Dissertation presented to the Instituto Tecnológico de Aeronáutica, in partial fulfillment of the requirements for the degree of Master of Science in the Graduate Program of Electronic Engineering and Computer Science, Area of Systems and Control.

Leonardo Mariga

SOLVING THE SIMULTANEOUS LOCALIZATION AND 3D MAPPING PROBLEM IN MOBILE ROBOTICS USING MULTI-LEVEL PARAMETERIZED REPRESENTATIONS

Dissertation approved in its final version by signatories below:

Prof. Dr. Cairo Lúcio Nascimento Júnior
Advisor

Prof[a]. Dra. Emília Villani
Pro-Rector of Graduate Courses

Campo Montenegro
São José dos Campos, SP - Brazil
2021

Cataloging-in Publication Data
Documentation and Information Division

Mariga, Leonardo
Solving the Simultaneous Localization and 3D Mapping Problem in Mobile Robotics Using Multi-Level Parameterized Representations / Leonardo Mariga.
São José dos Campos, 2021.
123f.

Dissertation of Master of Science – Course of Electronic Engineering and Computer Science. Area of Systems and Control – Instituto Tecnológico de Aeronáutica, 2021. Advisor: Prof. Dr. Cairo Lúcio Nascimento Júnior.

1. EKF. 2. SLAM. 3. 3D. I. Instituto Tecnológico de Aeronáutica. II. Title.

BIBLIOGRAPHIC REFERENCE

MARIGA, Leonardo. **Solving the Simultaneous Localization and 3D Mapping Problem in Mobile Robotics Using Multi-Level Parameterized Representations**. 2021. 123f. Dissertation of Master of Science – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSION OF RIGHTS

AUTHOR'S NAME: Leonardo Mariga
PUBLICATION TITLE: Solving the Simultaneous Localization and 3D Mapping Problem in Mobile Robotics Using Multi-Level Parameterized Representations.
PUBLICATION KIND/YEAR: Dissertation / 2021

Leonardo Mariga
Av. Jorge Zarur, 431
– São José dos Campos–SP

SOLVING THE SIMULTANEOUS LOCALIZATION AND 3D MAPPING PROBLEM IN MOBILE ROBOTICS USING MULTI-LEVEL PARAMETERIZED REPRESENTATIONS

Leonardo Mariga

Thesis Committee Composition:

Prof[a] Dra.	Gabriela Werner Gabriel	Chairperson	-	ITA
Prof. Dr.	Cairo Lúcio Nascimento Júnior	Advisor	-	ITA
Prof. Dr.	Carlos Henrique Costa Ribeiro	Intern Member	-	ITA
Prof. Dr.	Eleri Cardozo	Extern Member	-	UNICAMP

ITA

To my parents Sigrun and Carlos.

Acknowledgments

I would like to thank my family for the support during turbulent times. My mother Sigrun, for the love and caring. My father Carlos, for the strength and example. My sister Caroline, for chatting (and for taking me on travels that I completely went out of my comfort zone). You are my foundation, even at a distance, with the COVID pandemic, kept me sane. You have my unconditional love and gratitude, thank you.

My sincerest gratitude for my advisor, Prof. Cairo Nascimento, for all teachings that were essential for implementing this work. Thank you for all meetings, talks and pointing me in the right direction in my studies.

I would like to thank my colleagues at the Laboratory of Intelligent Machines (ITA, Division of Electronic Engineering), Larissa Pinto, Walber Lima, Pedro Gava, Marcus Ferraz, Michel Leles, Clayton Rodrigues, Pedro Henrique de Jesus, Elton Sbruzzi and Luiz Eugênio S. Araújo Filho for the advices and research discussions.

My gratitude for all members from the 2019 ITA Aerodesign team for all unthinkable effort in bringing the champion's trophy to ITA and representing Brazil in the EAST competition. You helped me achieve a long time dream.

For my friends Henrique Patusco, Manuel Rodríguez, Lucas Guimarães, Guilherme Garcia, Wilson Marques, Rafael Lehmkuhl, Emily Moreno, Kevin Timmermann, Fernanda Reckziegel, João Dreveck, Camila Zandavalli, Amanda Deus and Tânia Hoffmann. You have no clue in how important you were for me. Each one of you deserve an entire paragraph of acknowledgments, but they will probably not going to allow me to do it.

Thanks Prof. Sérgio R. B. dos Santos (UNIFEI) for permission to use and adapt the Omni robot. Thanks to CNPq Foundation for the scholarship during the research.

"... such is the life of an adventurer."
— Hilda

Resumo

O uso de robôs móveis para mapear e navegar em ambientes desconhecidos ganhou bastante importância recentemente em aplicações militares, missões de busca e resgate e atividades domésticas. Este trabalho propõe uma solução para o problema de Localização e Mapeamento Simultâneo (SLAM) com expansão para 3D do tradicional SLAM baseado em marcos usando o Filtro de Kalman Estendido (EKF-SLAM). Este trabalho propõe um algoritmo que usa uma representação ponto e plano para mapear ambientes internos usando um modelo de 3 graus de liberdade de um robô móvel com uma câmera RGBD.

A solução apresentada nesse trabalho reconstrói um mapa de nuvem de pontos inicialmente simplificando a cena em cilindros e segmentos de planos. Em seguida, a nuvem de pontos é associada e segmentada ao seu marco no vetor de estados do EKF. O uso unificado de marcos parametrizados junto com a nuvem de pontos permite criar representações com diferentes níveis de abstração, o que pode ser valioso para aplicações com requisitos de memória limitados.

Este trabalho começa com uma revisão de pesquisas disponíveis na literatura. Em seguida, é especificada a estrutura EKF-SLAM usada para o trabalho. No método de extração de marcos, RANSAC é usado para segmentação de planos. Os pontos não pertencentes a planos são agrupados usando DB-SCAN e parametrizados como cilindros após passar por validações. Em cada etapa, o EKF propaga e atualiza os estados dos marcos. Isso também dispara o crescimento da nuvem de pontos para cada marco observado. Além disso, é feito um estudo comparativo de cinco diferentes representações de mapeamento 3D (nuvem de pontos, geometrias primitivas e células) em termos de uso de memória.

Os resultados da simulação mostram que o método conseguiu reconstruir mapas com sucesso, reduzindo o critério IAE do erro de trajetória para 15 % em comparação com a odometria. Por fim, validamos em um ambiente real, usando uma câmera Intel Realsense. Apesar de sofrer de limitações na câmera e no processo de extração de marcos, o trabalho provou ser um método forte para a compreensão da cena junto com as funcionalidades básicas do SLAM.

Abstract

The use of mobile robots for mapping and navigating unknown environments has recently gained importance in military applications, search and rescue missions and domestic activities. This work proposes a solution for the Simultaneous Location and Mapping (SLAM) problem using an expansion for the 3D environment of the landmark-based Extended Kalman Filter SLAM (EKF-SLAM). We propose an algorithm that uses a point-and-plane representation to map indoor environments using a 3-DoF model of a mobile robot with an RGBD camera.

Our solution reconstructs a point cloud map by initially simplifying the scene into parameterized features: cylinders and planar segments. Then, we segment and associate the point cloud to its specific EKF feature. The unified use of both parameterized features and point cloud enables the creation of maps using different representations and levels of abstraction, which is helpful when working in applications with limited memory requirements.

This work starts by reviewing the methods available in the literature. Then, we specify the EKF-SLAM framework used for this work. In the feature extraction method, we use RANSAC for planar segmentation. The non-planar points are clustered using DB-SCAN and parameterized as cylinders after passing through validation gates. On each step, the EKF framework propagates and updates the features' states. This also triggers the point-cloud growth and increments the point cloud of the observed feature. Additionally, we make a comparative study of five different 3D mapping representations (point cloud, primitive geometries, and voxels) in terms of memory usage.

The simulation results show that this method successfully reconstructs maps, reducing the IAE criteria to 15% of trajectory error compared to the odometry. Finally, we validate the algorithm in a real environment, using an Intel Realsense camera. Despite suffering from limitations in the camera and in the feature extraction process, the work proved to be a strong method for scene understanding along with the basic SLAM capabilities.

List of Figures

List of Tables

List of Abbreviations and Acronyms

IMU	Inertial Measurement Unit
GPS	Global Positioning System
SLAM	Simultaneous Localization and Mapping
RGBD	Red-Green-Blue-Depth
KF	Kalman Filter
EKF	Extended KF
RBPF	Rao-Blackwellized Particle Filter
RGB	Red-Green-Blue
BRIEF	Binary Robust Independent Elementary Features
ORB	Oriented FAST and rotated BRIEF
BRISK	Binary Robust Invariant Scalable Keypoints
SURF	Speeded up robust features
FAST	Features from accelerated segment test
ICP	Iterative Closest Point
RANSAC	Random Sample Consensus
G2O	General Framework for Graph Optimization
HOG-Man	Hierarchical Optimization for Pose Graphs on Manifolds
GTSAM	Georgia Tech Smoothing and Mapping
DoF	Degrees of Freedom
ROS	Robot Operating System
LMI	Laboratório de Máquinas Inteligentes (Laboratory of Intelligent Machines)
ITA	Instituto Tecnológico de Aeronáutica
MAR	Minimum Area Rectangle
DB-SCAN	Density-based spatial clustering of applications with noise
MD	Mahalanobis Distance
JCBB	Joint Compatibility Branch and Bound
IAE	Integral Absolute Error

List of Symbols

$\mathfrak{I}$	Inertial coordinate frame
$\mathfrak{B}$	Body coordinate frame (camera)
ξ	6-DoF position of the body coordinate frame
η	6-DoF orientation of the body frame
x_v	3-DoF body frame pose vector
u	Pose displacement measured from odometry
$\hat{x}$	EKF state vector
P	EKF covariance matrix
V	Process noise
W	Measurement noise
$z^{\#}$	Observed feature
$\mathcal{V}$	Total set of points of a measured RGBD point cloud
$\mathcal{V}_i$	Set of points inliers to a plane
$\mathcal{V}_o$	Set of points that don't belong to a plane
T_r	Planar RANSAC Threshold
L_{ds}	Plane downsample grid size
A_{min}	Minimum area for a valid plane
τ_{max}	Maximum measurement of spreadiness of cylinder feature

Contents

1 Introduction

1.1 Introduction

Navigation was once a fundamental problem for world exploration. When faced with the discovery of the new world, humankind had to invent means to write precise maps with limited tools available at the time. That primitive technology only enabled rough estimates of time, position, and speed, putting up a big challenge for the early engineers. Faced with those problems, we ended up with the development of tools such as clocks, compasses, and strategies like resectioning and triangulation (CORKE, 2013). It was only in the last century that a new set of tools was created: GPS, radars, cameras, IMUs, and many others that enabled humans to create precise maps.

In the industrial environment, robots are formidable solutions for automating repetitive processes. However, tasks that are usually considered trivial for humans, such as navigating throughout our surroundings, are not as easy for machines (ECK, 2013). A solution for navigation problems can be a game-changer for applications such as domestic activities, assistance for the elderly, surveillance, search and rescue missions, and high-risk military missions. In this perception, what was once a fundamental problem for humankind is now a fundamental problem in robotics known as SLAM (Simultaneous Location and Mapping).

A robot requires reliable estimates of movement to achieve full autonomy. It needs to understand the characteristics of the surrounded environment to safely navigate avoiding collisions. The classic navigation problem is divided into three research topics as shown in Figure 1.1.

