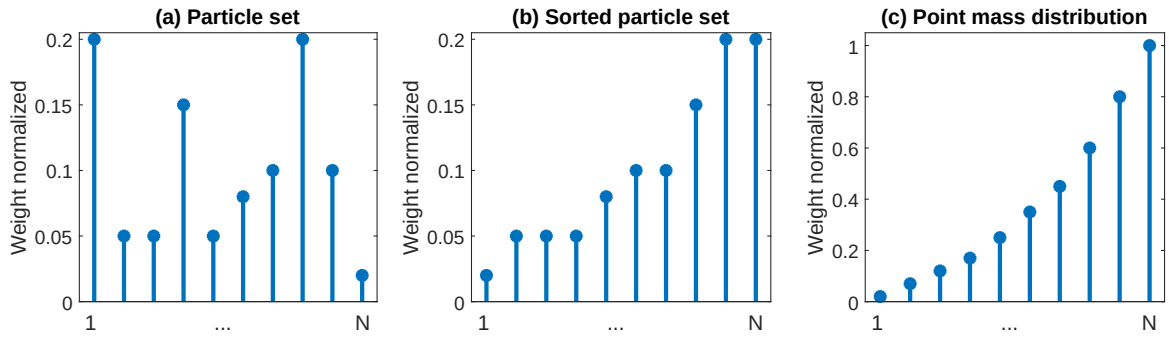


Figure 4.9: Graphics showing an example of the resampling step: (a) Particle set, X_{t-1}; (b) Sorted particle set, X^*_{t-1}; (c) Point mass distribution, $f(\mathrm{x}^{*[n]}_{t-1}) = \sum_{i=1}^{n} w^{[i]}_{t-1}$, used to sample a particle for the new particle set. N denotes the number of particles in a set.

4.3.2 Prediction

As described in Subsection 3.2.1, the robot predicts its new state based on the previous state (X^*_{t-1}) and motion parameters ($\delta_{rot1},\ \delta_{trans},\ \delta_{rot2}$), applying the motion model to each particle. The process of the prediction step is described in Algorithm 6.

Algorithm 6: Motion model algorithm adapted from [21].

Data: Particle $\mathrm{x}^{[n]}_{t-1} = \{x^{[n]}_{t-1}, y^{[n]}_{t-1}, \theta^{[n]}_{t-1}\}$ and motion parameters $\{\delta_{rot1}, \delta_{trans}, \delta_{rot2}\}$

$\hat{\delta}_{rot1} \leftarrow 0,\ \hat{\delta}_{trans} \leftarrow 0,\ \hat{\delta}_{rot2} \leftarrow 0;$ // Initialize

$\hat{\delta}_{rot1} \leftarrow \delta_{rot1} - \delta \sim \mathcal{N}(0,\ (\alpha_1 \cdot \delta^2_{rot1} + \alpha_2 \cdot \delta^2_{trans}));$

$\hat{\delta}_{trans} \leftarrow \delta_{tans} - \delta \sim \mathcal{N}(0,\ (\alpha_4 \cdot (\delta^2_{rot2} + \delta^2_{rot1}) + \alpha_3 \cdot \delta^2_{trans}));$

$\hat{\delta}_{rot2} \leftarrow \delta_{rot2} - \delta \sim \mathcal{N}(0,\ (\alpha_1 \cdot \delta^2_{rot2} + \alpha_2 \cdot \delta^2_{trans}));$

$x^{[n]}_t \leftarrow x^{[n]}_{t-1} + \hat{\delta}_{trans}\ \cos(\theta^{[n]}_{t-1} + \hat{\delta}_{rot1});$

$y^{[n]}_t \leftarrow y^{[n]}_{t-1} + \hat{\delta}_{trans}\ \sin(\theta^{[n]}_{t-1} + \hat{\delta}_{rot1});$

$\theta^{[n]}_t \leftarrow \theta^{[n]}_{t-1} + \hat{\delta}_{rot1} + \hat{\delta}_{rot2};$

return $\mathrm{x}^{[n]}_t \leftarrow \{x^{[n]}_t, y^{[n]}_t, \theta^{[n]}_t\}$

4.3.3 Update

The update step computes the particles' weight based on the global map and the observations given by the sensors. The system receives a 2D laser scan and a 3D point cloud, and this data is used according to the intended approach. The operation of each approach and the update step of the PF will be described in Sections 4.4, 4.5 and 4.6.

4.3.4 Pose estimation

Finally, after computing the particle weights, a pose estimate ($\hat{x}_t$) is obtained by the following equations:

$$\hat{\mathrm{x}}_t = \begin{cases} \hat{x}_t = \sum_{n=1}^{N} x_t^{[n]} \cdot w_t^{[n]} \\ \hat{y}_t = \sum_{n=1}^{N} y_t^{[n]} \cdot w_t^{[n]} \\ \hat{\theta}_t = \arctan 2(\sum_{n=1}^{N} \sin(\theta_t^{[n]}) \cdot w_t^{[n]}, \ \sum_{n=1}^{N} \cos(\theta_t^{[n]}) \cdot w_t^{[n]}) \end{cases} \tag{4.6}$$

with N being the number of particles (the number is adjusted by the KLD-sampling approach) and $\hat{\mathrm{x}}_t$ the estimated robot's pose.

4.4 2.5D Mapping and Localization

In this approach, the KLD-based filter estimates the robot's pose using odometry data, an *a priori* 2D map and 3D point cloud, and generates an updated 2.5D map. Receives a 3D point cloud, which is processed to obtain a 2.5D local representation. Initially, an *a priori* 2D map is given, which is converted into a 2.5D map as discussed in Subsection 4.2.1. The 2.5D map is regularly updated using the pose estimated by the filter and the local representation. Both the global and local maps are also described in Subsection 4.2.1.

As shown in Fig. 4.10, the computation of the particles' weight is based on an *a priori* 2.5D map, local and global map.

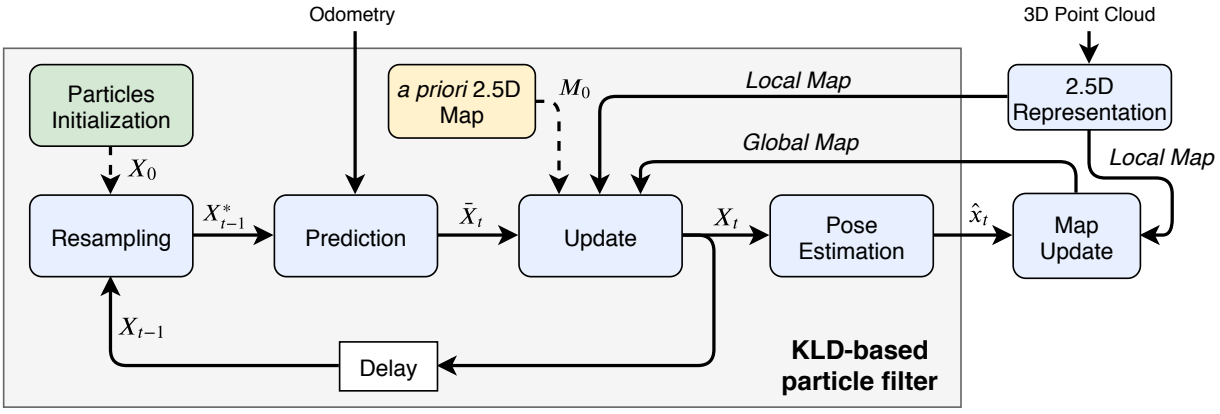


Figure 4.10: KLD-based particle filter diagram with: X_0 the initial set of particles; X_{t-1} and X_t being the previous and the actual particle sets; X^*_{t-1} the rearranged previous particle set; $\bar{X}_t$ the actual predicted set; $\hat{\mathrm{x}}_t$ the actual estimated pose; M_0 the 2.5D prior map; $LocalMap$ and $GlobalMap$ denotes the local and global map obtained from the mapping module.

The update step of the KLD-based filter is described in Algorithm 7 and has as inputs: the particle, data from the local and global map. The computation of the particle's weight is made in lines 2 to 7, and some voxels of the 2.5D local map are skipped (by ignoring C voxels for each voxel processed) in order to decrease the processing time. Each selected voxel is transformed to the world frame applying a rigid transformation (line 3) and then used in the *nearest* procedure. This procedure searches in the global 2.5D map the nearest voxel to the selected voxel (from the local 2.5D map), and returns the voxel coordinates that gets the highest overlap (IoU) with the selected voxel. In line 5, the weight is computed by making the difference between the selected voxel and the voxel (obtained from the *nearest* procedure) into the normal distributions $\mathcal{N}_x$ and $\mathcal{N}_y$.

Algorithm 7: Update stage algorithm.

Data: Particle $\mathrm{x}_t^{[n]} = \{x_t^{[n]}\ y_t^{[n]}\ \theta_t^{[n]}\}$, set of voxels from the local map (LM), set of voxels from the global map (GM) and C a constant.

1 $w_t^{[n]} \leftarrow 0$, $j \leftarrow 0$;
2 **for** $i = 1$; $i < size(LM)$; $i = i + C$ **do**
3 $\quad pt^i \leftarrow \text{transform}(\mathrm{x}_t^{[n]},\ LM_x^i,\ LM_y^i,\ LM_{zmin}^i,\ LM_{zmax}^i)$; `// Transforms the selected voxel to the world frame`
4 $\quad (x_c^i, y_c^i) \leftarrow \text{nearest}(pt^i,\ GM)$; `// Checks neighborhood`
5 $\quad w_t^{[n]} \leftarrow w_t^{[n]} + \mathcal{N}_x(0,\ pt_x^i - x_c^i) \cdot \mathcal{N}_y(0,\ pt_y^i - y_c^i)$;
6 $\quad j \leftarrow j + 1$;
7 **end**
8 $w_t^{[n]} \leftarrow \frac{w_t^{[n]}}{j}$;
9 **return** $w_t^{[n]}$;

4.5 3D Localization and Mapping (SLAM)

A SLAM approach from a KLD-based filter is presented here, where the robot builds a 3D map and estimates its position simultaneously. The estimation of the pose is determined based on the odometry data and the 3D point cloud. The 3D map is updated using the estimated pose and data from the 3D local map, provided by a 3D point cloud as described in Subsection 4.2.2. After the pose estimate obtained by the update stage of the KLD-based filter, an optimization procedure is used, in which minimizes the pose error based on gradient descent (Optimization bock). The description of this block will be presented in the Subsection 4.5.1. Local and global map data is used to calculate the particles' weight in the update step of the KLD-based filter (see Fig. 4.11).

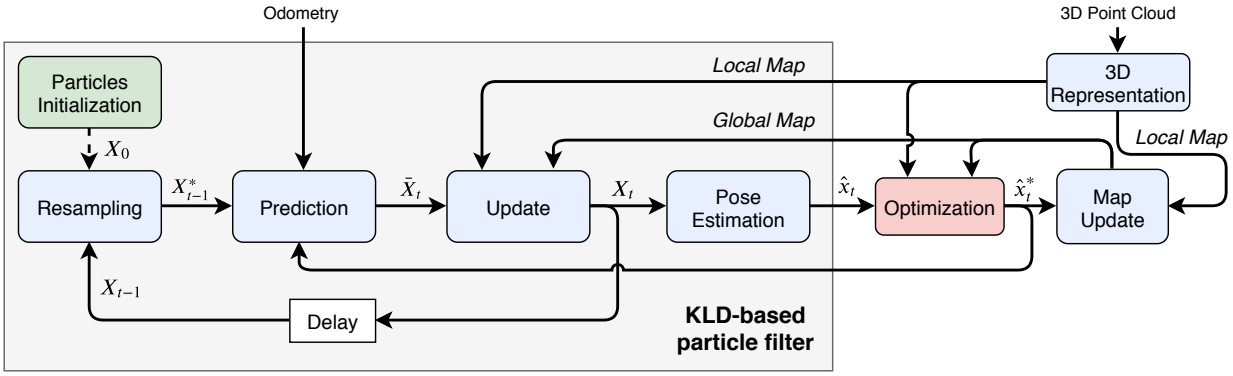


Figure 4.11: KLD-based particle filter diagram with: X_0 the initial set of particles; X_{t-1} and X_t being the previous and the actual particle sets; X^*_{t-1} the rearranged previous particle set; $\bar{X}_t$ the actual predicted set; $\hat{\mathrm{x}}_t$ the actual estimated pose; $\hat{\mathrm{x}}^*_t$ the optimized estimated pose; $LocalMap$ and $GlobalMap$ denotes the local and global map obtained from the mapping module.

Algorithm 8 describes the update step of KLD-based filter and has as inputs: the particle and data from the local and global map. As in the 2.5D Mapping and Localization approach, only a few voxels are used to compute the particle weight. In line 3, each voxel is transformed from the robot frame to the world frame and used in the *nearest* procedure, which searches in the neighborhood of the selected voxel for occupied voxels in the global map, and returns the coordinates of the nearest voxel. The particle's weight is obtained by computing the difference between the selected voxel and the nearest occupied voxel, and applying normal distributions for each difference ($\mathcal{N}_x$, $\mathcal{N}_y$ and $\mathcal{N}_z$).

Algorithm 8: Update stage algorithm.

Data: Particle $\mathrm{x}_t^{[n]} = \{x_t^{[n]}\ y_t^{[n]}\ \theta_t^{[n]}\}$, set of voxels from the local map (LM), set of voxels from the global map (GM) and C a constant.