SLAM is primarily focused on solving the intersection (I). Given a robot in an initially unknown environment, how is it possible to create a map of its surroundings and simultaneously know its pose? To better understand the complexity of this answer, imagine the following problem. When the map is precisely known, the main goal is to estimate the robot's pose. The robot can compare measurements from sensors such as cameras and lasers and associate it with the known map, solving a problem called **localization**. The other option is to have the robot's pose known, and build a map by appending a sequence

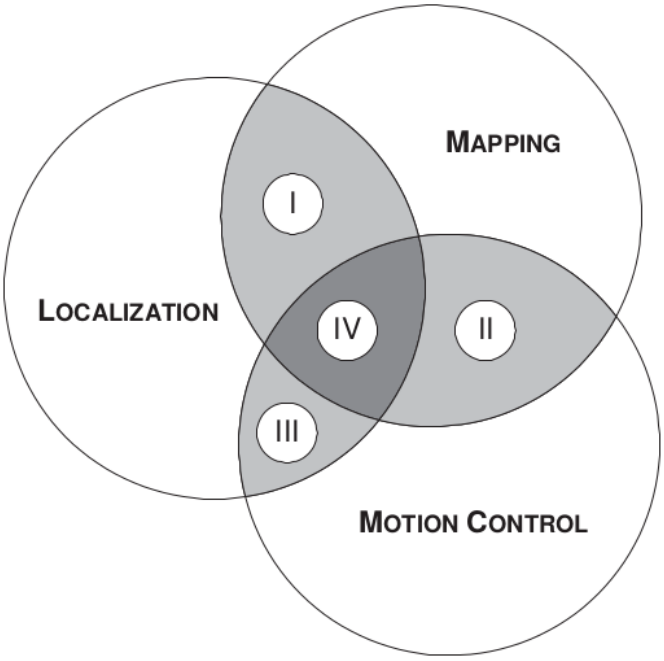


FIGURE 1.1 – The classical problem of navigation usually described as (I) SLAM, (II) Classic exploration, (III) Active localization, (IV) Integrated exploration (MAKARENKO *et al.*, 2002).

of measurements together, solving a problem called **mapping**.

Unfortunately, localization and mapping can only be disassociated in simulated environments. In real experiments the robot faces three challenges:

- The robot does not know its pose since odometry sensors have errors that accumulate over time.
- The robot does not know its map because measurements from the mapping sensor also are vulnerable to uncertainties.
- The robot is uncertain about which part of the map it is looking at all steps.

Hence, the nature of SLAM is known as a chicken-and-egg type of problem (CORKE, 2013), meaning it is necessary to estimate both map and robot's pose concurrently.

Methods for SLAM may require high computational power and high memory usage. For many years, a solution to this problem was restricted to 2D due to the limitations of 3D sensors available in the market. The 2D classical solutions achieve good results on the localization problem. This enables the robot to navigate in a planar movement avoiding collisions and creating an approximate map of obstacles. Nevertheless, the map creation is only bi-dimensional which does not provide enough realism for an application that requires a more humanized representation.

With RGBD (Red-Green-Blue-Depth) cameras and 3D laser sensors, it is now possible to map in a new dimension with reaseanable prices compared to the 3D lasers. The main solutions adopted for 3D SLAM today are based on raw point data, which is not memory efficient for some embedded applications. Even more, many 2D solutions for SLAM were not yet extensively explored when thinking of expanding it for 3D. That raises a question: how is it possible to expand the classical 2D landmark-based EKF-SLAM (SMITH *et al.*, 1987) (Extended Kalman Filter SLAM) to 3D without losing the rich details of an RGBD camera and yet take advantage of the EKF's probabilistic approach?

Furthermore, there are many ways to describe a 3D environment, each one with unique characteristics of memory consumption and processing time. How do different mapping representations perform in relation to memory usage when they are growing in size? How much time does it take to construct each map? This work aims to study and answer all the questions above and collect simulated and experimental data to support its conclusions.

1.2 Document structure

This work is divided into the chapters described as follows. Chapter 2 presents related works of the topics presented in this dissertation. The review details from the pioneers in the field to today's most popular methods and state of the art. Chapter 3 will present the mathematical modeling and overall explanation of the solution proposed to the following problems: (i) EKF-SLAM framework, (II) Feature extraction, (III) Data association, (IV) Feature growth, (V) Mapping representations.

Next, Chapter 4 describes the simulated environment and exposes its results in terms of mapping precision and vehicle path. Furthermore, this chapter compares different types of maps in relation to memory storage and processing time. The "Experimental results" chapter presents the developed algorithm using different scenarios from real-life environments and its results compared to the real measured environment.

Finally, the Conclusion chapter summarizes the results from previous chapters, giving insights into the main difficulties and ideas for future works.

The code for this work is available at: `https://github.com/Intelligent-Machines-Lab/3D_EKF_SLAM` or `ftp://labattmot.ele.ita.br/ele/leonardo`

1.3 Main contributions

The main contributions of this work can be summarize as follows:

- Use multiple abstraction layers to build a point cloud map on a landmark-based EKF-SLAM.
- Feature growth method for different views of observed planar or arbitrary objects.
- Construction of a parameterized hybrid map based of simplification of the point cloud representation.

2 Related works

2.1 SLAM algorithms

The pioneers in SLAM research can be traced back to the late 80's with works from Smith *et al.* (1987), Leonard and Durrant-Whyte (1991) and Castellanos *et al.* (1999). The approach in those works was to use the Extended Kalman Filter SLAM algorithm (EKF SLAM) to create a probabilistic simplified map using a set of landmarks. These landmarks are extracted from measurements of the environment (such as points of reference), which are commonly called features. These features are used in the EKF state vector along with the robot's pose. As a mobile robot explores the environment the probabilistic state vector is updated, along with its covariance matrix. This way, the EKF can create an interconnected map in which the update of one feature not only corrects the robot's pose but also the position of each other landmark.

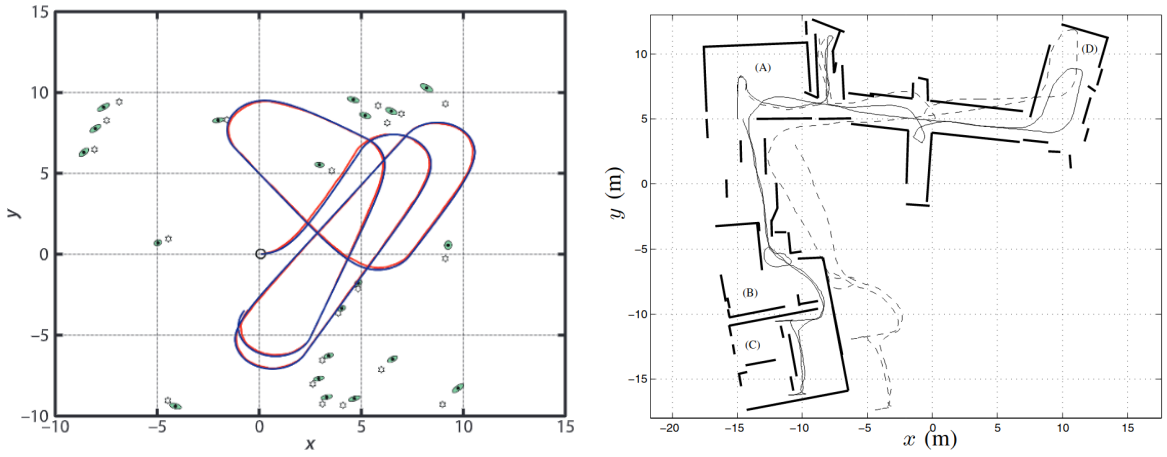


FIGURE 2.1 – The EKF-SLAM algorithm can build maps using different types features, such as the traditional point-based SLAM on the left figure (CORKE, 2013), or a line-based SLAM on the right figure (GARULLI *et al.*, 2005).

After the popularity of SLAM skyrocketed, didactic works such as Thrun *et al.* (2005), Corke (2013) and Riisgaard and Blas (2005) explain in detail the working principle of SLAM and were essential for the theoretical background for this work. Other 2D-SLAM works Garulli *et al.* (2005) Connette *et al.* (2007), Choi *et al.* (2008) and Pinto *et al.*

(2021) defined the landmarks as line segments or an unified point-and-line method. The advantage of these approaches is to increase the number of observed features which can better represent indoor scenes, simplifying walls to lines and corners to points. Pedduri *et al.* (2009) discuss the performance of such methods. Figure 2.1 exemplifies different types of maps.

2.1.1 SLAM - An overview

Let us assume the existence of a mobile robot that is exploring an indoor environment and its goal is to create a map $\mathbf{m}$. The robot's pose $\mathbf{x_{vk}}$ will vary discretely on each time step $\mathbf{t_1}, \mathbf{t_2}, ..., \mathbf{t_k}$. To estimate its movements, the robot has odometry sensors, which measures a displacement $\mathbf{u_k}$ in its pose after a movement from $\mathbf{t_k}$ to $\mathbf{t_{k+1}}$. On each time step, the robot is capable of making measurements $\mathbf{z_k}$ of the environment and correcting its pose. Therefore, on each time step, the robot predicts and corrects its pose and uses the measurements to estimate a coherent map.

The depicted processes revolve around variables of two categories: estimations or measurements. Usually, this set of variables are also called hidden states or data variables, respectively (ELIAZAR; PARR, 2003). Figure 2.2 illustrates this process.

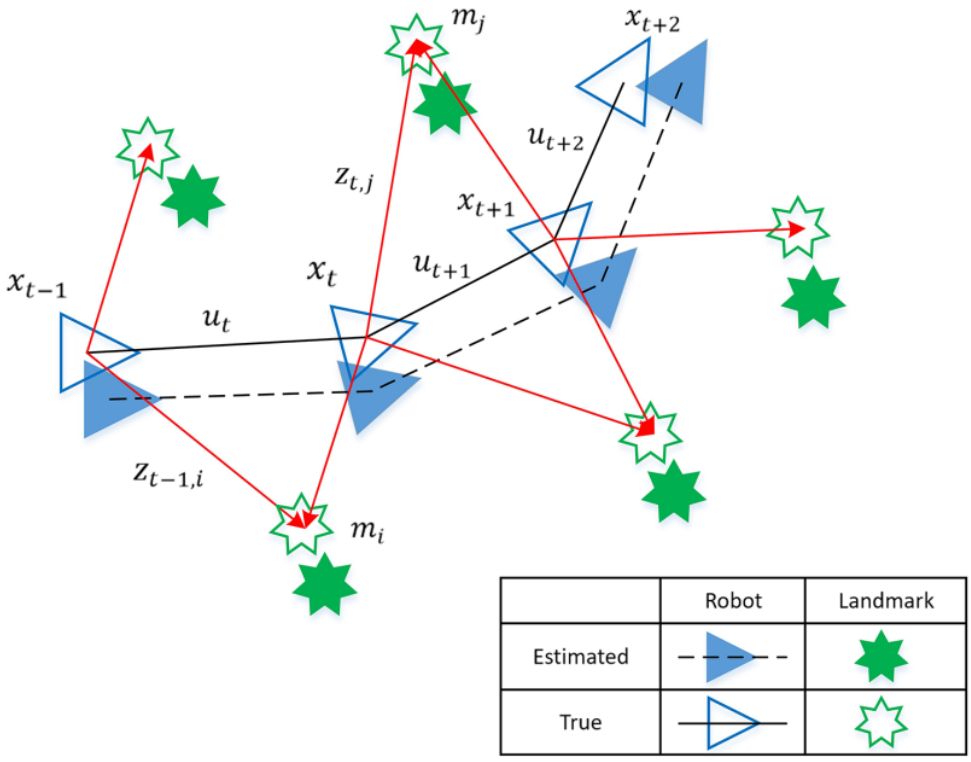


FIGURE 2.2 – The robot's pose $\mathbf{x_k}$ is estimated by receiving an action $\mathbf{u_k}$ and making a measurement of the environment $\mathbf{z_k}$, which also creates a estimation of the set of features that composes the map $\mathbf{m}$. Figure from Lei *et al.* (2020)

Important considerations (ECK, 2013):

m represents the full feature-based map estimation which contains the set of landmarks. In other words, the map $\boldsymbol{m}$ is an array of features.