1 $w_t^{[n]} \leftarrow 0$, $j \leftarrow 0$;
2 **for** $i = 1$; $i < size(LM)$; $i = i + C$ **do**
3 $\quad p^i \leftarrow \text{transform}(x_t^{[n]},\ y_t^{[n]},\ \theta_t^{[n]},\ LM_x^i,\ LM_y^i,\ LM_z^i)$; `// Transforms the selected voxel to the to the world frame`
4 $\quad (x^i_{occu}, y^i_{occu}, z^i_{occu}) \leftarrow \text{nearest}(\ p_x^i,\ p_y^i,\ p_z^i,\ GM)$; `// Checks neighborhood`
5 $\quad w_t^{[n]} \leftarrow w_t^{[n]} + \mathcal{N}_x(0,\ p_x^i - x^i_{occ}) \cdot \mathcal{N}_y(0,\ p_y^i - y^i_{occ}) \cdot \mathcal{N}_z(0,\ p_z^i - z^i_{occ})$;
6 $\quad j \leftarrow j + 1$;
7 **end**
8 $w_t^{[n]} \leftarrow \frac{w_t^{[n]}}{j}$;
9 **return** $w_t^{[n]}$;

4.5.1 Optimization

This block was implemented based on [39], it uses a gradient descent to minimize the error between the 3D map and the map obtained from the 3D point cloud. The main aim is to improve the pose estimate ($\hat{\mathrm{x}}_t$) from the PF. The cost function to minimize is defined as follows:

$$F = \sum_{i=1}^{N} \left| v_i - M(v_i) \right|^2 \tag{4.7}$$

with $M(v_i)$ the occupied voxel of the 3D map inside a R radius of the voxel measured v_i. The update step of the gradient descent is done by A.5 with n the current iteration and $x_0 = \hat{x}_t$. A number n_I defines the number of iterations that the descending gradient is applied. $\nabla F(x_n)$ is given by the approximation of the partial derivatives of F:

$$\begin{bmatrix} \frac{\sum_{i=1}^{n_l}(x - x_i)}{n_l} \\ \frac{\sum_{i=1}^{n_l}(y - y_i)}{n_l} \\ \begin{pmatrix} \frac{\sum_{i=1}^{n_l}(s_x\cos(\theta) - s_y\sin(\theta))(y - y_i + s_y\cos(\theta) + s_x\sin(\theta))}{n_l} \\ -\frac{\sum_{i=1}^{n_l}(s_y\cos(\theta) + s_x\sin(\theta))(x - x_i + s_x\cos(\theta) - s_y\sin(\theta))}{n_l} \end{pmatrix} \end{bmatrix} \tag{4.8}$$

with $M(v_i) = (x_i, y_i)$ and $x_n = (x, y, \theta)$.

Approaches using Gauss-Newton or Levenberg-Marquardt optimization were considered but not implemented. These optimization methods would significantly increase the computational cost (e.g. Jacobian matrix products and computing an inverse matrix). Gradient descent optimization has a lower computational cost and for the intended approach, it can successfully minimize the estimated pose error.

In Fig. 4.11, the prediction step receives the estimated pose from the optimization block and some particles (stratified approach) use that pose to predict the new state.

4.6 3D Mapping and 2D/3D Localization

In this proposed approach, the robot pose is estimated by the KLD-based filter and generates an updated 3D map. Odometry data, an *a priori* 2D map, 2D scan measurements, and 3D point cloud are used to estimate the pose. The 3D map is updated using data from the 3D local map obtained from a 3D point cloud, and the estimated pose from the filter.

The particle weight is obtained by merging the data from 2D and 3D sensors. In comparison to the 3D SLAM approach, an *a priori* 2D map (M) and the laser scan measure (z_t) are added to compute the particle weight (see Fig. 4.12).

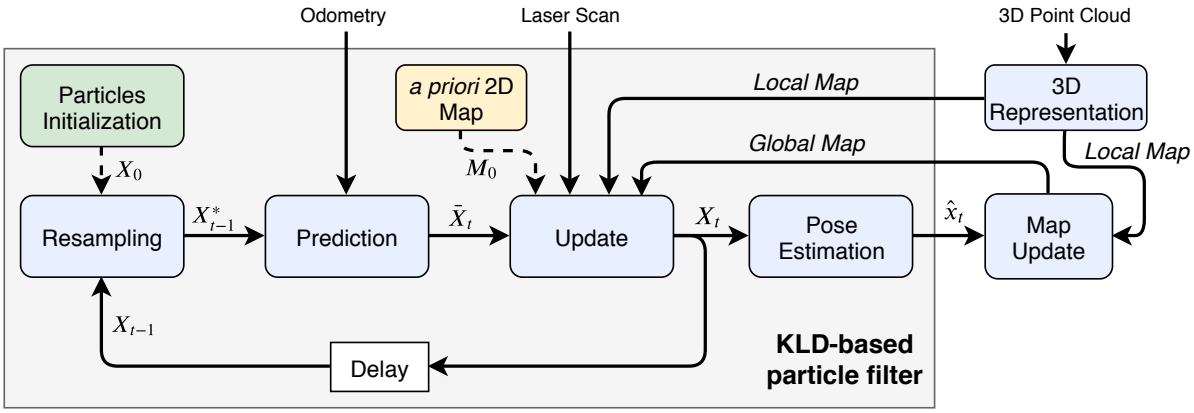


Figure 4.12: KLD-based particle filter diagram with: X_0 the initial set of particles; X_{t-1} and X_t being the previous and the actual particle sets; X^*_{t-1} the rearranged previous particle set; $\bar{X}_t$ the actual predicted set; $\hat{\mathrm{x}}_t$ the actual estimated pose; M_0 the 2D prior map; $Laser\ Scan$ the observations (z_t); $LocalMap$ and $GlobalMap$ denotes the local and global map obtained from the mapping module.

The calculation of the weight is divided into two parts, where the first part is related to the 2D data given by the laser scanner and the second part with 3D data given by the Velodyne sensor. Each part is associated with a weight factor (K_{2D} and K_{3D}), which allows setting the importance given to 2D or 3D data.

The process of the update step of the KLD-based filter is presented in Algorithm 9. Considering the particles state, the measure z_t is transformed to the world frame using a rigid transformation (line 5). As in the previous approaches, only some scanned points are used. In lines 7 and 8, the coordinates of the measured point (x^i_{hit}, y^i_{hit}) are computed and used in a 2D Bresenham's algorithm ($nearest2D$). The $nearest2D$ procedure returns the coordinates of the first occupied cell (x_{occu}, y_{occu}) along the beam between the particle and the measured point. The first part of the weight calculation is obtained in line 10, making the difference between the point selected and the first occupied cell, and applying normal distributions for each difference ($\mathcal{N}_x$ and $\mathcal{N}_y$). The second part of the calculation is done as in the 3D SLAM approach, obtaining the total weight of the particle by the sum of the two weights computed in lines 10 and 17.

Algorithm 9: Update stage algorithm.

Data: Particle $\mathrm{x}_t^{[n]} = \{x_t^{[n]}\ y_t^{[n]}\ \theta_t^{[n]}\}$, laser scan $z_t = \{z_1, ..., z_K\}$ with K being the number of ranges of the laser scan, set of voxels from the local map (LM), set of voxels from the global map (GM), M the 2D occupancy grid map and C a constant.

$\alpha \leftarrow \alpha_{min}$; // Minimum angle of the laser scan range

$Inc \leftarrow \alpha_{inc}$; // Angle increment between measurements, z_k

$w_{2D}^{[n]} \leftarrow 0$, $w_{3D}^{[n]} \leftarrow 0$, $x_{hit} \leftarrow 0$, $y_{hit} \leftarrow 0$, $j \leftarrow 0$;

$d = \text{transform}(x_t^{[n]},\ y_t^{[n]},\ \theta_t^{[n]},\ z_t)$; // Transforms the scan points to the world frame

for $i = 1$; $i < K$; $i = i + C$ **do**

  $x_{hit}^i \leftarrow x_t^{[n]} + d^i \cdot cos(\alpha)$;

  $y_{hit}^i \leftarrow y_t^{[n]} + d^i \cdot sin(\alpha)$;

  $(x_{occ}^i, y_{occ}^i) \leftarrow \text{nearest2D}(x_t^{[n]},\ y_t^{[n]},\ x_{hit}^i,\ y_{hit}^i,\ M)$; // 2D Tracing algorithm

  $w_{2D}^{[n]} \leftarrow w_{2D}^{[n]} + \mathcal{N}_x(0, x_{hit}^i - x_{occ}^i) \cdot \mathcal{N}_y(0, y_{hit}^i - y_{occ}^i)$;

  $\alpha \leftarrow \alpha + C \cdot Inc$;

  $j \leftarrow j + 1$;

end

for $i = 1$; $i < size(LM)$; $i = i + C$ **do**

  $p^i \leftarrow \text{transform}(x_t^{[n]},\ y_t^{[n]},\ \theta_t^{[n]},\ LM_x^i,\ LM_y^i,\ LM_z^i)$; // Transforms the voxel to the world frame

  $(x_{occu}^i, y_{occu}^i, z_{occu}^i) \leftarrow \text{nearest}(\ p_x^i,\ p_y^i,\ p_z^i,\ GM)$; // Checks neighborhood

  $w_{3D}^{[n]} \leftarrow w_{3D}^{[n]} + \cdot\mathcal{N}_x(0,\ p_x^i - x_{occ}^i) \cdot \mathcal{N}_y(0,\ p_y^i - y_{occ}^i) \cdot \mathcal{N}_y(0,\ p_z^i - z_{occ}^i)$;

  $j \leftarrow j + 1$;

end

$w_t^{[n]} \leftarrow \frac{K_{2D} w_{2D}^{[n]} + K_{3D} w_{3D}^{[n]}}{j}$;

return $w_t^{[n]}$;

Chapter 5

Validation platform

The mobile robot shown in Fig. 5.1, known as InterBot [40], was developed in ISR-UC and can currently navigate with a high degree of autonomy. This chapter presents an overview of the physical setup of the InterBot platform and describes its hardware and software architectures.

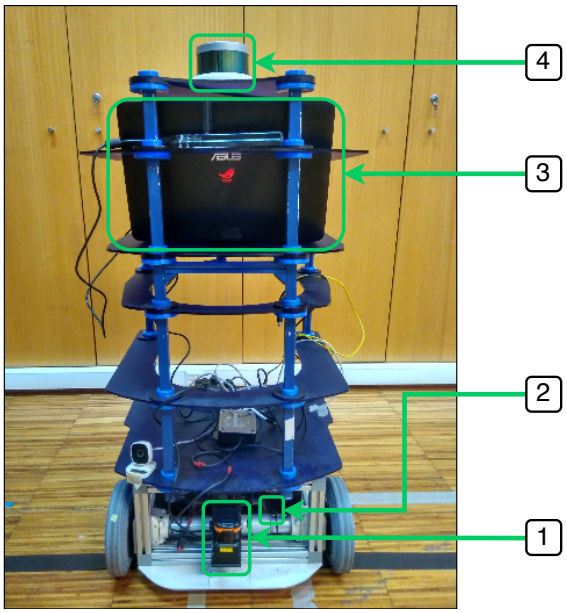


Figure 5.1: InterBot Platform is composed by: (1) Hokuyo Laser Scanner; (2) RoboteQ Motor Controller; (3) Processing Unit laptop and (4) Velodyne LiDAR.

5.1 Hardware Architecture

The InterBot system contains a low-level part for power and mechanical components, and a hight-level part related to processing. Hardware architecture of Interbot platform is illustrate in Fig. 5.2.

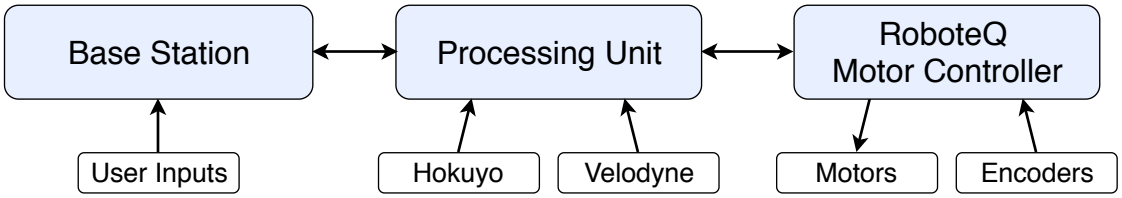


Figure 5.2: InterBot's Architecture.

5.1.1 Base station

The base station is comprised of a remote computer that allows control of the InterBot platform through a joystick controller connected via Bluetooth. This base only publishes and subscribes to ROS, not being directly connected to the lower levels of InterBot Architecture. The base station allows to observe several data that is being processed in Interbot, such as: laser scan points from the Hokuyo Laser Scanner, 3D point cloud from the Velodyne, odometry data, mapping and localization data, among others.

5.1.2 Processing unit

A laptop located on the InterBot platform executes the ROS package developed for the respective approach. Receives as inputs the 3D point cloud from the Velodyne LiDAR, laser scan data from the Hokuyo laser scanner, odometry data, and the linear and angular speed from the Base Station.

5.1.3 RoboteQ Motor Controller

The RoboteQ Motor Controller transforms speed commands into voltage and current outputs to move one or two DC motors. The platform odometry is estimated based on the wheel encoder information. This module receives speed commands from the Processing Unit and returns the odometry data. This data exchange is made through USB communication.

5.1.4 Velodyne LiDAR and Hokuyo Laser Scanner

The Velodyne sensor and the Hokuyo laser scanner provide data about the surrounding space of the platform, through a 3D point cloud and scan measurements, respectively. These sensors are connected to the Processing Unit through an USB connection. The sensors are not aligned with the base of the platform and a rigid transformation is applied to the sensors, in order to use the data correctly.

$$^{Robot}T_{LiDAR} = \begin{bmatrix} 1 & 0 & 0 & -0.13 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1.06 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad ^{Robot}T_{Hokuyo} = \begin{bmatrix} 1 & 0 & 0 & 0.09 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.17 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Table 5.1 and 5.2 show the main specifications of the Velodyne sensor and Hokuyo laser scanner, respectively.

Table 5.1: Velodyne LiDAR VLP-16 main specifications [41].

Supply Voltage	12 V (DC) W	**Nº Channels**	16 (∼300,000 points/sec)
Power Consumption	8 W	**Range Accuracy**	Up to ±3 cm
Measurement Range	100 m	**Rotation Rate**	5 Hz – 20 Hz
Field of View (Vertical)	+15.0° to −15.0° (30°)	**Angular Resolution (Vertical)**	2.0°
Field of View (Horizontal)	360°	**Angular Resolution (Horizontal)**	0.1°

Table 5.2: Hokuyo's UTM-30LX laser main specifications [42].

Supply Voltage	12 V (DC)	**Scan Speed**	25 ms
Guaranteed Range	0.1 ∼ 30 m	**Maximum Range**	0.1 ∼ 60 m
Scan Angle	270°	**Angular Resolution**	0.25°
Measurement Step	1080	**Measurement Resolution**	1 mm

5.2 Software Architecture

The high-level part is the Processing Unit, so it has to perform all the computation concerning mapping, localization, receive speed commands from the Base Station, send relevant information and communicate with the low-level parts. The software in the Processing Unit was all developed in the C++ language and runs inside the ROS environment [43], a framework widely used in the robotics field. Fig. 5.3 shows how the Base Station and the InterBot Processing Unit are related to each other.

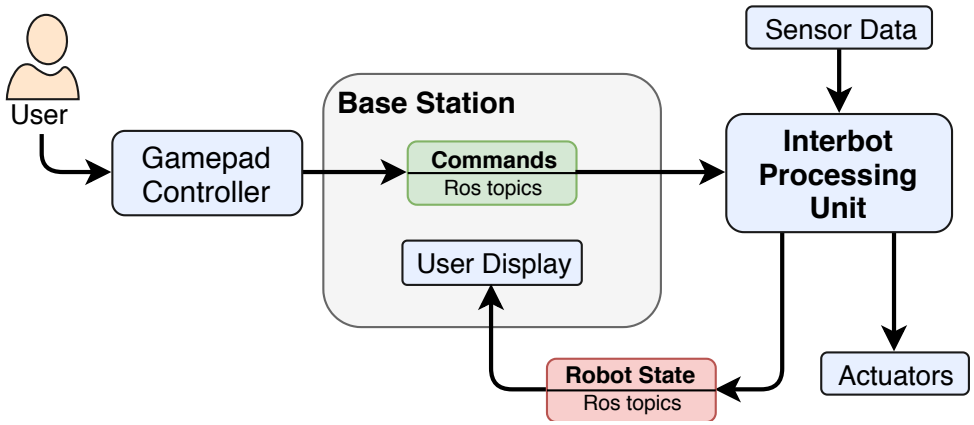


Figure 5.3: Overview of the InterBot software architecture.

5.2.1 Base Station

The Base Station functions as a remote control station, which is responsible for running the software required to control the InterBot's movements and visualization information about mapping and localization (see Fig. 5.4). For the visualization of the results obtained in this dissertation, as well as the data acquired by the sensors, the Base Station uses the RVIZ program, which is a tool afforded by ROS to visualize the available topics published by the online nodes.

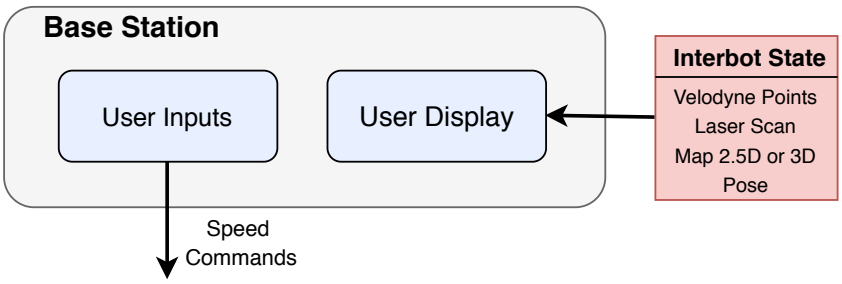


Figure 5.4: Overview of the Base Station software architecture, showing the inputs and the outputs.

5.2.2 Processing Unit

As stated in 5.2, the Processing Unit is the most important module of the platform, and the software architecture is illustrated in Fig. 5.5.

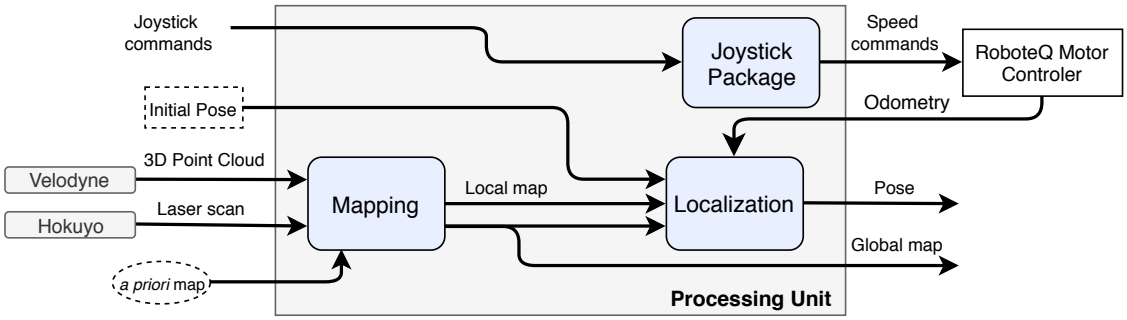


Figure 5.5: Processing Unit software architecture, displaying the nodes and their inputs/outputs.

5.2.2.1 Mapping and Localization

As stated in Section 1.3, three different mapping and/or localization approaches were addressed in this dissertation and the Processing Unit executes the required ROS package. The outputs are the robot pose (x, y, θ) and an updated map of the environment (2.5D or 3D depending on the evaluated approach).

5.2.2.2 Joystick Package

The Joystick package is a module able to send speed commands to the platform. This module receives as input the current state of each of the joystick's axes and buttons and converts them into speed commands.

5.3 Software Design

In order to ensure the correct operation of the InterBot, it is required to establish a connection between the Base Station and the Processing Unit, i.e. use the ROS system on both machines. The master node is running on the Base Station along with the joystick node and the remaining remote nodes are running on the Processing unit. It is necessary to use export ROS_MASTER_URI on the Processing unit to enable the nodes to use the master node executed on the other machine. The Processing unit runs several nodes required for the platform's operation, as well as for the mapping and localization approaches.

5.3.1 Physical Layer

Fig. 5.6 represents the Physical Layer module of the Base Station and Processing Unit. The ROS community provides the software drivers for Velodyne LiDAR, Hokuyo Laser Scanner and Joystick controller. The **cloud_node** reads data from the /velodyne_packets topic,

converts to the sensor_msgs/PointCloud2 format, and publishes to the /velodyne_points topic. The **hokuyo_node** publishes a /scan topic message of the obtained laser scanner data. The **joy** node allows to interface with a generic joystick (e.g., PS3-like gamepad), publishing a /joy topic containing information about the axes and buttons. Each time a /joy topic message is published, it is converted into a speed message (/cmd_vel). The **RoboteQ_node** subscribes the /cmd_vel topic to move the InterBot and this motion is expressed through the odometry data that is published in the/odom topic.

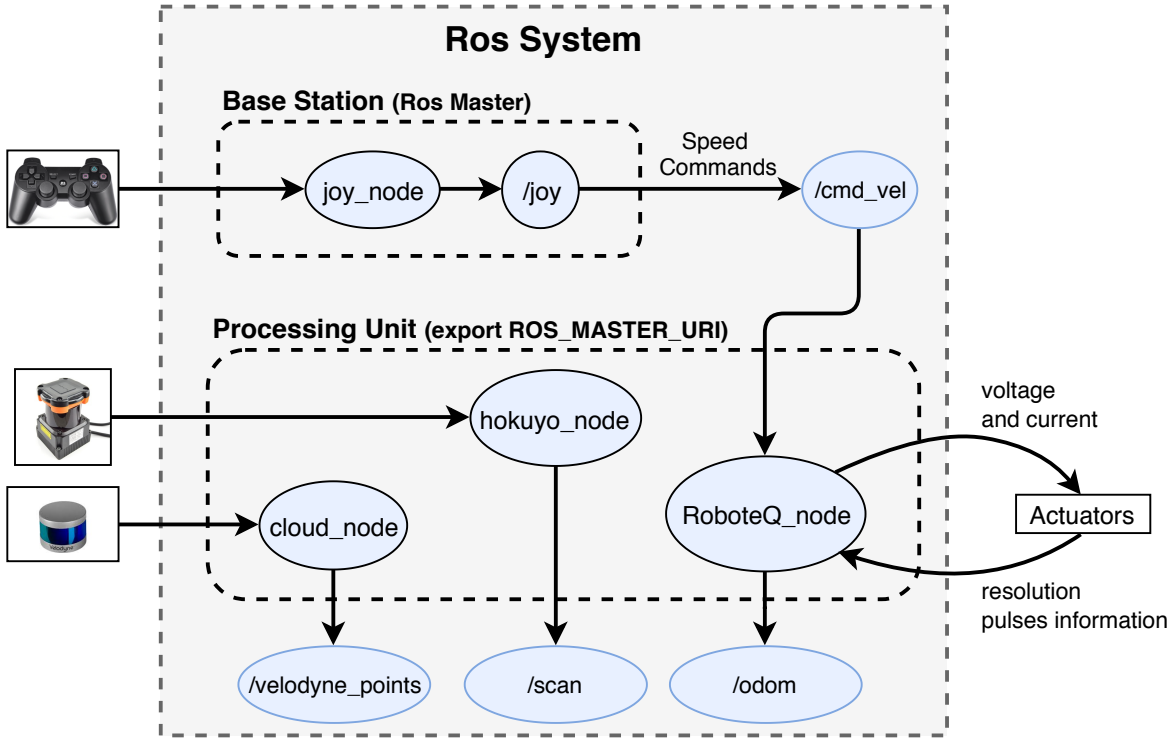


Figure 5.6: Physical Layer nodes, along with the subscribed and published topics.

Figures 5.7, 5.8 and 5.9, show the messages that are used in each of the proposed approaches.

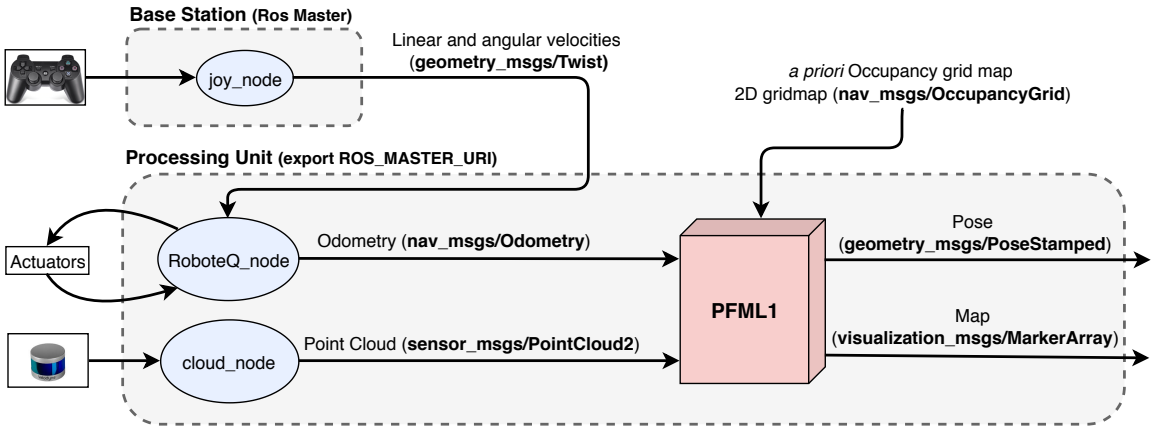


Figure 5.7: ROS schematic for the 2.5D Mapping and Localization approach, which contains the ROS nodes used and the messages type.

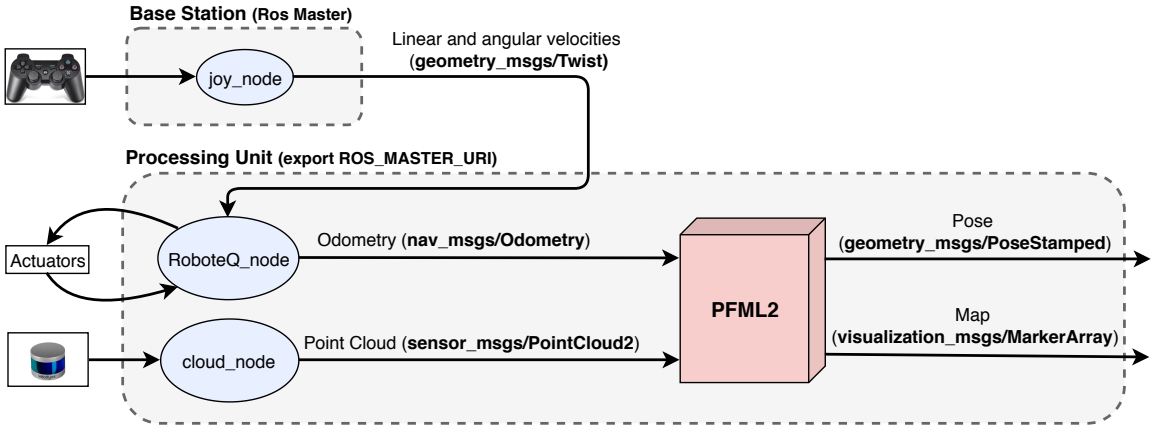


Figure 5.8: ROS schematic for the 3D Localization and Mapping (SLAM) approach, which contains the ROS nodes used and the messages type.