$\mathbf{x_{v\,k}}$ is the the robot's pose $[x_v \quad y_v \quad \theta_v]^T$ at time k. We can refer to the trajectory of the robot as $x_{0:k} = x_0, x_1, ..., x_{k-1}, x_k$.

$\mathbf{u_k}$ is an action at step k that represents a pose displacement between x_k and x_{k+1}. Usually, this is a measurement from an odometry sensor that returns a linear and angular motion as $u_k = [\delta_x \quad \delta_\theta]^T$. We can represent a set of odometry measurements as $u_{0:k} = u_0, u_1, ..., u_{k-1}, u_k$.

$\mathbf{z_k}$ is a feature acquired by the robot's measurement of the environment observed from the robot's point of view. The set of map measurements can be represented as $z_{0:k} = z_0, z_1, ..., z_{k-1}, z_k$.

We assume the robot is moving through a static environment. Therefore, an observed feature in t_k will have the same characteristics when observed after n steps further, in t_{k+n}. Additionally, this solution assumes that x_k compresses all previous information from control, measurements, and states. No additional data from older measurements will improve future poses (ECK, 2013). This assumption is called *completeness* of space. (THRUN *et al.*, 2005)

There are two distinct ways to approach the SLAM problem:.

- **Online SLAM** is the strategy of incrementally estimating the map m and the current robot's pose x_{t+1} based on the input u and measurement z. It does not repeat the entire map estimation on each step nor previous robot's poses (THRUN *et al.*, 2005). Usually, online SLAM techniques are based on filters such as EKF-SLAM, DP-SLAM and FAST-SLAM (GRISETTI *et al.*, 2010b) The block diagram of this strategy is presented at 2.3.

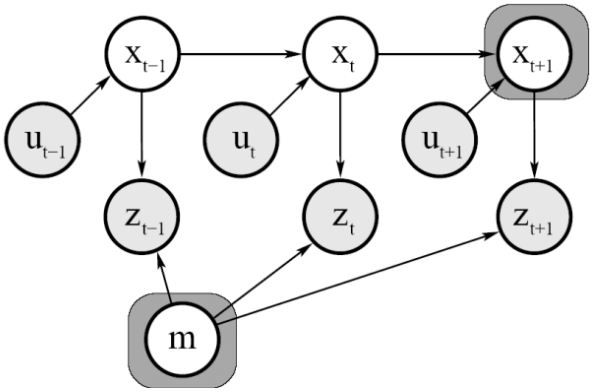


FIGURE 2.3 – In the Online SLAM strategy, the estimation (gray areas) are restricted to the current robot's pose and map (THRUN *et al.*, 2005). The arrows indicate influence between variables.

- On the other hand, **Full SLAM** intends to estimate all robot's trajectory and map. This approach prevents the integration of the poses, which are stored and optimized by methods such as graph-SLAM (GRISETTI *et al.*, 2010b). The block diagram is presented in 2.4.

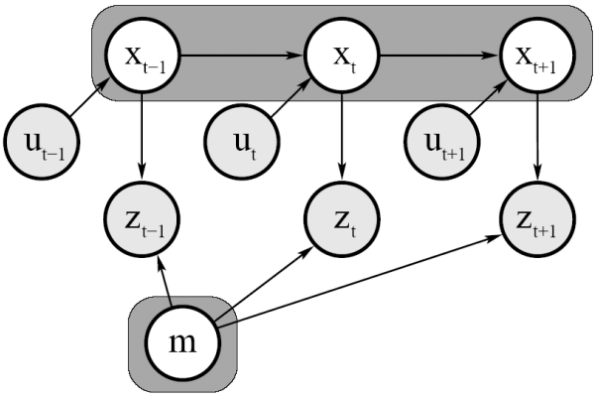


FIGURE 2.4 – In the block diagram of full SLAM strategy, the map m is estimated along the robot's trajectory (THRUN *et al.*, 2005). The arrow indicates influence from a variable to another.

2.1.2 Kalman Filter (KF)

A possible solution for the SLAM problem is the use of an estimator. In this context, the literature proposes a framework based on a Bayes-Filter (THRUN *et al.*, 2005) which divides the solution into two steps that repeat interactively: (I) Prediction and (II) Update (CORKE, 2013).

The Kalman Filter (KF) (KALMAN, 1960) can be explained based on this idea. Consider the following linear time-invariant discrete-time system:

$$x_{k+1} = F_k\, x_k + E_k\, u_k + v_k \tag{2.1}$$

$$z_k = H_k\, x_k + w_k \tag{2.2}$$

where $x_k \in \mathbb{R}^n$ is the state vector, $u_k \in \mathbb{R}^p$ is the input vector and $z_k \in \mathbb{R}^m$ describes the measured output vector. The matrix $F_k \in \mathbb{R}^{n\times n}$ describes how the previous state will dynamically evolve over time. The input coupling is represented by the matrix $E_k \in \mathbb{R}^{n\times p}$. The matrix $H_k \in \mathbb{R}^{m\times n}$ maps the influence of state vector in the measured output (CORKE, 2013).

The system is subject to a process noise $v_k \in \mathbb{R}^n$ and a measurement noise $w_k \in \mathbb{R}^m$ (CORKE, 2013). Both $v_k \approx \mathcal{N}(0, V_k)$ and $w_k \approx \mathcal{N}(0, W_k)$ are gaussian, white and zero-mean uncorrelated random variables, where V_k and W_k are respectively their covariance matrices.

In this context, the KF algorithm calculates the optimal state estimate based on

statistical uncertainty of the system's random inputs (NASCIMENTO JR., 1988). The optimal estimate of x_k is denoted by $\hat{x}_k$ and the uncertainty of this estimate is given by the covariance matrix P_k (NASCIMENTO JR., 1988).

The recursive KF algorithm can be summarized by the following equations:

$$\text{Prediction Step:} \quad \hat{x}_{k+1}^{-} = F_k\,\hat{x}_k^{+} + E_k\,u_k \tag{2.3}$$

$$P_{k+1}^{-} = F_k\,P_k^{+}\,F_k^{T} + V_k \tag{2.4}$$

$$\text{Update Step:} \quad \nu_{k+1} = z_{k+1} - H_{k+1}\,\hat{x}_{k+1}^{-} \tag{2.5}$$

$$L_{k+1} = P_{k+1}^{-}\,H_{k+1}^{T}\left(H_{k+1}\,P_{k+1}^{-}\,H_{k+1}^{T} + W_{k+1}\right)^{-1} \tag{2.6}$$

$$\hat{x}_{k+1}^{+} = \hat{x}_{k+1}^{-} + L_{k+1}\,\nu_{k+1} \tag{2.7}$$

$$P_{k+1}^{+} = P_{k+1}^{-} - L_{k+1}\,H_{k+1}\,P_{k+1}^{-} \tag{2.8}$$

Thus, given $\hat{x}_k^{+}$, P_k^{+} and the deterministic input u_k, the prediction step calculates $\hat{x}_{k+1}^{-}$ and P_{k+1}^{-}. Then, using the measured output z_{k+1}, the update step calculates $\hat{x}_{k+1}^{+}$ and P_{k+1}^{+} .

Therefore the KF algorithm produces the following sequence (NASCIMENTO JR., 1988):

1. Starting with $\hat{x}_0^{+}$, P_0^{+} and input u_0, eqs. (2.3)-(2.4) are used to calculate $\hat{x}_1^{-}$ and P_1^{-}.
2. Using $\hat{x}_1^{-}$, P_1^{-} and the measured output z_1, eqs. (2.5)-(2.8) are used to calculate $\hat{x}_1^{+}$ and P_1^{+}.
3. $\hat{x}_1^{+}$, P_1^{+} and u_1 are used to calculate $\hat{x}_2^{-}$ and P_2^{-}.
4. $\hat{x}_2^{-}$, P_2^{-} and z_2 are used to calculate $\hat{x}_2^{+}$ and P_2^{+}.
5. And so forth...

2.1.3 EKF-SLAM

2.1.3.1 Extended Kalman Filter

The Kalman Filter algorithm generates optimal state estimates when the system's dynamic is linear and time-invariant, and the process and measurement noise are gaussian, white, zero-mean and uncorrelated. Unfortunately, in several cases the system's dynamics is nonlinear and the Extended Kalman Filter (EKF) algorithm can be used to solve the state estimation problem (CORKE, 2013; NASCIMENTO JR., 1988).

Eqs. (2.9)-(2.10) describe a discrete-time nonlinear dynamical system and its output. The linearized version of these equations around a nominal trajectory (x_k^{nom}, u_k^{nom}) are given by eqs. (2.11)-(2.12) :

$$x_{k+1} = f(x_k, u_k, v_k) \tag{2.9}$$

$$z_k = h(x_k, w_k) \tag{2.10}$$

$$\Delta x_{k+1} \approx F_x\,\Delta x_k + F_u\,\Delta u_k + F_v\,v_k \tag{2.11}$$

$$\Delta z_k \approx H_x\,\Delta x_k + H_w\,w_k \tag{2.12}$$

where, by definition, $\Delta x_k = x_k - x_k^{nom}$, $\Delta z_k = z_k - z_k^{nom}$, $z_k^{nom} = h(x_k^{nom}, 0)$. The so-called Jacobian matrices are defined as:

$$F_x = \frac{\partial f(x,u,v)}{\partial x} \in \mathbb{R}^{n\times n} \quad F_u = \frac{\partial f(x,u,v)}{\partial u} \in \mathbb{R}^{n\times p} \quad F_v = \frac{\partial f(x,u,v)}{\partial v} \in \mathbb{R}^{n\times n} \tag{2.13}$$

$$H_x = \frac{\partial h(x,w)}{\partial x} \in \mathbb{R}^{m\times n} \quad H_w = \frac{\partial h(x,w)}{\partial w} \in \mathbb{R}^{m\times m} \tag{2.14}$$

Even though $f(.,.,.)$ and $h(.,.)$ are nonlinear functions, their linearized versions can be used to create an approximation of equations (2.1) and (2.2). Therefore the EKF algorithm uses a procedure similar to the KF algorithm to estimate the state vector and its covariance, but it is not an optimal state estimator anymore.

Algorithm 1 summarizes the basic EKF algorithm (adapted from Corke (2013) and Nascimento Jr. (1988)) where:

1. Matrices F_x and F_v are calculated at the point ($x = \hat{x}_k^+$, $u = u_k$, $v = 0$).
2. Matrices H_x and H_w are calculated at the point ($x = \hat{x}_{k+1}^-$, $w = 0$).