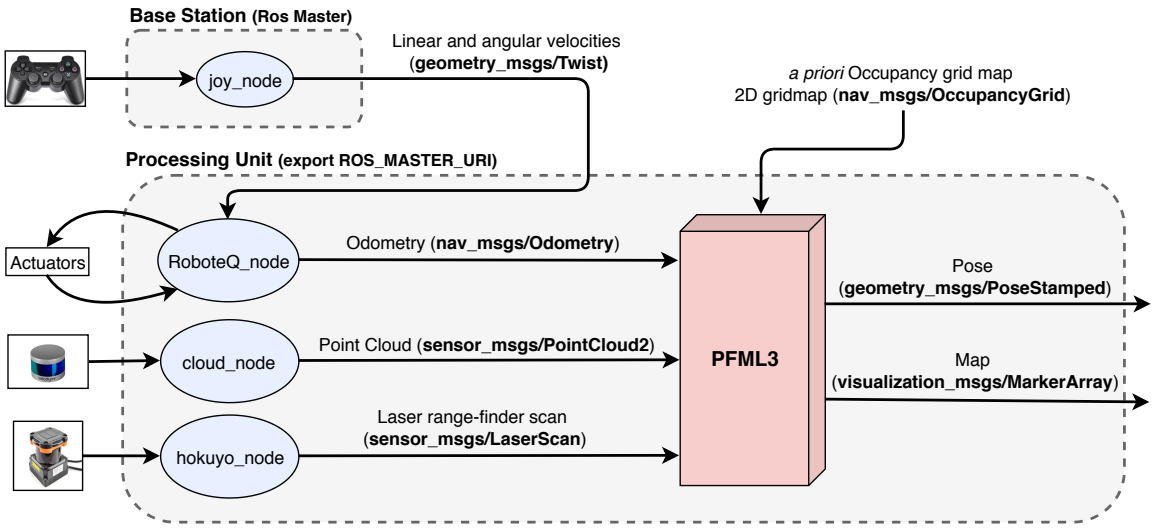


Figure 5.9: ROS schematic for the 3D Mapping and 2D/3D Localization approach, which contains the ROS nodes used and the messages type.

Chapter 6

Experimental Results

In this chapter the experimental results of each approach in two different scenarios and the correspondent analysis are presented. The scenarios selected for the validation results were the ISR Shared Experimental Area (scenario 1) and the floor 0 of the ISR-UC building (scenario 2). The experiments include:

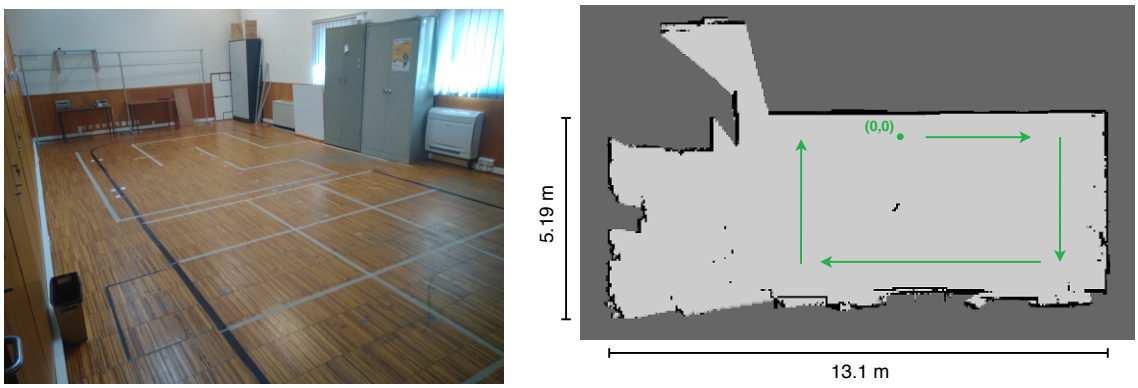
1. **2.5D Mapping and Localization:**
   (a) Without map update: geometric path and 2.5D score;
   (b) With map update: geometric path and 2.5D map;
2. **3D Localization and Mapping (SLAM):**
   (a) Without optimization: geometric path and 3D map;
   (b) With optimization: geometric path and 3D map;
3. **3D Mapping and 2D/3D Localization:**
   (a) With map update: geometric path, 2D score and 3D map.
   (b) Without map update: geometric path and 3D score.

In order to test the different approaches, a dataset was recorded for each scenario using a ROS tool (rosbag file, *i.e* ".bag" file), which saves data from the sensors.

In both scenarios the algorithms were tested with the following parameters: $\Delta\theta = 0.01$ rad; $\Delta d = 0.05$ m; $N_{min} = 100$ and $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = 0.2$. The parameters $\Delta\theta$ and Δd are used to control the execution of the KLD-based filter, while the error sampling parameters (α_1, α_2, α_3, α_4) assign more or less error to the prediction step where the new particles' states are estimated. The size of the local and global maps was defined as 25x25 m and 125x125 m respectively, and a resolution of 0.05 m per cell.

6.1 Scenario 1

Initially, a smaller and less complex environment was chosen to evaluate the performance of the developed approaches. Figure 6.1a shows the environment where the tests were performed. The executed trajectory is also represented in Fig. 6.1b by the green arrows.



(a) (b)

Figure 6.1: Illustration of the ISR Shared Experimental Area: (a) Picture of the ISRsea; (b) Occupancy grid map representing the ISRsea

The evaluation of the performance of each approach consisted in analyzing the influence of the maximum number of particles in the KLD-based filter: on the pose estimation, the map generated and the estimation of the localization score. The score is only valid for situations where the map is not updated and its calculation for each approach will be explained.

6.1.1 2.5D Mapping and Localization

For this approach, it was observed the performance of the pose estimation with and without map update. In the situation where the map is not updated, the score is determined based on Algorithm 7. The score is given by:

$$s^{[i]} = \mathcal{N}_x(0,\ pt_x^i - x_c^i) \cdot \mathcal{N}_y(0,\ pt_y^i - y_c^i), \quad with \ \ 0 < i \leq N_v \tag{6.1}$$

where N_v is the number of voxels.

The 2.5D localization score is given by:

$$S = \frac{\sum_{i=1}^{N_v} s^{[i]}}{N_v} \times 100, \quad with \ \ 0 \leq S \leq 100 \tag{6.2}$$

Figure 6.2 shows the scores obtained according to the maximum number of particles. The analysis shows that the best result is obtained using 1800 particles presenting the highest mean value (70.27 %) and the lowest standard deviation value (11.78 %).

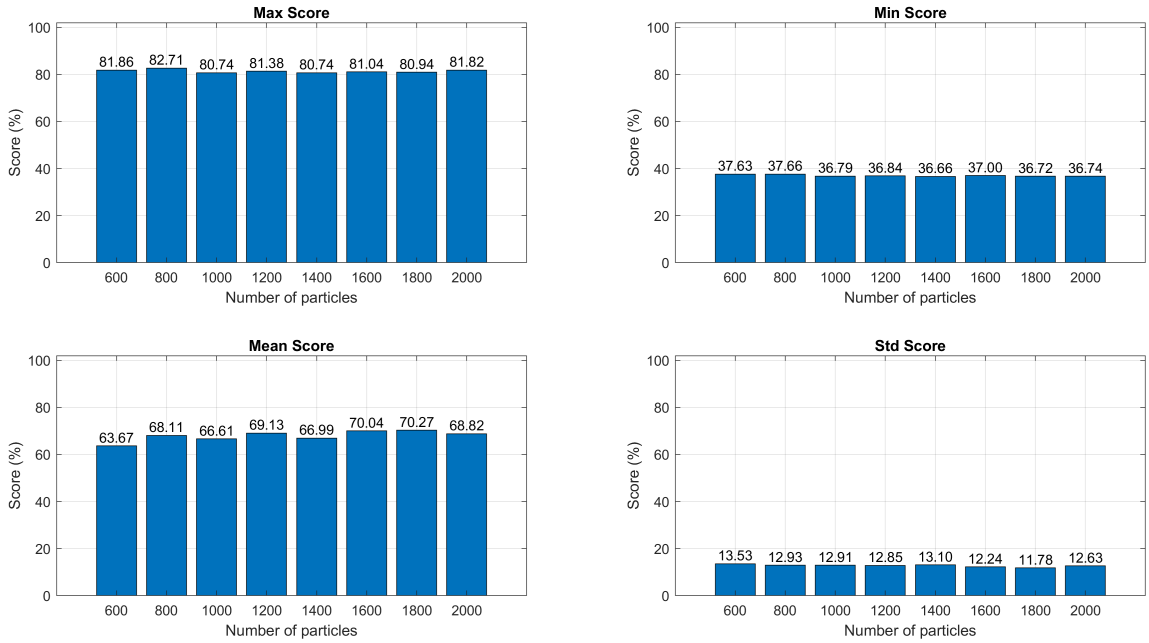


Figure 6.2: Bar graph with the maximum, minimum, mean and standard deviation of the 2.5D localization score for each number of particles used.

The maps obtained when using the update for the respective maximum number of particles are shown in Fig. 6.3.

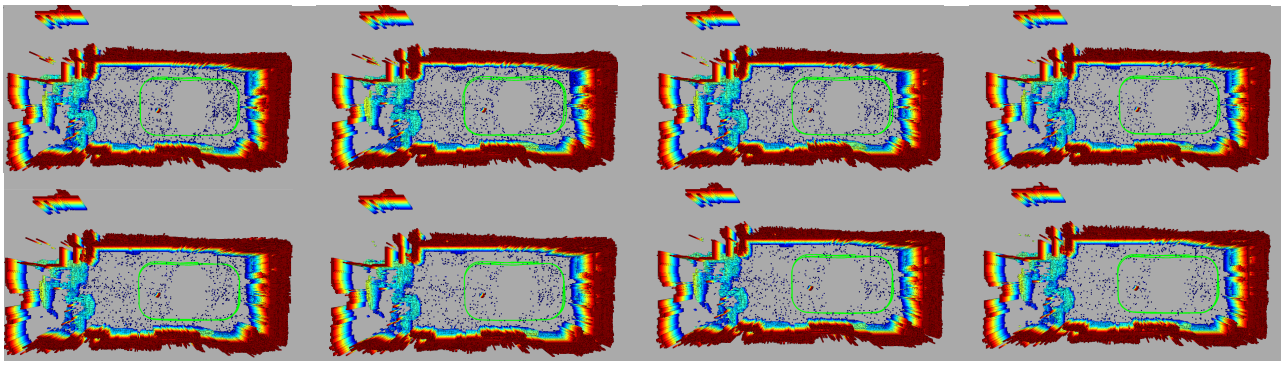


Figure 6.3: Representation of the geometric paths and corresponding maps from the 2.5D Mapping and Localization approach with map update, changing the maximum number of particles defined for the KLD-based filter. From left to right and top to bottom: 600, 800, 1000, 1200, 1400, 1600, 1800 and 2000 particles.

As shown in Fig. 6.3, there are no considerable differences in the 2.5D maps obtained when using different numbers of particles. In all the maps, it is possible to notice that the doorway is marked as occupied due to one of the shortcoming of this representation. This could be solved by defining a height threshold in the 3D point cloud, but the maps obtained would have less definition. Taking into account the results obtained, it would be possible to obtain a good estimation of the pose with only 600 particles.

In this approach, an *a priori* 2D map was given, which does not contain height information, neither obstacles with higher height (e.g., table, chairs). In Fig. 6.4, the geometrical paths show some variations when the map is not updated. This is because the 2D map was transformed into a 2.5D map, which does not contain height information from obstacles and since the map is not updated, there is not a good matching between the local and global map during the computation of the particles' weight. In Fig. 6.4, it is also possible to see that the geometrical paths obtained with map update better represent the path performed by the robot. With an updated map, it leads to a better adjustment of the particles' weights on the KLD-based filter, and consequently a better estimation of the pose. In general, the obtained results were satisfactory.