Algorithm 1 The basic EKF algorithm

Input $\hat{x}_k^+$, P_k^+, u_k, z_{k+1}, V_k, W_{k+1}
Output $\hat{x}_{k+1}^+$, P_{k+1}^+

Calculate F_x, F_v, H_x, H_w

▷ Prediction step

$\hat{x}_{k+1}^- = f(\hat{x}_k^+, u_k)$
$P_{k+1}^- = F_x\,P_k^+\,F_x^T + F_v\,V_k\,F_v^T$

▷ Update step

$\nu_{k+1} = z_{k+1} - h(\hat{x}_{k+1}^-)$
$L_{k+1} = P_{k+1}^-\,H_x^T\big(H_x\,P_{k+1}^-\,H_x^T + H_w\,W_{k+1}\,H_w^T\big)^{-1}$
$\hat{x}_{k+1}^+ = \hat{x}_{k+1}^- + L_{k+1}\,\nu_{k+1}$
$P_{k+1}^+ = P_{k+1}^- - L_{k+1}\,H_x\,P_{k+1}^-$

2.1.3.2 EKF in a landmark-based SLAM

In SLAM algorithm, the map m is also an estimation and should be part of the state vector x. Thus, consider the state vector being a union of the vehicle pose $X_v = [x_v \quad y_v \quad \theta_v]^T$ and i-th map landmark position in 3D space $m_i = [x_i \quad y_i \quad z_i]^T$. The state vector can be constructed as in eq. (2.15).

$$\hat{x} = [\underbrace{x_v,\ y_x,\ \theta_v}_{X_v},\ \underbrace{x_1,\ y_1,\ z_1}_{m_1},\ \underbrace{x_2,\ y_2,\ z_2}_{m_2},\ \ldots,\ \underbrace{x_M,\ y_M,\ z_M}_{m_M}]^T \in \mathbb{R}^{3+3M\times 1} \tag{2.15}$$

The state's covariance matrix will also be divided into three parts, as stated in eq. (2.16). $P_{vv} \in \mathbb{R}^{3\times 3}$ describes the robot's uncertainty, $P_{mm} \in \mathbb{R}^{3M\times 3M}$ is the covariance between each landmark, and $P_{vm} \in \mathbb{R}^{3\times 3M}$ is the correlation between the robot and the landmarks. (CORKE, 2013)

$$P = \begin{bmatrix} P_{vv} & P_{vm} \\ P_{vm}^T & P_{mm} \end{bmatrix} \in \mathbb{R}^{3+3M\times 3+3M} \tag{2.16}$$

As a consequence of the union between the robot's pose and landmarks' position in the state vector, the Jacobian H_x also needs to change. The observation of a landmark affects both the robot's pose and the observed landmark's previous position. The mathematical representation of this statement gives a new H_x in eq (2.17) (CORKE, 2013).

$$H_x = [H_{x_v} \ldots 0 \ldots H_m \ldots 0] \in \mathbb{R}^{3\times 3M+3} \tag{2.17}$$

where H_{x_v} maps the effect of the landmark's correction in the robot's pose and H_m in the previous observation of this same landmark.

$$H_{x_v} = \frac{\partial h(x,v)}{\partial x_v} \in \mathbb{R}^{3\times 3} \qquad H_m = \frac{\partial h(x,v)}{\partial m} \in \mathbb{R}^{3\times 3} \tag{2.18}$$

So far, we have explained the state vector with static dimensions, but that's not the case for this problem. The robot discovers new landmarks as it explores the environment. To handle this situation, the state vector should be augmented accordingly:

$$\hat{x}_{k\,(new)} = \begin{bmatrix} \hat{x}_{k\,(old)} \\ g(\hat{x}_k, z_{k+1}) \end{bmatrix} \in \mathbb{R}^{1\times n+3} \quad P_{k\,(new)} = Y_z \begin{bmatrix} P_{k\,(old)} & 0 \\ 0 & W_k \end{bmatrix} Y_z \in \mathbb{R}^{n+3\times n+3} \tag{2.19}$$

where $g(.)$ is the inverse of $h(.)$. The covariance matrix should also be augmented by using

a mathematical tool called insertion Jacobian Y_z (CORKE, 2013) defined in eq (2.20).

$$Y_z = \begin{bmatrix} I_{n\times n} & & 0_{n\times 3} \\ G_x & 0_{3\times n-3} & G_z \end{bmatrix} \in \mathbb{R}^{n+3\times n+3} \tag{2.20}$$

$$Gx = \frac{\partial g(x,z)}{\partial x} \in \mathbb{R}^{3\times 3} \qquad Gz = \frac{\partial g(x,z)}{\partial z} \in \mathbb{R}^{3\times 3} \tag{2.21}$$

In short, we can expand the Algorithm 1 to comply with these changes. Since a single scene can include more than one landmark, the update step is embraced by a for-loop, as shows Algorithm 2 (based on Corke (2013)).

Algorithm 2 The landmark-based EKF-SLAM Step

Input $\hat{x}_k^+$, P_k^+, u_k, z_{k+1}, V_k, W_{k+1}
Output $\hat{x}_{k+1}^+$, P_{k+1}^+

Calculate F_x, F_v, H_x, H_w
$\hat{x}_{k+1}^- = f(\hat{x}_k^+, u_k)$ ▷ Prediction step
$P_{k+1}^- = F_x\, P_k^+\, F_x^T + F_v\, V_k\, F_v^T$
$\hat{x}_{k+1}^+ = \hat{x}_{k+1}^-$
$P_{k+1}^+ = P_{k+1}^-$
for each observation z_{k+1} **do**
    if z_{k+1} is an observation from a previously known feature **then** ▷ Update step
        Calculate H_{x_v}, H_m, H_x and H_w
        $\nu_{k+1} = z_{k+1} - h(\hat{x}_{k+1}^+)$
        $L_{k+1} = P_{k+1}^+\, H_x^T \big(H_x\, P_{k+1}^+\, H_x^T + H_w\, W_{k+1}\, H_w^T\big)^{-1}$
        $\hat{x}_{k+1}^+ = \hat{x}_{k+1}^+ + L_{k+1}\, \nu_{k+1}$
        $P_{k+1}^+ = P_{k+1}^+ - L_{k+1}\, H_x\, P_{k+1}^+$
    else ▷ State vector augmentation
        Calculate G_x, G_z and Y_z
        $\hat{x}_{k+1}^+ = \begin{bmatrix} \hat{x}_{k+1}^+ \\ g(\hat{x}_{k+1}^+, z_{k+1}) \end{bmatrix}$
        $P_{k+1}^+ = Y_z \begin{bmatrix} P_{k+1}^+ & 0 \\ 0 & W_k \end{bmatrix} Y_z^T$
    end if
end for

2.1.3.3 EKF-SLAM - further considerations

The EKF approach was also applied for SLAM in a 3D environment. Weingarten (2006) introduced the idea of recreating an indoor scene using small planar sections, which are fused into larger plane segments (Figure 2.5). It also shows interesting insights about different plane equations and provide rich details of the algorithms used in feature extraction methods. Further studies such as Jafri *et al.* (2014) proposed a unified EKF framework implementing multiple update functions to build a partial 3D structure using corner points and lines feature. Another work, (OZAKI; KURODA, 2020) also took advan-

tage of plane segments to build an indoor scene. This last work describes the update and transformation equation for planes using a compact normal distance planar representation, which is also used in our research.

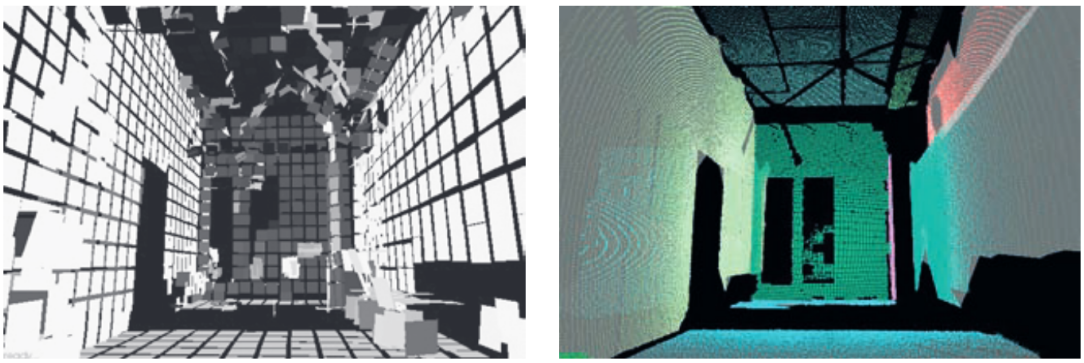


FIGURE 2.5 – The work from Weingarten (2006) uses planes to map a 3D environment using the EKF framework. The algorithm create small planar segments from a point cloud (left figure) which are merged into larger planes (right image)

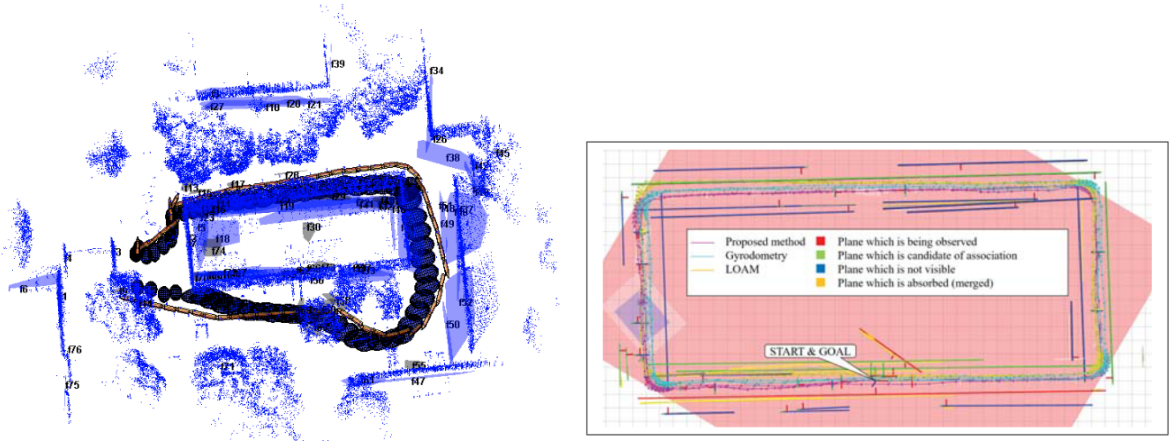


FIGURE 2.6 – The work from Ulas and Temeltas (2012) uses a UKF to estimate a planar environment (left image). The right images shows a 2D projection of a corridor build using a planar EKF framework from Ozaki and Kuroda (2020))

On the other hand, the EKF also has its downsides. One major limitation is the computational complexity, which is $\mathcal{O}(K^2)$ where K is the number of elements in the state vector. This complexity comes from the fact that even when a single landmark is measured, the whole covariance matrix must also be updated. This limits the number of landmark to just a few hundred (MONTEMERLO *et al.*, 2002), meaning that only small or medium maps can be built.

Another problem is the singular hypothesis of the robot's pose. Due to the Gaussian error distribution of the EKF correction equations, we can only have a single estimate of the robot's pose on each step. As consequence of that, the algorithm is not robust to recover from wrong data associations, which might lead to a filter divergence. Posterior works improved some of these disadvantages. Jensfelt and Kristensen (2000) proposed

global localization using EKF with multiple hypotheses. Other frameworks such as the FAST-SLAM (MONTEMERLO *et al.*, 2002) also manage to overcome this limitation.

The last major downside of the EKF is its sub-optimal behavior due to its Jacobian linearizations. Different from the Kalman Filter (KF), the prediction and measurement models from the EKF in the SLAM context are non-linear. Depending the equations, the Jacobians can have a large quantity of terms and be difficult to calculate and validate. Furthermore, its linearization implies that the estimate is only an approximation of the optimal state. Moreover, the use of Unscented Kalman Filter (UKF) (JULIER; UHLMANN, 1997) is an alternative method to avoid calculation of Jacobian and, consequentially, the linearization errors of the EKF. In the context of 3D-SLAM Ulas and Temeltas (2012) used UKF to create a planar SLAM algorithm and compared its efficiency to the EKF (Figure 2.6).

2.1.4 Data association

A side problem when using a landmark-based SLAM is the data association (THRUN *et al.*, 2005). Fundamentally, this problem consists in creating an association between a landmark that was seen in the previous step and the observed landmark in the current step. A possible way to attack this problem is to calculate the euclidean distance between the existing and the observed feature and verify the proximity between them. However, this approach does not account for the uncertainty of the map, which complicates the association.