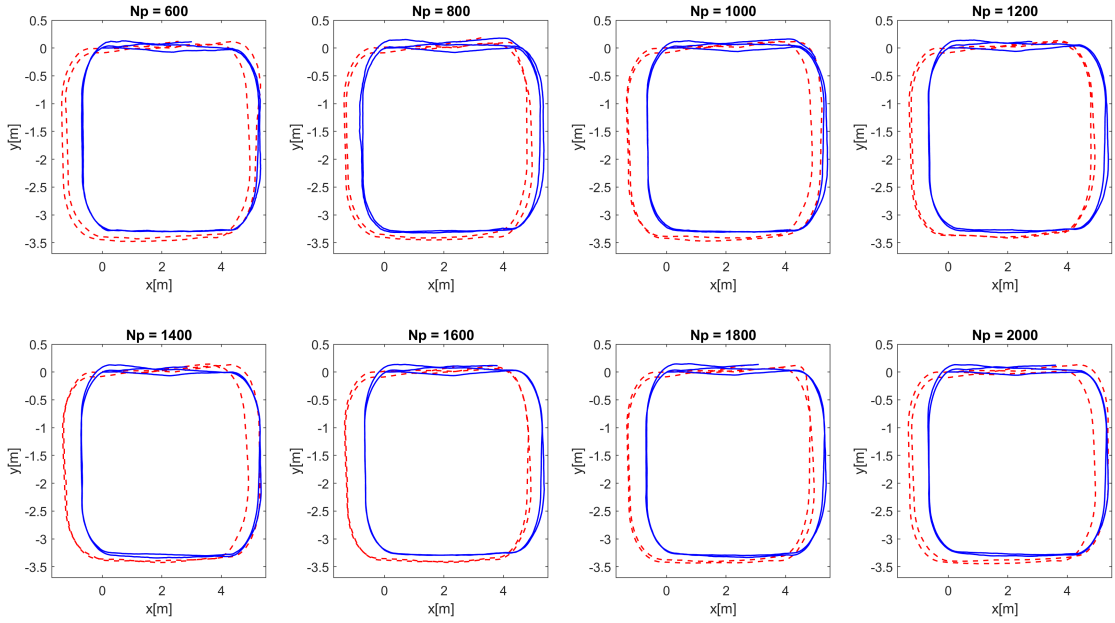


Figure 6.4: Representation of the geometric paths obtained without (in dashed red) and with map update (in blue). N_p denotes the maximum number of particles used.

6.1.2 3D Localization and Mapping (SLAM)

Since this approach contains an optimization block that corrects the estimated pose, the performance of the algorithm without that block was tested. A score is not computed since the map is constantly updated over time. The purpose of the test is to evaluate the pose estimated by the filter according to the maximum number of particles, as well as the influence of the optimization block in that estimation. The resulting geometric paths from these tests can be seen in Fig. 6.5. The results from tests with the optimization block present better pose estimates than the results without it.

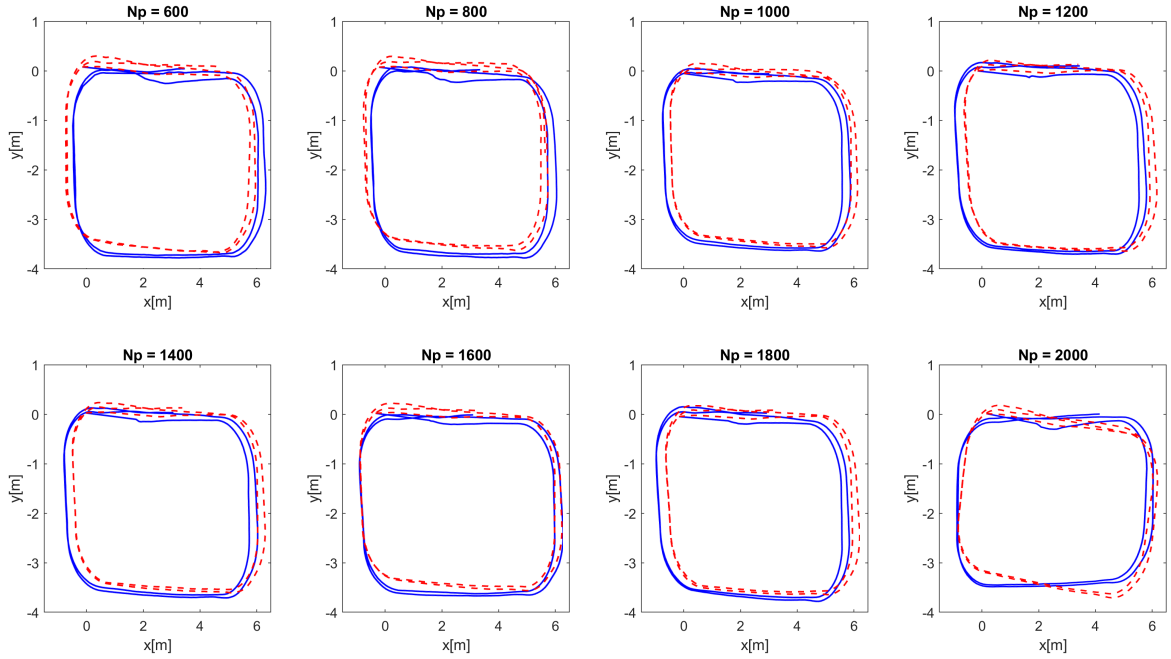


Figure 6.5: Representation of the geometric paths obtained without (in dashed red) and with the optimization block (in blue). N_p denotes the maximum number of particles used.

The environment representations and the respective geometric path are illustrated in Figs. 6.6 and 6.7. As can be observed in Fig. 6.6, the best result was obtained for 1600 particles, although the maps obtained are quite similar.

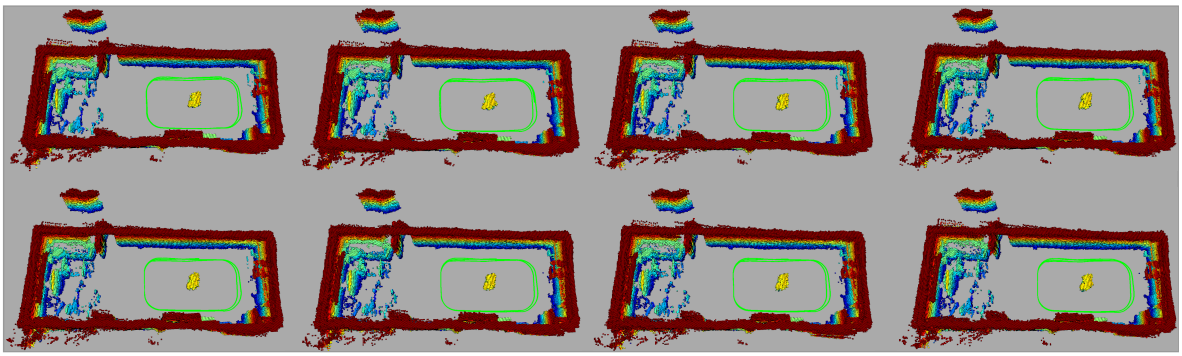


Figure 6.6: Representation of the geometric paths and corresponding maps from the SLAM approach without the optimization block, based on the maximum number of particles defined for the KLD-based filter. From left to right and top to bottom: 600, 800, 1000, 1200, 1400, 1600, 1800 and 2000 particles.

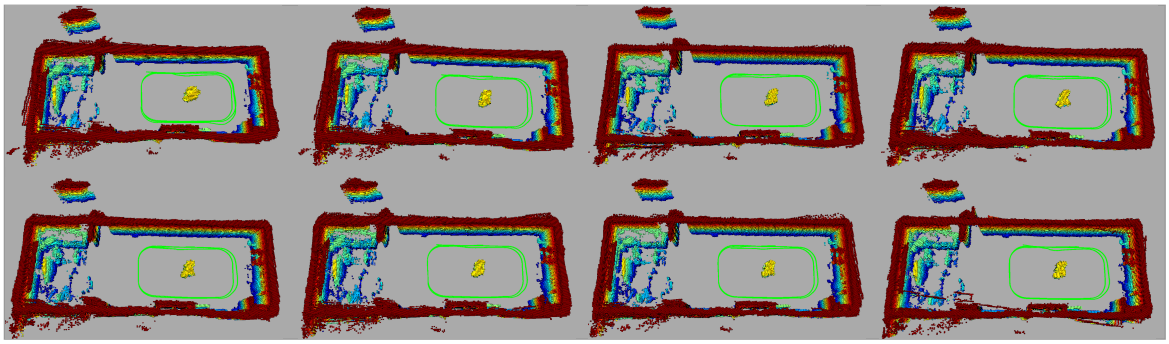


Figure 6.7: Representation of the geometric paths and corresponding maps from SLAM approach with the optimization block, based on the maximum number of particles defined for the KLD-based filter. From left to right and top to bottom: 600, 800, 1000, 1200, 1400, 1600, 1800 and 2000 particles.

Comparing Figs. 6.6 and 6.7, it is possible to observe that the best results were obtained when using the optimization block. The resulting 3D maps present a better definition, as it was expected once this block minimizes the alignment error between the map and the local map obtained from the 3D point cloud. By the analysis of Fig. 6.7, the best result was obtained when 1000 particles were used and a poorer one for 2000 particles. This may be caused by the time spent in computing the particles' weight and in the optimization process, leading to the point cloud data being an instant later. Since the scenario is small and not very complex, the contributions of the optimization block for the pose estimation were minimal. Finally, without the optimization block, the best result obtained was for 1600 particles. This number can be reduced with the introduction of the optimization block, reducing the computational complexity.

6.1.3 3D Mapping and 2D/3D Localization

As aforementioned, this approach uses 2D and 3D data to estimate the robot's pose. Since it requires an *a priori* 2D map and does not change over time, the laser scan data is used to compute a 2D localization score. The score is determined based on the first part of Algorithm 9. The score is given by:

$$s^{[i]} = \mathcal{N}_x(0, x^i_{hit} - x^i_{occ}) \cdot \mathcal{N}_y(0, y^i_{hit} - y^i_{occ}), \quad with \;\; 0 < i < N_m \tag{6.3}$$

where N_m denotes the number of measurements from the laser.

The 2D localization score is computed as follows:

$$S = \frac{\sum_{i=1}^{N_m} s^{[i]}}{N_m} \times 100, \quad with \;\; 0 \leq S \leq 100 \tag{6.4}$$

Figure 6.8 presents the results obtained for the score according to the maximum number of particles and Fig. 6.9 shows the correspondent maps.

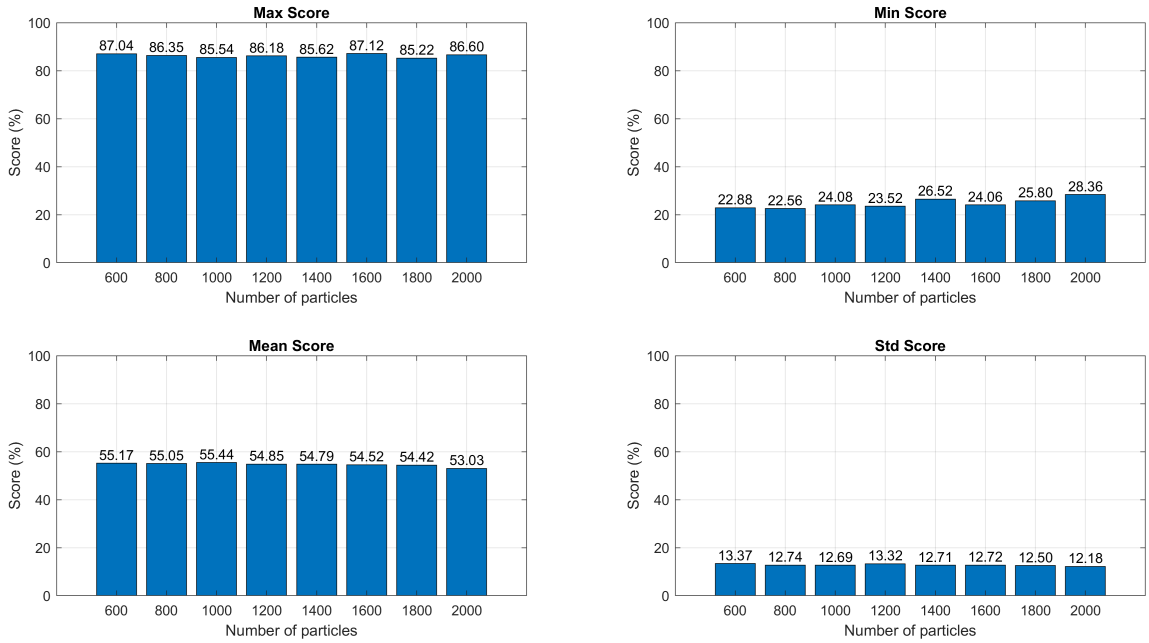


Figure 6.8: Bar graph with the maximum, minimum, mean and standard deviation of the 2D localization score for each number of particles used.

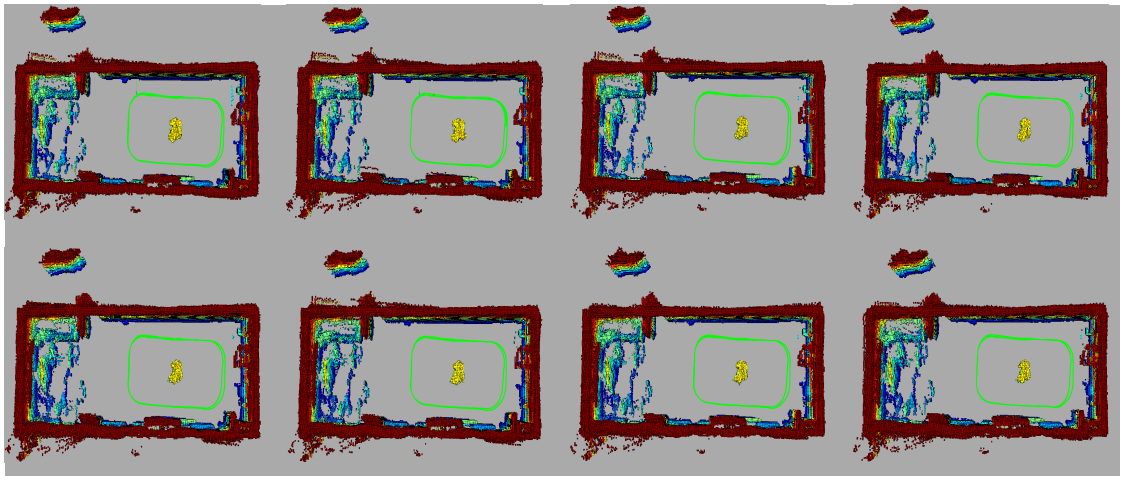


Figure 6.9: Representation of the geometric paths and corresponding 3D maps from 3D Mapping and 2D/3D Localization approach, based on the maximum number of particles defined for the KLD-based filter. From left to right and top to bottom: 600, 800, 1000, 1200, 1400, 1600, 1800 and 2000 particles.