In this context, the Mahalanobis Distance (MD) (MAHALANOBIS, 1936) is one of the most popular data association equations in the literature. It uses the correlation of a dataset to create a scale-invariant distance between features. It is also practical to use along with the EKF as it uses the same covariance matrix. Unfortunately, the MD only associates features individually. In other words, it is not able to associate a group of observed features to a group of existing features, which could lead to significantly less ambiguous associations (LI; OLSON, 2012).

To consider association in groups of features, the Joint Compatibility Branch and Bound (JCBB) (SHEN *et al.*, 2016) is an interesting probabilistic solution. This method uses a tree structure to search for the largest set of associations that are bounded to an error value. Other alternatives for group associations are the Maximum Common Subgraph (QUER *et al.*, 2020) or the Incremental Posterior Joint Compatibility (LI; OLSON, 2012) which is considered a faster version of the JCBB.

2.1.4.1 Loop closing

Loop closing is the task of deciding whether or not the vehicle has returned to a previously visited area after moving an arbitrary distance from the map (MASÓ; JOSEP, 2011). Essentially, the problem of loop closing is an extreme case of data association, challenged by high ambiguity and environment symmetries. In the context of EKF-SLAM, when a loop closure happens, it reduces the uncertainty of robot and landmark estimates and this can be exploited when exploring an environment. However, wrong loop closure can lead to filter divergence (STACHNISS, 2011). The loop closure can be detected by a simple method such as the MD, which updates the EKF state and covariance matrix. Other works such as Masó and Josep (2011) propose loop closing techniques for joining sub-maps.

2.1.5 Other Related SLAM techniques

2.1.5.1 FAST-SLAM

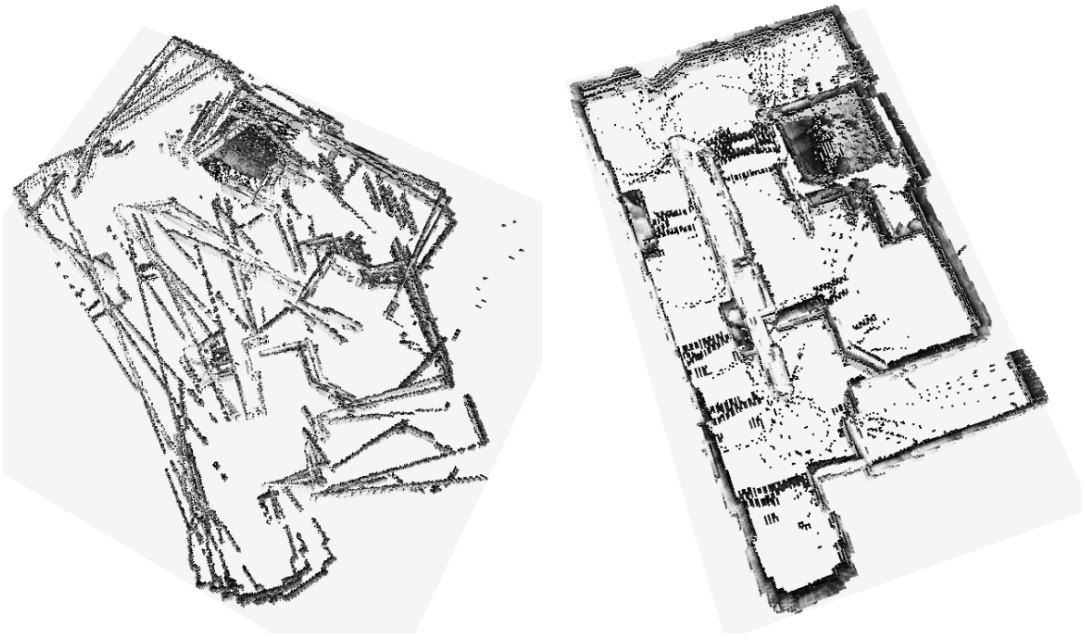


FIGURE 2.7 – Work from Eck (2013) uses the Rao-Blackwellized Particle Filter (RBPF) to estimate a 3D map. The left image is the odometry-based map and the right image is the correction after the SLAM algorithm.

The FAST-SLAM comes as solution for many EKF-SLAM limitations. To address efficiency, particle filters (GORDON, 1993) is a well-adopted technique for global optimization. The FAST-SLAM (MONTEMERLO *et al.*, 2002) applies the Rao-Blackwellized Particle Filter (RBPF) (CASELLA; ROBERT, 1996) and uses multiple estimation of the robot pose, which generates more robust estimations along the time. Furthermore, this

technique scales linearly with the number of features, having complexity of $\mathcal{O}(M\,K)$ (MONTEMERLO *et al.*, 2002) where M is the number of particles and K is the number of features. On the other hand, its performance deeply depends on the choice value for M, and may also be difficult to manage the particles (KWAK *et al.*, 2007). Works in 3D environment were also successful at Eck (2013) (Figure 2.7), Jo *et al.* (2014) and Gharatappeh *et al.* (2015).

2.1.5.2 Invariant features

Working with RGB image enables using invariant feature detection and description techniques such as ORB (RUBLEE *et al.*, 2011), BRISK (LEUTENEGGER *et al.*, 2011), BRIEF (CALONDER *et al.*, 2010), SURF (BAY *et al.*, 2006) and FAST (ROSTEN; DRUMMOND, 2006). This family of algorithms can extract reliable features (usually points) from an RGB image. This extraction follows pre-defined rules and goals (i.e. detecting corners) based on a series of filtering techniques. The extraction of these features from a series of images allows to align two consecutive frames and estimate a spatial transformation. These methods are also known as " visual odometry " because they are measurements of displacement that only use one camera. However, they are subject to an accumulation of errors because they are not a direct pose measurement.

2.1.5.3 Point cloud registration

When a depth sensor is available, it is possible to use raw point cloud data as a driver for displacement measurement. This approach is called point cloud registration (WEN-PENG *et al.*, 2018) and consists in aligning two point clouds and calculating the 3D transformation. The iterative closest point (ICP) (BESL; MCKAY, 1992) is the most classic method addressing point cloud registration. It searches for the nearest neighbor point pairs that meet certain constraints and uses an optimization process to iteratively calculate a rotation and translation matrix to minimize the error function (XU *et al.*, 2021). This algorithm is time-consuming and only locally optimal, which needs a good initial guest to reach the global optimal, usually used as a fine registration process (BACK *et al.*, 2018). For the coarse registration, Random Sample Consensus (RANSAC) (HAN *et al.*, 2015) (LI *et al.*, 2021) and Fast Global Registration (ZHOU *et al.*, 2016) are a well-adopted strategy in the literature.

Both invariant features and point cloud registration only reconstruct a local map. To create a global map and perform loop closures, these techniques can complementary use a graph-based optimizer, which is called Graph-based SLAM (LU; MILIOS, 1997).

2.1.6 Graph SLAM

In graph-based SLAM, the robot's pose is the node of a graph, and the edges define the movement and its uncertainty (GRISETTI *et al.*, 2010b). Whenever the algorithm finds a relational match between nodes, it executes a global optimizer such as G2O, (KüMMERLE *et al.*, 2011), HOG-Man(GRISETTI *et al.*, 2010a) and GTSAM (DELLAERT; KAESS, 2006), which updates all robot poses and map estimates (full SLAM). Hence, Graph SLAM is one of the most researched procedures in 3D SLAM, as it manages to reconstruct a realistic and consistent map of the environment. Many works manage to successfully combine these techniques such as Choi *et al.* (2015) and Mahmoud *et al.* (2014).

2.2 Point cloud manipulation

There are many tools available in the literature to manipulate point clouds. This includes methods for segmentation, clusterization, parameterization, filtering and fitting. This sections summarize some of the main techniques used in this research.

2.2.1 Geometric fitting

A famous problem related to point cloud datasets is fitting geometric primitives, such as lines, planes and circles. A straight-forward solution is to use least squares to find pre-determined shapes (MARSHALL *et al.*, 2001), which gives decent results in a low-noise environment. However, for high noise applications, the RANSAC method (FISCHLER; BOLLES, 1981) is known as a robust alternative.

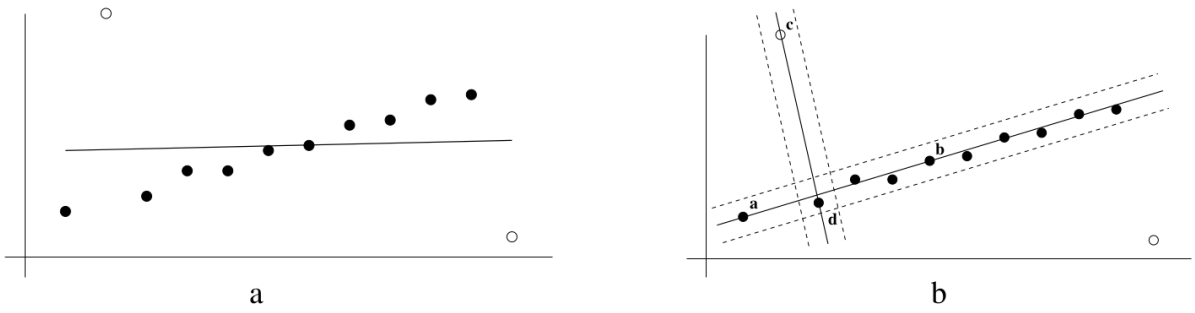


FIGURE 2.8 – Hartley and Zisserman (2003) exemplifies RANSAC in this simple case of fitting a line equation. Figure (a) shows the result of using least-squares (orthogonal regression), which is extremely affected by noise. Figure (b) shows the winning candidate as lines $\langle a, b\rangle$ and a looser candidate as line $\langle c, d\rangle$. The dotted line indicates the threshold distance, and the points between them are considered inliers.

In this technique, a parametric model is solved by randomly selecting $\boldsymbol{N_{pt}}$ points from the cloud. Then, all $\boldsymbol{n_v}$ points from the point cloud are classified based on threshold values

into **inliers** or **outliers**. The inliers are those which are inside the predefined threshold, the others are considered outliers. After repeating this process $\boldsymbol{n_i}$ times the model with more inliers is considered the best model. Figure (2.8) shows an example of this algorithm.

The RANSAC complexity is $\mathcal{O}(n_i \cdot n_v)$ (WEINGARTEN, 2006) and a general rule for finding the number of iterations can be calculated as eq. (2.22) (HARTLEY; ZISSERMAN, 2003):

$$n_i = \frac{log(1 - p(sucess))}{log(1 - p(inliner)^{N_{pt}})} \tag{2.22}$$

where $p(success)$ is the probability of the best model is found in a iteration, and $p(inlier)$ is the percentage of inliers in the point cloud (URBANčIč *et al.*, 2014). Unfortunately, both parameters are unknown in a scene, and must be estimated empirically.

Other works extended RANSAC capabilities in the context of primitive shape extraction. Liu and Wu (2014) implemented histogram analysis and boundaries trimming techniques to decompose a point cloud into a set for primitive shapes, as illustrated in Figure 2.9. An efficient RANSAC version is used in Schnabel *et al.* (2007) which performed the segmentation in high noise point clouds. Other fitting techniques are the Hough transform (HOUGH, 1962) and PGP2X (TOONY *et al.*, 2015).