In Fig. 6.9, it is possible to see that some maps have a small amount of noise, maybe by misaligned scans. In general, the 3D maps obtained were good and the approach showed stable results for a low number of particles.

The second part of the test was deployed using the 3D map from the previous test (with map update) and evaluate without updating. A 3D localization score was determined based

on the data from the 3D point cloud and the known 3D map. Considering the second part of the Algorithm 9 and N_v the number of voxels, the score for each measurement is given by:

$$s^{[i]} = \mathcal{N}_x(0,\ p_x^i - x_{occ}^i) \cdot \mathcal{N}_y(0,\ p_y^i - y_{occ}^i) \cdot \mathcal{N}_z(0,\ p_z^i - z_{occ}^i),\quad with\ \ 0 < i < N_v \tag{6.5}$$

The 3D localization score is determined by the following equation:

$$S = \frac{\sum_{i=1}^{N_v} s^{[i]}}{N_v} \times 100,\quad with\ \ 0 \leq S \leq 100 \tag{6.6}$$

Figure 6.10 shows the scores for each variation in the number of particles. The best result was obtained with 800 particles with the highest mean value (90.03%) and the lowest standard deviation value (6.99%), but in general the proposed approach is stable even with a low number of particles. For 1600 particles it was registered a break in the minimum score value, perhaps due to a peak in the processing of the approach, however, the influence on the final result was minimal.

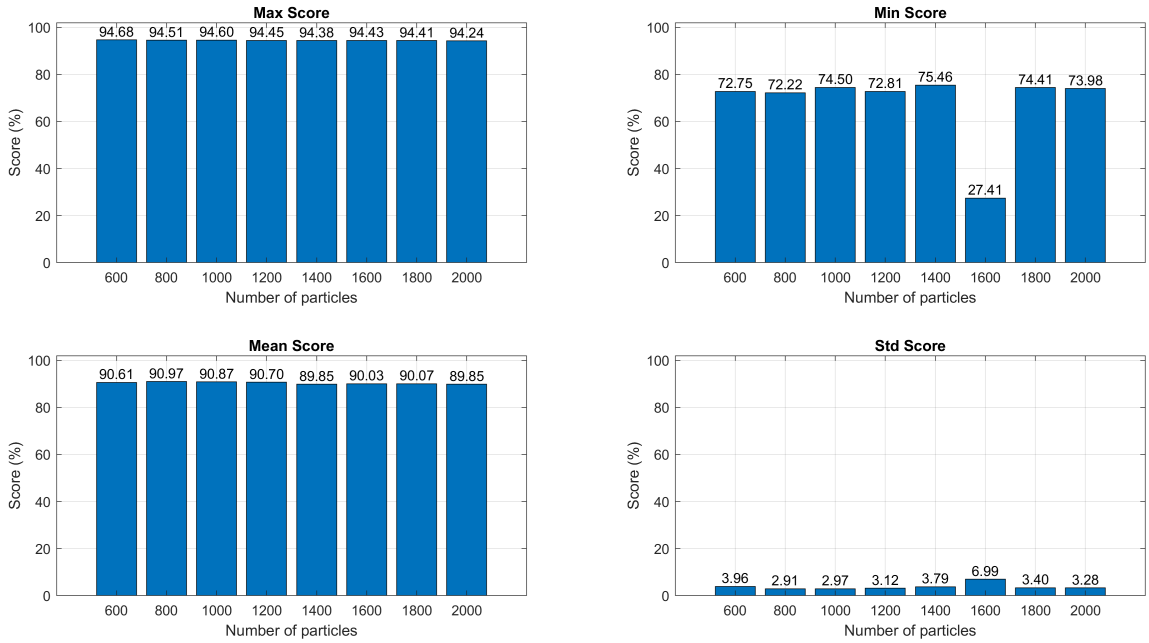


Figure 6.10: Bar graph with maximum, minimum, mean and standard deviation of the 3D localization score for each number of particles used.

Observing the Fig. 6.11, the geometric paths for the different number of particles were identical, however the obtained environment representation in some cases presented slightly differences due to orientation errors of the estimated pose.

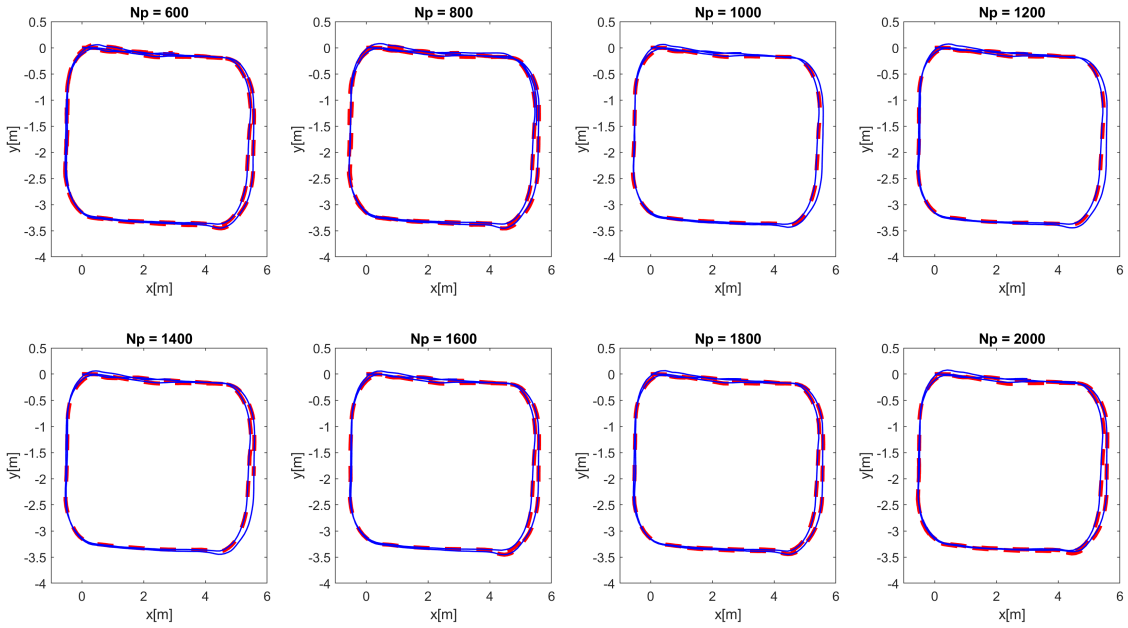


Figure 6.11: Representation of the geometric paths obtained without (in dashed red) and with map update (in blue). N_p denotes the maximum number of particles used.

6.2 Scenario 2

Once the tests were done in the ISR Shared Experimental Area, the next challenge was to evaluate the performance of the algorithms in a more complex environment. For this scenario, it was defined that the InterBot would make two laps, the first lap in a counterclockwise direction (see arrow in Fig. 6.12) and the second lap in a clockwise direction.

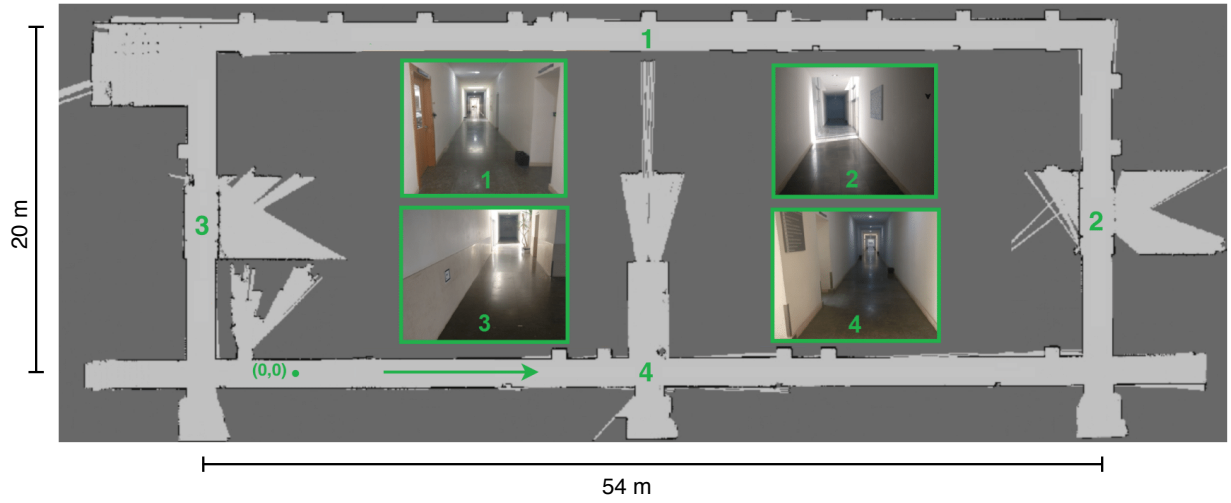


Figure 6.12: Occupancy grid map representing the floor 0 of the ISR-UC. The origin of the axis marked as the point (0,0). Each corridor contains a number with the respective image and the arrow indicates the direction of movement.

In this scenario when the dataset was recorded, in some situations the robot crossed with people, which added dynamic elements to the dataset.

In the following tests, the maximum number of particles in the KLD-based filter was not changed, the number was selected from the best results achieved in Section 6.1.

6.2.1 2.5D Mapping and Localization

The best result for the tests performed in the room was achieved using 1800 particles in the KLD-based filter, and therefore this was the value used to evaluate the behaviour of the algorithm in this environment. Similar to the room test, the situations without and with map update were analysed, and the score was calculated in the same way. As in the previous scenario, an *a priori* 2D map was given, and then transformed into 2.5D. The first test was without map update and a mean score value of 72.18% and a standard score deviation value of 9.57% were obtained.

The second test was with map update and the map obtained is shown in Fig. 6.13. Observing the figure, the geometric path obtained with the map update closely represents the path performed by the robot. Concerning the map, this one presents a accurate representation and closes the loop well.

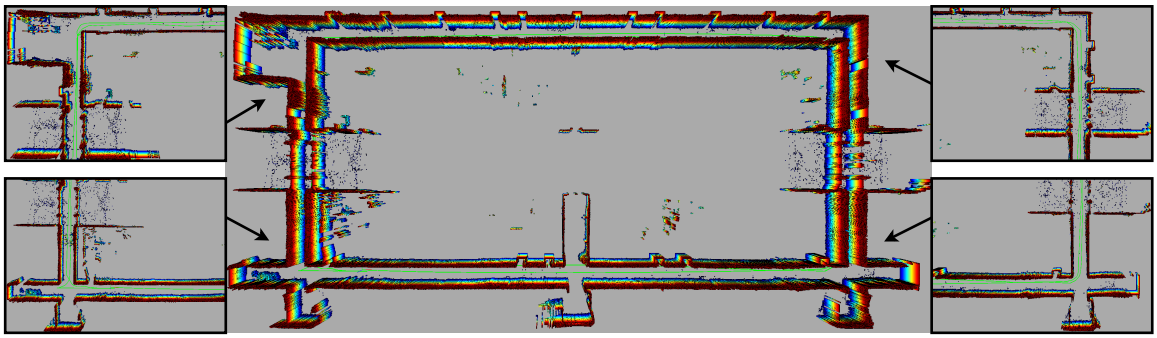


Figure 6.13: Illustration of the 2.5D map generated using 1800 particles in the KLD-based filter for the 2.5D Mapping and Localization approach. The curves have been zoomed for a better visualization

The geometric paths obtained without and with map update are represented in Fig. 6.14. As can be seen, both paths are similar which demonstrates the stability of both approaches. The zoomed part does not correspond to any pose estimation error, but to a failure in the robot's teleoperation, forcing it to move backwards.

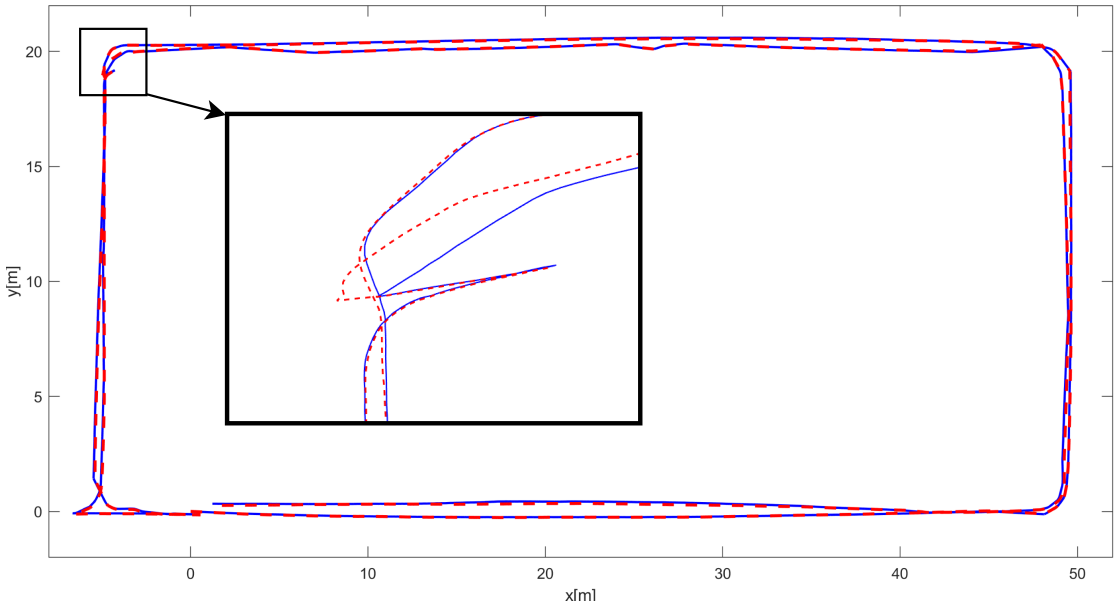


Figure 6.14: Representation of the geometric paths obtained without (in dashed red) and with map update (in blue) for 2.5D Mapping and Localization approach.

6.2.2 3D Localization and Mapping (SLAM)

For this approach, in the previous scenario, the best result was achieved when 1000 particles were used in the KLD-based filter, and therefore that was the maximum number of particles used to evaluate the performance without and with the optimization block. Figs. 6.15 and 6.16 present the maps without and with the integration of this block respectively.

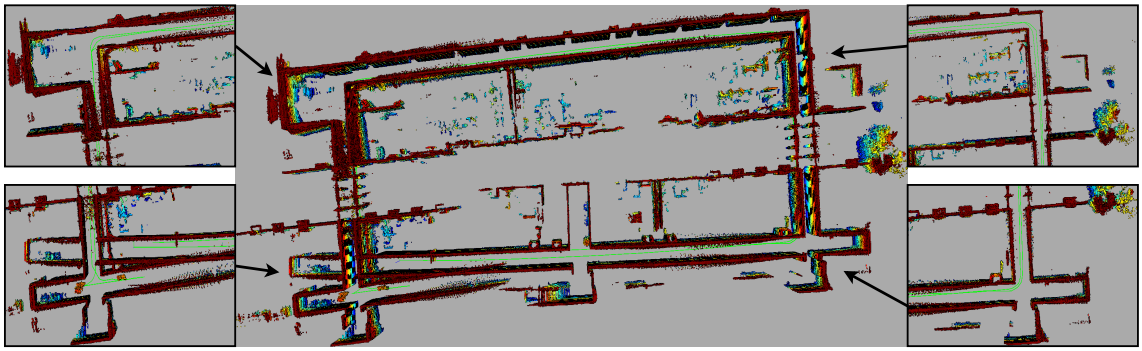


Figure 6.15: Illustration of the 3D map generated using 1000 particles in the KLD-based filter without optimization block. The curves have been zoomed for visualization purposes.

Analyzing Figs. 6.15 and 6.16, it can be seen that without the optimization block, the approach is unable to close the path correctly (loop closure) and consequently the map obtained appears misaligned.

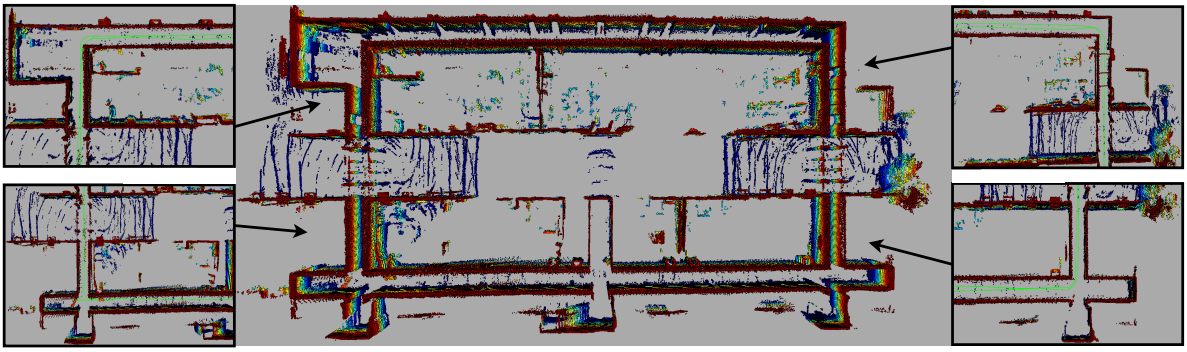


Figure 6.16: Illustration of the 3D map generated using 1000 particles in the KLD-based filter with optimization block. The curves have been zoomed for visualization purposes.

As can be seen in Figs. 6.16 and 6.17, a better estimation of the pose was obtained when comparing with the pose obtained without the optimization block. This approach, even not implementing loop closure detection, can reduce the estimation error of the pose enough to successfully close the map. In the lower right corner of the map it is possible to notice some noise in the corridor, but looking at Fig. 6.18 the corridor is well represented, only with a small amount of noise on the ceiling by the fact that Velodyne sensor was only of 16 channels and the approach was not able to update that part so well. Also, due to the fact that it has only 16 channels, some gaps on the map may be visible (see Fig. 6.18).

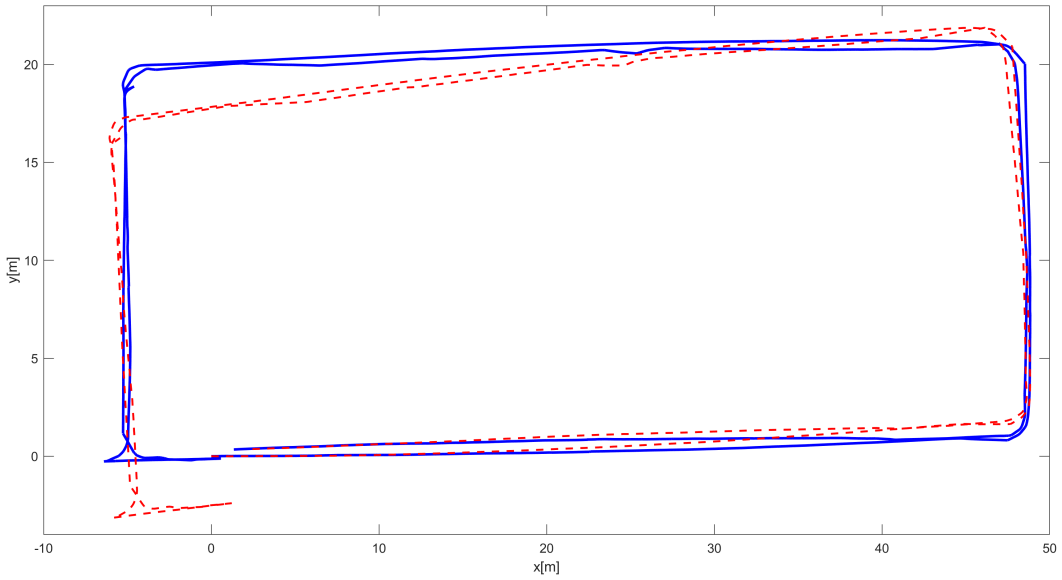


Figure 6.17: Representation of the geometric paths obtained without (in dashed red) and with optimization block (in blue).

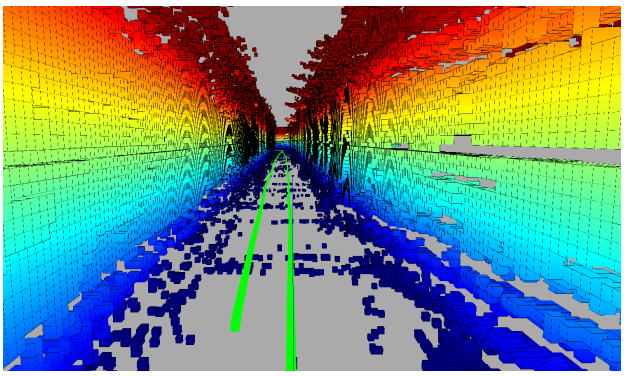


Figure 6.18: Illustration of the robot's point of view in a corridor area.

6.2.3 3D Mapping and 2D/3D Localization

The maximum number of particles used in this scenario was 1200, since it was the best result that was obtained in the previous scenario. The two situations were also tested, with and without map update and determined the respective score. In the first test, with map update, a mean score value of 95.77% and standard deviation value of 11.67% were obtained. The environment representation obtained for this test is shown in Fig. 6.19. In general, the generated map shows a satisfactory representation of the environment, containing a small amount of noise in all curves. This may be related to abrupt changes in the robot's orientation, causing the PF to diverge slightly and the estimated pose to lose precision. One solution would be to adjust the parameters (α) of the motion model.

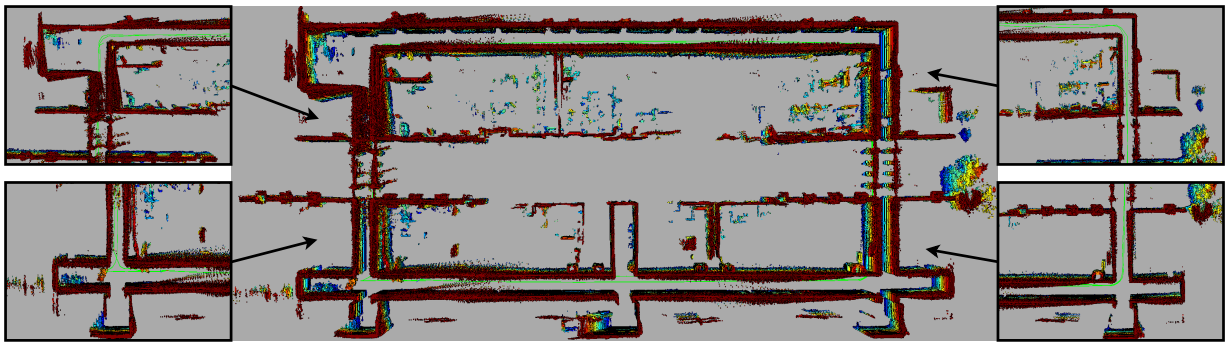


Figure 6.19: Illustration of the 3D map generated using 1200 particles in the KLD-based filter for the 3D Mapping and 2D/3D Localization approach. The curves have been zoomed for visualization purposes.

The second test, as in scenario 1, consisted in not updating the map and using an *a priori* 3D map obtained in Fig. 6.19. The geometric paths presented in Fig. 6.20, closely represents the path performed by the robot. The 3D score was determined and a mean value of 90.21% and a standard deviation of 4.56% were obtained. Both scores (2D and 3D) show that a good estimation of the robot's pose was obtained.

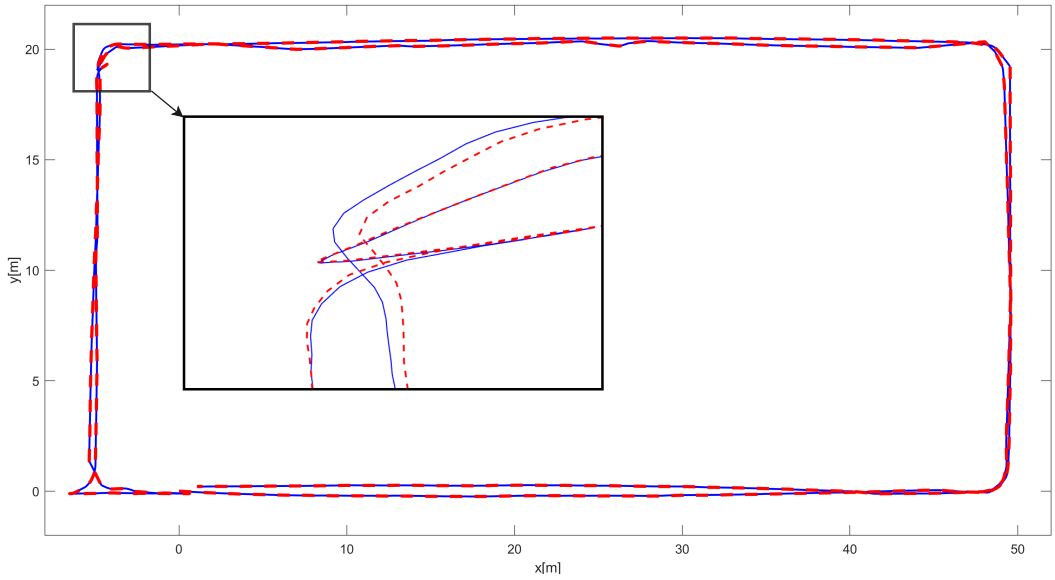


Figure 6.20: Representation of the geometric paths obtained without (in dashed red) and with map update (in blue) for 3D Mapping and 2D/3D Localization approach.

The results obtained for these scenarios for each of the tested approaches show a very similar geometrical path.

Chapter 7

Conclusion and future work

7.1 Conclusion

The focus of this dissertation was to develop and study an indoor localization and mapping approach using different sensory data (2D and 3D) on a PF for mobile robots (in this case, the InterBot). To reach this goal two more approaches (2.5D Mapping and Localization, and 3D SLAM) were developed, and all of them were tested in different scenarios. Some conclusions can be drawn from the work developed in this dissertation.

All the proposed approaches have shown similar results. Despite not presenting the execution times of the proposed approaches, the 2.5D Mapping and Localization approach presents a lower computational burden due to the simplicity of its representation.