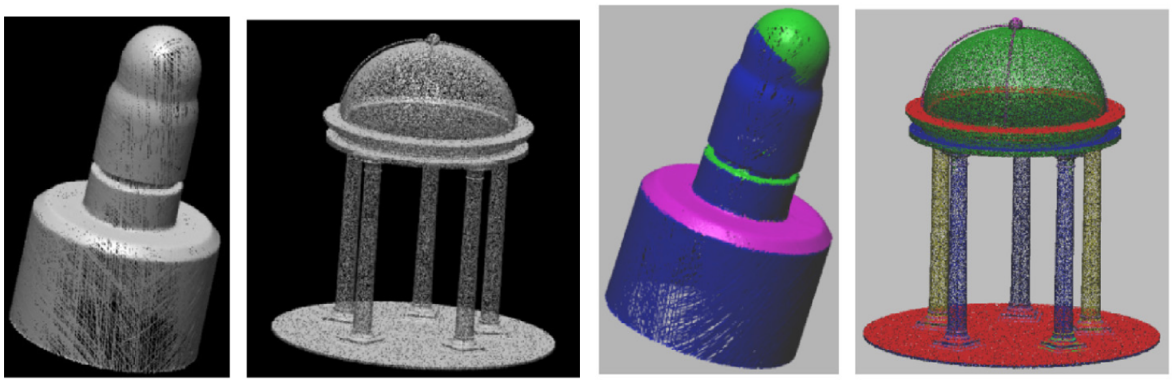


FIGURE 2.9 – An adaptive RANSAC method from Liu and Wu (2014) decomposes and segment the points into geometric primitives such as planes, cylinder and spheres. The left image is the original point cloud and the right image is the points segmented into their primitive shape.

2.2.2 Clusterization

A widely explored problem in the literature the classification a specific subset of points into different groups (or clusters). Famous strategies worth citing are K-means (LLOYD, 1982), Spectral clustering (NG *et al.*, 2001), Density-based spatial clustering of applications with noise (DBSCAN) (ESTER *et al.*, 1996), Mean-shifiting (FUKUNAGA; HOSTETLER, 1975) and Ward algorithm (WARD, 1963).

In this context, the DBSCAN has many advantages compared to other methods. First the algorithm can find an arbitrary number of clusters in the dataset, which is not possible

in the K-means. Second, different from K-means, Mean-shiffting and Ward, this method can create clusters of arbitrary shapes, due to its neighborhood detection procedure (ESTER *et al.*, 1996), as we can see in Figure 2.10.

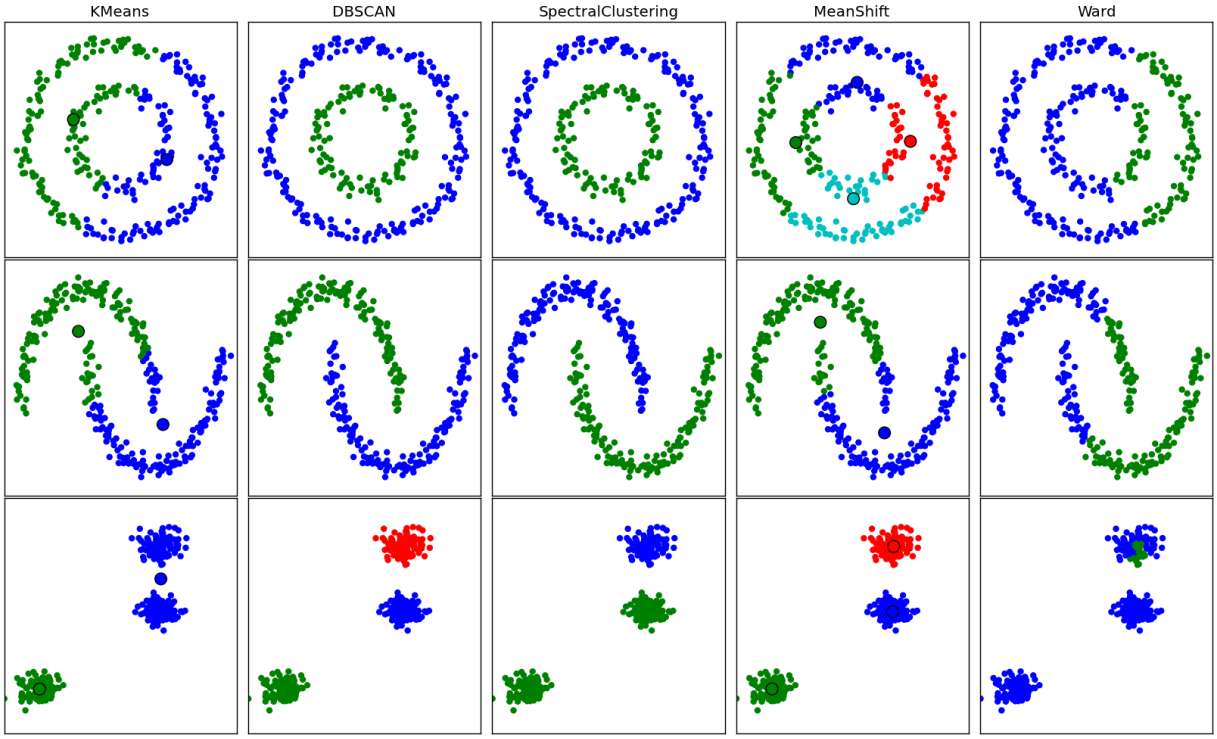


FIGURE 2.10 – Different clustering algorithms (PEDREGOSA *et al.*, 2011). The DB-SCAN algorithm has better performance in arbitrary shapes of point cloud clouds and does not need to know *a priori* the number of clusters.

DBSCAN works by using a simple minimum density level estimation based on a threshold for the number of neighbors, m_{pts} within the radius η from a selected initial point. In case this assumption is true, the point is considered a *core point*. For efficiency reasons the algorithm consider all neighbors within the radius of the previous *core point*, also as *core points*. Points that are not *core points* but are density connected are *border points*. Points that are not density reachable from any core point are considered *noise points* (SCHUBERT *et al.*, 2017). Figure 2.11 illustrates each type of point.

The time complexity of the DBSCAN technique is not trivial to calculate and depends on the point spatial distribution which is discussed in Schubert *et al.* (2017). Another interesting curiosity it that due to its importance in today's research, the DBSCAN won the 2014 SIGKDD Test of Time Award (MACHINERY, 2013).

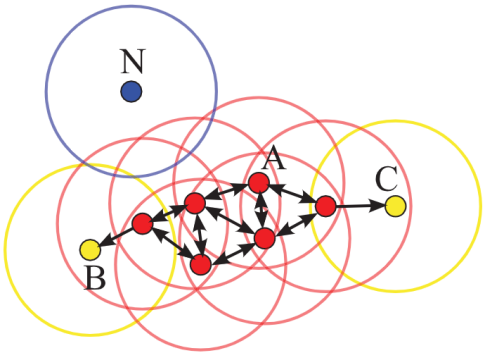


FIGURE 2.11 – Illustration from Schubert *et al.* (2017) shows the different categories of points in a cluster. Core point (A), border points (B and C) and noise point (N).

2.3 Map representations

The SLAM literature presents many ways to build a maps in a SLAM system. Generally, choosing a map representation seeks for (1) be aware of geometry of a scene, which results in better accuracy and geometry reasoning and (2) be efficient in terms of computation time and memory (MUGLIKAR *et al.*, 2020).

The majority of works in 3D SLAM uses point cloud to represent an environment (CHOI *et al.*, 2015). Point clouds can create realistic representations, but are inefficient in terms of memory and processing time since they store raw data from sensors. It also does not give sense of geometry of a scene without additional processing techniques (MUGLIKAR *et al.*, 2020).

Another type of map representation uses geometric primitives such as planes (KAESS, 2015), lines and points (NARDI *et al.*, 2019). This system is efficient due its parametric nature, and provides interesting details of the scene geometry. The limitation of this method is making assumptions of the environment (MUGLIKAR *et al.*, 2020) which decreases its accuracy in non-structured environment.

Maps can also be represented as voxel-grids. This representation divides the space in tridimensional grids where the occupied spaces are voxels. Instead of using a traditional dense matrix of elements, many of today's voxel-grid implementations uses hash tables to store its occupied voxels (MUGLIKAR *et al.*, 2020). Other works, such as Zhou *et al.* (2018) uses containers with allocators for dynamically handling storage needs, such as unordered map. This last data structure has an average constant-time complexity (C++, 2021) for operations such as search, insertion and removal of elements which speedup creating and managing maps.

An alternative to the voxel-grid is to use sparse voxels. One of these techniques widely used is the octree (LABORATORY; MEAGHER, 1980). The creation of an octree starts by defining a *root node* with the size of the map to be represented. This node is subdivided

into eight other nodes of equal size, representing each of the octants of this larger cell. If there are points within the subdivided cells, a new subdivision is made recursively in the octant where the point is located. This is called an *intermediate node*. Finally, when a tree stops its subdivision, the node that contains the point is called a *leaf node*. Figure 2.12 shows a graphical representation of this tree structure.

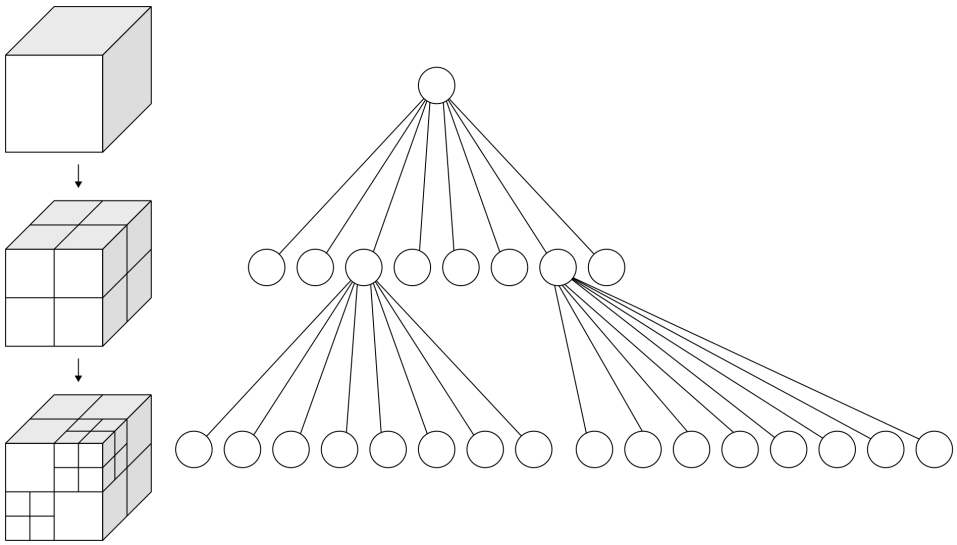


FIGURE 2.12 – Octree structure (Image available at `https://en.wikipedia.org/wiki/Octree`)

3 Proposed solution

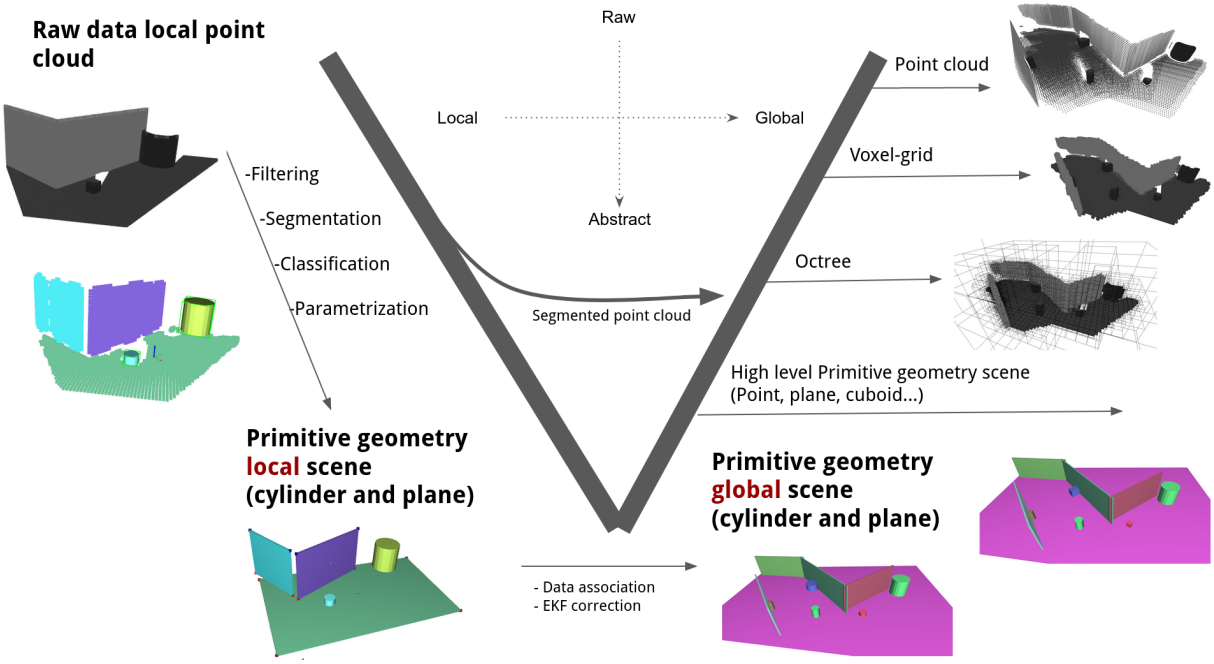


FIGURE 3.1 – Overview of the implemented algorithm.