The results obtained for the 2.5D Mapping and Localization approach show that this approach provides a good representation of the environment and successfully locates the robot on the 2.5D map. Moreover, the results are consistent when considering the tests with and without the update of the 2.5D map. In the 3D SLAM approach, the results clearly show that only when the optimization block is used, a correct representation of the environment, and therefore a better estimation of the pose, can be obtained. Regarding the 3D Mapping and 2D/3D Localization approach, the results obtained show that a 3D representation of the environment can be constructed (and used for the localization of the robot) from a localization approach with an *a priori* 2D map. Moreover, a good estimation of the robot's pose is obtained when a laser scan and a 3D point cloud are used as inputs.

In this work, solutions with different characteristics were proposed, which can be chosen by considering different constraints. The selection of the approach to be used will always depend on the application/context. For scenarios where there is already previous knowledge of the map (2D map) and a computationally lighter approach is needed, then the 2.5D Mapping and Localization approach with map update can be used. When there is no processing power

constraints, meaning that a more faithful representation of the environment can be explored, the solution is to implement the 3D Mapping and 2D/3D Localization approach. If there is no *a priori* knowledge of the environment, the 3D SLAM solution with optimization can be used.

In light of the results achieved in this dissertation, it is concluded that the objectives were achieved.

7.2 Future work

To increase the performance of this work, some topics could be further researched, for example:

- **3D Mapping and 2D/3D Localization approach:** making a smart switch between 2D and 3D data during the update step of PF. For areas such as corridors, the pose estimation using 2D data may be enough to use while areas such as offices with a lot of features, it may be better to use 3D data for the pose estimation.
- **Map update:** improve the map updating to deal with dynamic changes in the environment (e.g. human beings, mobile robots, etc.)
- **PF optimization:** decrease the processing time on each iteration of the filter ($\simeq 200ms$). Possible directions: smarter selection of selected voxels and multi-threading (e.g. CUDA, OpenMP or OpenCL).
- **Loop closure detection and place recognition:** implement a loop closure detection algorithm, to detect when the robot is in a previously visited location, and then update the map improving the pose estimate.

Bibliography

[1] Sebastian Thrun and Arno Bücken. Integrating grid-based and topological maps for mobile robot navigation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume2.*, pages 944–950, 1996.

[2] Martial Hebert. Terrain modeling for autonomous underwater navigation. In *Proceedings of the 7th International Symposium on Unmanned Untethered Submersible Technology*, June 1989.

[3] Cristiano Premebida, João Sousa, Luis Garrote, and Urbano Nunes. Polar-grid representation and kriging-based 2.5D interpolation for urban environment modelling. *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, 2015.

[4] L. Garrote, C. Premebida, D. Silva, and U. J. Nunes. HMAPs - Hybrid Height-Voxel Maps for environment representation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2018.

[5] Adrián Canedo-Rodriguez, Víctor Alvarez-Santos, Carlos V. Regueiro, Roberto Iglesias, Senén Barro, and Jesús María Rodríguez Presedo. Particle filter robot localisation through robust fusion of laser, wifi, compass, and a network of external cameras. *Information Fusion*, 27:170–188, 2016.

[6] Takoua Grami and Ali Sghaier Tlili. Indoor mobile robot localization based on a particle filter approach. *2019 19th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, 2019.

[7] Patricia Javierre, Biel Piero E. Alvarado Vasquez, and Paloma de la Puente. Particle filter localization using visual markers based omnidirectional vision and a laser sensor. *2019 Third IEEE International Conference on Robotic Computing (IRC)*, 2019.

[8] Miguel Torres. Exploiting particle-filter based fusion in mobile robot localization. Master's thesis, University of Coimbra, 2018.

[9] José A. Castellanos, José Mario Martínez, José Luis Hernández Neira, and Juan D. Tardós. Experiments in multisensor mobile robot localization and map building. 1998.

[10] Michael Bosse, Paul Newman, John Leonard, and Seth Teller. Simultaneous localization and map building in large-scale cyclic environments using the atlas framework. *The International Journal of Robotics Research*, 23(12):1113–1139, 2004.

[11] Andrew J. Davison, Yolanda González Cid, and Nobuyuki Kita. Real-time 3D SLAM with wide-angle vision. 2004.

[12] J. Folkesson and H. Christensen. Graphical SLAM - a self-correcting map. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, April 2004.

[13] José E. Guivant and Eduardo Mario Nebot. Optimization of the simultaneous localization and map-building algorithm for real-time implementation. *IEEE Trans. Robotics and Automation*, 17:242–257, 2001.

[14] Stefan Williams, Paul Newman, Gamini Dissanayake, and Hugh Durrant-Whyte. Autonomous underwater simultaneous localisation and map building. volume 2, pages 1793 – 1798 vol.2, 02 2000.

[15] Paul Newman and John J. Leonard. Pure range-only sub-sea SLAM. *2003 IEEE International Conference on Robotics and Automation*, 2003.

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

[17] Yuval Roth-Tabak and Ramesh Jain. Building an environment model using depth information. *IEEE Computer*, 22(6):85–90, 1989.

[18] Julian Ryde and Huosheng Hu. 3D mapping with multi-resolution occupied voxel lists. *Auton. Robots*, 28(2):169–185, February 2010.

[19] I. Dryanovski, W. Morris, and J. Xiao. Multi-volume occupancy grids: An efficient probabilistic 3D mapping model for micro aerial vehicles. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2010.

[20] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3D mapping framework based on octrees. *Auton. Robots*, 34(3):189–206, April 2013.

[21] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[22] Feng Lu and Milios. Robot pose estimation in unknown environments by matching 2D range scans. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, June 1994.

[23] Simon J. Julier and Jeffrey K. Uhlmann. A New Extension of the Kalman Filter to Nonlinear Systems. *Proc. SPIE*, 3068, 1999.

[24] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Monte carlo localization for mobile robots. In *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1999.

[25] Dieter Fox. Adapting the sample size in particle filters through KLD-sampling. *The International Journal of Robotics Research*, 22(12):985–1003, 2003.

[26] A. Garulli, A. Giannitrapani, A. Rossi, and A. Vicino. Mobile robot SLAM for line-based environment representation. In *Proceedings of the 44th IEEE Conference on Decision and Control*, Dec 2005.

[27] Gert Kootstra, Sjoerd de Jong, and Daniel Wedema. Comparing the EKF and FastSLAM solutions to the problem of monocular simultaneous localization and mapping. 2009.

[28] João Santos, David Portugal, and Rui Rocha. An evaluation of 2D SLAM techniques available in robot operating system. In *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2013*, 2013.

[29] Mathieu Labbé and François Michaud. RTAB-Map as an Open-source LiDAR and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, 36(2):416–446, 2019.

[30] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, Feb 2007.

[31] Stefan Kohlbrecher, Oskar Von Stryk, Technische Universität Darmstadt, Johannes Meyer, and Uwe Klingauf. A flexible and scalable SLAM system with full 3D motion estimation. In *International Symposium on Safety, Security, and Rescue Robotics. IEEE*, 2011.

[32] W. Hess, D. Kohler, H. Rapp, and D. Andor. Real-time loop closure in 2D LiDAR SLAM. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016.

[33] Mathieu Labbe and François Michaud. Appearance-based loop closure detection for online large-scale and long-term operation. *Robotics, IEEE Transactions on*, 29:734–745, 06 2013.

[34] Ji Zhang and Sanjiv Singh. LOAM: LiDAR Odometry and Mapping in Real-time. In *Robotics: Science and Systems*, 2014.

[35] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[36] M. Bennewitz, C. Stachniss, S. Behnke, and W. Burgard. Utilizing reflection properties of surfaces to improve mobile robot localization. In *2009 IEEE International Conference on Robotics and Automation*, May 2009.

[37] Piotr Kozierski, Marcin Lis, and Joanna Zietkiewicz. Resampling in particle filtering - comparison. *Studia z Automatyki i Informatyki*, 38:35–64, 01 2013.

[38] T. Li, M. Bolic, and P. M. Djuric. Resampling methods for particle filtering: Classification, implementation, and strategies. *IEEE Signal Processing Magazine*, 32(3):70–86, May 2015.

[39] L. Garrote, T. Barros, R. Pereira, and U. J. Nunes. Absolute indoor positioning-aided laser-based particle filter localization with a refinement stage. In *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, Oct 2019.

[40] R. Cruz, L. Garrote, A. Lopes, and U. J. Nunes. Modular software architecture for human-robot interaction applied to the interbot mobile robot. In *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, April 2018.

[41] Velodyne LiDAR VLP-16 user manual. `https://velodynelidar.com/downloads.html`.

[42] Hokuyo. *Scanning Laser Range Finder UTM-30LX/LN Specification.* 2008.

[43] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[44] E.K.P. Chong and S.H. Zak. *An Introduction to Optimization*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 2013.

[45] Xiongwei Liu and Kai Cheng. Three-dimensional extension of bresenham's algorithm and its application in straight-line interpolation. *Proceedings of The Institution of Mechanical Engineers Part B-journal of Engineering Manufacture - PROC INST MECH ENG B-J ENG MA*, 216:459–463, 03 2002.

Appendix A

Extra content

A.1 Optimization algorithms

Optimization algorithms [44] are executed iteratively to find an optimal solution. When applied to the localization module, it reduces the Scan Matching error, i.e., the alignment of the laser scan points with the grid map. Some optimization algorithms such as Gauss-Newton, Gradient descent, and Levenberg–Marquardt are described below.

- **Gauss-Newton:** implemented in HectorSlam [31] during the scan matching process to find a rigid transformation $\xi = (p_x, p_y, \psi)$ that minimizes

$$\xi^* = \underset{\xi}{argmin} \sum_{i=1}^{n} \left[1 - M(S_i(\xi))\right]^2 \tag{A.1}$$

  with $S_i(\xi)$ the coordinates (world reference system) of scan endpoints $s_i = (s_{i,x}, s_{i,y}^T)$ and $M(S_i(\xi))$ denotes the map value for the coordinates $S_i(\xi)$. This minimization is solved using the following Gauss-Newton equation:

$$\xi^{s+1} = \xi^s - \Delta\xi \tag{A.2}$$

$$\Delta\xi = H^{-1} J^T \left[1 - M(S_i(\xi))\right], \quad with \;\; H = J^T J \tag{A.3}$$

$$\xi^{s+1} = \xi^s - (J^T J) J^T \left[1 - M(S_i(\xi))\right] \tag{A.4}$$

- **Gradient descent:** a first-order iterative method that follows a direction given by the decreasing function $(F(x))$, i.e., the direction of the negative gradient $(-\nabla F(x))$ to find a minimum. At each iteration is associated with the term designated step (γ), which tends to be larger the further away the current point from the endpoint (minimum). Then follows

$$x_{n+1} = x_n - \gamma_n \nabla F(x_n), \quad n \geq 0 \tag{A.5}$$

- **Levenberg–Marquardt:** method that can be obtained by replacing A.3 with:

$$\Delta\xi = [J^T J + \lambda\ diag(J^T J)] J^T \left[1 - M(S_i(\xi))\right] \tag{A.6}$$

$$[J^T J + \underbrace{\lambda\ diag(J^T J)}_{LM}]\Delta\xi = J^T \left[1 - M(S_i(\xi))\right] \tag{A.7}$$

A.2 3D Line Tracing algorithm

Algorithm 10: 3D line tracing based on [45] to compute voxels between the sensor and the measured voxel.

Data: Grid coordinates of the robot pose ($x0,\ y0,\ z0$) and the measured voxel ($x1,\ y1,\ z1$).

```
1
2  dx ← abs(x1-x0); dy ← abs(y1-y0); dy ← abs(z1-z0);
3  if x0 < x1 then sx ← 1; else sx ← -1;
4  if y0 < y1 then sy ← 1; else sy ← -1;
5  if z0 < z1 then sz ← 1; else sz ← -1;
6  if dx >= dy && dx >= dy then
7      err1 ← dy - dx; err2 ← dz - dx;
8      for i = 0; i < dx - 1; i++ do
9          M ← voxelUpdate(x0, y0, z0, val);
10         if err1 > 0 then
11             y0 ← y0 + sy; err1 ← err1 - dx;
12         end
13         if err2 > 0 then
14             z0 ← z0 + sz; err2 ← err2 - dx;
15         end
16         err1 ← err1 + dy; err2 ← err2 + dz; x0 ← x0 + sx;
17     end
18 else if dy >= dx && dy >= dz then
19     err1 ← dx - dy; err2 ← dz - dy;
20     for i = 0; i < dy - 1; i++ do
21         M ← voxelUpdate(x0, y0, z0, val);
22         if err1 > 0 then
23             x0 ← x0 + sx; err1 ← err1 - dy;
24         end
25         if err2 > 0 then
26             z0 ← z0 + sz; err2 ← err2 - dy;
27         end
28         err1 ← err1 + dx; err2 ← err2 + dz; y0 = y0 + sy;
29     end
30 else
31     err1 ← dy - dz; err2 ← dx - dz;
32     for i = 0; i < dz - 1; i++ do
33         M ← voxelUpdate(x0, y0, z0, val);
34         if err1 > 0 then
35             y0 ← y0 + sy; err1 ← err1 - dz;
36         end
37         if err2 > 0 then
38             x0 ← x0 + sx; err2 ← err2 - dz;
39         end
40         err1 ← err1 + dy; err2 ← err2 + dx; z0 ← z0 + sz;
41     end
42 end
```