In previous chapters, we presented the overall algorithms used in this work. In this chapter we describe an in-depth view of our implemented algorithm which includes: (I) Coordinate frame definition, (II) EKF framework adapted to the context of this work (III) Feature extraction, (IV) Data association, (V) Feature growth, (VI) Map representations, and (VII) implementation details.

An overview the algorithm is illustrated in Figure 3.1 as a V-shaped process. The vertical axis represents map density (which is related to its level of abstraction) and the horizontal axis is the processing order. Starting as a point cloud and going down to a simplified primitive and parameterized local scene, the features are associated and merged to a global map.

By doing that, the robot's pose is also corrected and a global high level map can be generated using the parameterized feature. If the application chooses to carry the point cloud, it can create other types of maps such as hybrid representation or voxel-based maps

as output. All this process will be explained in details in this chapter

3.1 World modeling

3.1.1 Coordinate frames

In this work, the motion dynamics happens in two main coordinate frames. The definition of these frames helps to describe linear and angular movements of the robot and the point-cloud transformation throughout the work. Figure 3.2 shows the adopted configuration.

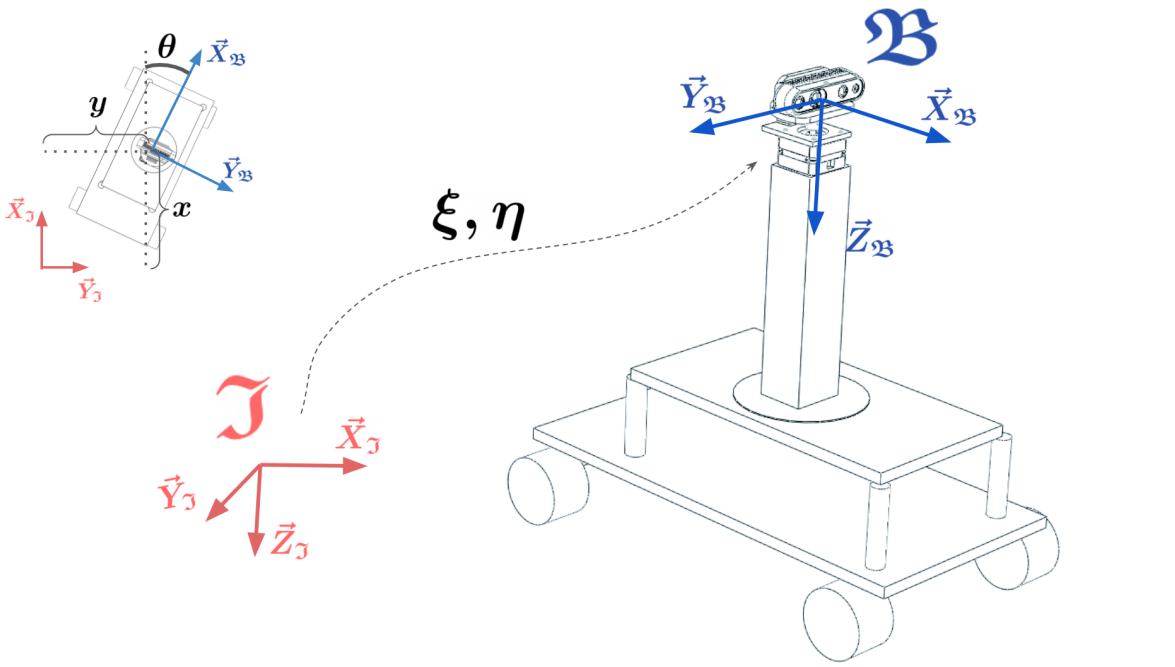


FIGURE 3.2 – Coordinate frames and spacial displacements.

The first coordinate frame is the **inertial frame** $\mathfrak{I} = \{\vec{X}_{\mathfrak{I}}, \vec{Y}_{\mathfrak{I}}, \vec{Z}_{\mathfrak{I}}\}$, which is fixed in the ground and is base for global map. The **body frame** $\mathfrak{B} = \{\vec{X}_{\mathfrak{B}}, \vec{Y}_{\mathfrak{B}}, \vec{Z}_{\mathfrak{B}}\}$ is fixed on the RGBD camera and rotates rigidly with it.

3.1.2 6-DoF Kinematics

The robot's pose (position and orientation) regarding to the inertial frame $\mathfrak{I}$ is detailed in eqs. (3.1) and (3.2):

$$\xi = [x \quad y \quad z]^T \in \mathbb{R}^3 \tag{3.1}$$

$$\eta = [\alpha \quad \beta \quad \theta]^T \in \mathbb{R}^3 \tag{3.2}$$

A rotation matrix handles the transformation between different types of frames. This matrix consists of a sequence of independent and pre-defined rotations around their axes. For this work, we use the ZYX rotation pattern, which consists of the rotation sequence θ-β-α, shows eq. (3.3).

$$R_{zyx}\{\eta\} = \begin{bmatrix} c(\beta)c(\theta) & s(\alpha)s(\beta)c(\theta) - c(\alpha)s(\theta) & c(\alpha)s(\beta)c(\theta) + s(\alpha)s(\theta) \\ c(\beta)s(\theta) & s(\alpha)s(\beta)s(\theta) + c(\alpha)c(\theta) & c(\alpha)s(\beta)s(\theta) - s(\alpha)c(\theta) \\ -s(\beta) & s(\alpha)c(\beta) & c(\alpha)c(\beta) \end{bmatrix} \tag{3.3}$$

where $c(x) = cos(x)$ and $s(x) = sin(x)$. Therefore, a point $P \in \mathbb{R}^3$ defined in $\mathfrak{B}$ is represented in $\mathfrak{I}$ through the rotation $R_{zyx}\{\eta\} \in \mathbb{R}^{3x3}$ with:

$$P_{\mathfrak{I}} = R_{zyx}\{\eta\}P_{\mathfrak{B}} \tag{3.4}$$

Hence, the inverse rotation $\mathfrak{I} \longrightarrow \mathfrak{B}$ is given by $R^{-1}_{zyx}\{\eta\}$ since $R_{zyx}\{\eta\}$ is orthogonal, the transformation from the inertial frame to the body is:

$$P_{\mathfrak{B}} = R^{T}_{zyx}\{\eta\}P_{\mathfrak{I}} \tag{3.5}$$

3.1.3 3-DoF Kinematics

Although this work creates maps in a 3D environment, the robot only makes planar movements on the floor. Consequently, we can safely assume that $z = \beta = \alpha = 0$ [1] which leads to a 3-DoF pose definition. This simplification is helpful for this work since now we can describe the vehicle pose vector as a joint position-orientation representation described in eq. (3.6).

$$X_v = [x_v \quad y_v \quad \theta_v]^T \in \mathbb{R}^3 \tag{3.6}$$

With this simplification, eq. (3.3) reduces to the rotation matrix in eq. (3.7).

$$R_{yx}\{\theta\} = \begin{bmatrix} c(\theta) & -s(\theta) \\ s(\theta) & c(\theta) \end{bmatrix} \tag{3.7}$$

3.2 Applied EKF solution

In this section, we describe in detail the particular solution for the EKF-SLAM framework used in this work. This includes the definition of the update and measurement function, feature representations, and state space definition.

[1] Unless stated otherwise we will also assume $\xi = [x \quad y \quad 0]^T$ and $\eta = [0 \quad 0 \quad \theta]^T$. The use of vectors ξ and η in $\mathbb{R}^3$ are necessary to make coherent dimensional matrix operations in further sections.

3.2.1 Prediction step

The prediction step is the first part of the Kalman filter. It defines a mathematical model $f(.)$ which transforms the last state and input in a prediction of the current state.

For this problem we assume the actor of the solution is a mobile robot that moves itself in the environment using only planar movements. This imply that no movements are made in $\vec{Z}_{\mathfrak{B}}$, and $z = 0$ for all steps. Furthermore, the robot only makes unidirectional displacement in $\vec{X}_{\mathfrak{B}}$ and rotations around $\vec{Z}_{\mathfrak{B}}$. Hence, it is safe to assume no movements in $\vec{Y}_{\mathfrak{B}}$ and no rotations in $\vec{X}_{\mathfrak{B}}$ and $\vec{Y}_{\mathfrak{B}}$ which lead to $\alpha = \beta = 0$. Finally, for this to be true, the robot's frame orientation $\vec{Z}_{\mathfrak{B}}$ should be initially aligned to the inertial frame $\vec{Z}_{\mathfrak{I}}$ as $\vec{Z}_{\mathfrak{B}0} = \vec{Z}_{\mathfrak{I}0}$.

Based on assumptions above the input vector $\boldsymbol{u}$ can be defined as the odometry measurement of displacements, where $\boldsymbol{\delta_x}$ is the displacement in $\vec{X}_{\mathfrak{B}}$ and $\boldsymbol{\delta_\theta}$ is the angular displacement in $\vec{Z}_{\mathfrak{B}}$. Thus, the input vector at step k is defined in eq. (3.8).

$$u_k = [\delta_x \quad \delta_\theta]^T \in \mathbb{R}^{2\times 1} \tag{3.8}$$

The new state can be calculated by adding the displacement to the old state. Notice, however that the input vector is defined in $\mathfrak{B}$ but the state vector in $\mathfrak{I}$. Therefore, $f(.)$ function requires a coordinate frame conversion. The process noise $v_k = [v_x, v_\theta] \in \mathbb{R}^{2\times 1} \approx \mathcal{N}(0, V)$ is also added to the displacement due to odometry noise.

$$X_{v\,k+1} = f(X_{v\,k}, u_k, v_k) = \begin{bmatrix} x_k + (\cos\theta_k)(\delta_x + v_x) \\ y_k + (\sin\theta_k)(\delta_x + v_x) \\ \theta_k + (\delta_\theta + v_\theta) \end{bmatrix} \tag{3.9}$$

Finally, the jacobians defined in chapter 2, from eq. (2.13), are computed in eq. (3.10).

$$F_x = \begin{bmatrix} 1 & 0 & (-\sin\theta_k)\delta_x \\ 0 & 1 & (\cos\theta_k)\delta_x \\ 0 & 0 & 1 \end{bmatrix} \qquad F_v = \begin{bmatrix} \cos\theta_k & 0 \\ \sin\theta_k & 0 \\ 0 & 1 \end{bmatrix} \tag{3.10}$$

3.2.2 Feature representation

The update step is the second part of the Kalman Filter. It takes measurements from the environment and corrects the robot's predicted pose. This section discusses the feature representation for landmarks, and the functions responsible for its corrections.

In this work, we map a semi-structured indoor environment, where its surrounding contains walls, floor, and arbitrary objects. The solution adopted is an expansion to

3D from the point and line dual representation (PINTO *et al.*, 2021) and (PEDDURI *et al.*, 2009). Hence, our work proposes the use of *point and plane* representation for the EKF's landmarks.

Though this representation is used for the internal calculations, the actual world map is instead based on *plane segment* and a *cylinder* which gives the plane and point features respectively a boundary coherence for 3D objects. In other words, the EKF only carries the infinite plane representation of the plane segment and the centroid from the cylinder-equivalent of an object. Other information required to further describe the feature is carried outside of the EKF.

3.2.2.1 Plane features

TABLE 3.1 – Different plane equations from literature. 1- Hessian Normal Form, 2- Minimal with no singularity, 3- Single unit vector form (WEINGARTEN, 2006)

#	Plane Equation	Parameters
1	$Ax + By + Cz + D = 0$	$\{A, B, C, D\}$
2	$x\,cos\theta\,cos\psi + ycos\theta sin\psi + zsin\theta - \rho = 0$	$\{\theta, \psi, \rho\}$
3	$\nu = \frac{1}{\sqrt{1+d^2}} \begin{bmatrix} n \\ d \end{bmatrix}$	$\nu = [\nu_1, \nu_2, \nu_3, \nu_4]^T$

The algorithm implemented for this work uses planar segments to describe flat surfaces, and the plane's infinite equation is used in EKF to construct the map. There are many ways to describe a plane, each one with its own advantages and drawbacks when applied to EKF. The most traditional representation is called Hessian Normal Form, which can be generalized in eq. (3.11), where the set of scalars $\{a, b, c, d'\}$ defines the plane model and $\{x, y, z\}$ are the coordinates of a point in 3D space.

$$ax + by + cz + d' = 0 \tag{3.11}$$

Equation (3.11) can be rewritten as (3.12) compressing all module information to d. In this new representation the plane's normal vector is explicitly defined $\vec{\boldsymbol{n}} = [\boldsymbol{n_a}, \boldsymbol{n_b}, \boldsymbol{n_c}]^{\boldsymbol{T}} \in \mathbb{R}^{3\times1}$ and $\boldsymbol{d}$ is closest distance from the plane to the origin.

$$n_a x + n_b y + n_c z + d = 0 \qquad || \qquad |n_a, n_b, n_c| = 1 \tag{3.12}$$

This representation is not minimal since only three parameters are necessary to define a plane. Assuming that $\vec{n}$ is always a unit vector, the scalar distance d can be incorporated

to $\vec{n}$ leading to a new plane model in eq. (3.13).

$$N = d.\vec{n} = [N_x \quad N_y \quad N_z]^T \in \mathbb{R}^{3\times 1} \tag{3.13}$$

This plane representation has a intuitive explanation: it is the coordinate of the 3D point where the plane is tangent to a sphere of radius d and its center is the origin. The advantage of this representation is to use only three scalars, which minimizes the size of filter's state vector. Furthermore, the plane's orientation is obtained by $\vec{\boldsymbol{n}} = \frac{\vec{N}}{|\vec{N}|}$ and the distance from origin, described by the module $\boldsymbol{d} = |\vec{\boldsymbol{N}}|$. On the other hand, different from the Hessian Normal Form, the eq. (3.13) has a singularity and do not define planes at the origin.

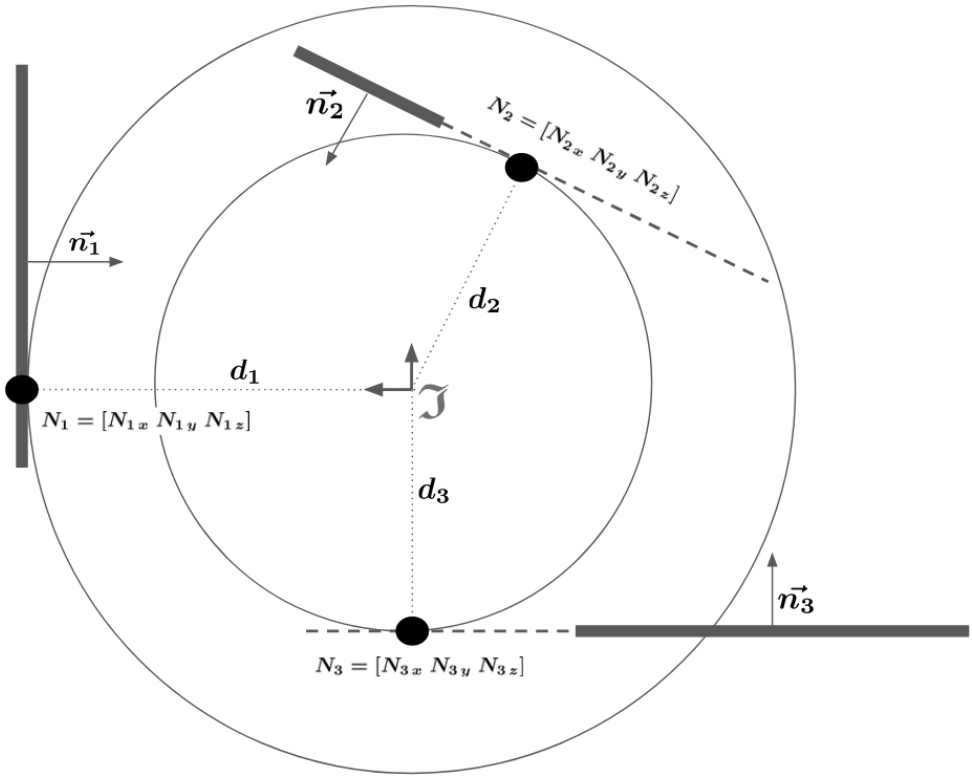


FIGURE 3.3 – The plane feature (projected as 2D as the thick line) representation is the coordinate which the infinite plane intersects a sphere of radius d from the origin.

Table 3.1 shows a summary of the main equations used in the literature. Aside from number 1 and 2, which was already mentioned before, there are two more relevant representations. Model 3 is also a minimal plane model and does not have singularities. However, its corresponding nonlinear regression problem is difficult to handle analytically. Additionally, Model 4 manages to use a single unit vector for plane representation but it is not minimal. (WEINGARTEN, 2006)

Thus, Model 2 was the choice for plane representation in this work because of its efficiency and simplicity. Furthermore, it can easily be understood and operated. Preliminary empirical tests showed that the singularity of the model didn't presented itself as a

practical problem, and can be addressed computationally if necessary.

Finally, the set of plane features defined in the global map $\mathfrak{I}$ can be expressed as

$$\boldsymbol{m_n} = [N_1^T \quad N_2^T \quad ... \quad N_n^T] \subset \hat{\boldsymbol{x}} \quad || \quad N_i = d.[n_a \quad n_b \quad n_c]^T \in \mathbb{R}^3 \tag{3.14}$$

Now, we have to define the EKF function $\boldsymbol{h_p(\hat{x})}$, which transforms the feature from $\mathfrak{I}$ to $\mathfrak{B}$. Figure 3.4 illustrates this operation. The i-th observed plane feature $\boldsymbol{z_p}$ at time step k, defined in $\mathfrak{B}$, can be obtained by the following algebraic operation (OZAKI; KURODA, 2020):

$$\boldsymbol{z_{p\,i}} = \boldsymbol{h_p(\hat{x}_k)} = R_{zyx}^T\{\eta_k\}(N_{i,k} + \frac{\xi_k \cdot N_{i,k}}{||N_{i,k}||^2} N_{i,k}) + w_k \tag{3.15}$$

where $\boldsymbol{w} = [w_x \quad w_y \quad w_z]^T$ is the measurement noise on each axis resulting from the camera depth noise plus the feature extraction process, η is robot's orientation in the 3D space and ξ is the robot's position in the 3D space. The Jacobians H_{x_v} and H_m from eq. (2.14) and eq. (2.18) are computed using the symbolic toolbox from Matlab®.

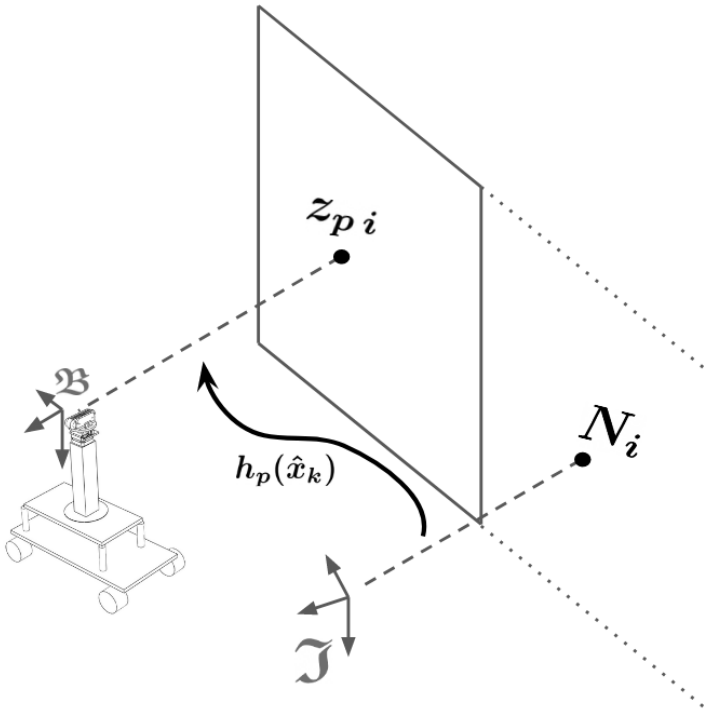


FIGURE 3.4 – The $h_p(\hat{x}_k)$ function transforms the plane from the inertial coordinate frame to the robot's coordinate frame.

3.2.2.2 Point Feature

This work uses a cylindrical representation to map general non-planar bodies. This type of feature is expressed in the EKF as a point in the cylinder's centroid. The set of

point features defined in the global map $\mathfrak{I}$ can be expressed as:

$$\boldsymbol{m_c} = [C_1^T \quad C_2^T \quad ... \quad C_n^T] \subset \hat{\boldsymbol{x}} \quad || \quad C = [c_x, c_y, c_z]^T \in \mathbb{R}^3 \tag{3.16}$$

The i-th point feature C_i is measured by the robot at coordinate z_{ci} at time step k. This conversion is computed by $\boldsymbol{h_c(\hat{x}_k)}$ function which maps the centroid from the inertial frame $\mathfrak{I}$ to the robot's frame $\mathfrak{B}$ in eq. (3.17).

$$\boldsymbol{z_{ci} = h_c(\hat{x}_k)} = R_{zyx}^T\{\eta_k\}(C_{i,k} - \xi_k) + w_k \tag{3.17}$$

where $\boldsymbol{w} = [w_x \quad w_y \quad w_z]^T$ is also the measurement noise on each axis resulting from the camera depth noise plus the feature extraction process. Function $h(.)$ can be expanded in eq. (3.18) which gives jacobians in eq. (3.19) and (3.20).

$$h_c(\hat{x}_k) = \begin{bmatrix} \cos\theta_k(c_{x\,i,k} - x_k) + \sin\theta_k(c_{y\,i,k} - y_k) \\ -\sin\theta_k(c_{x\,i,k} - x_k) + \cos\theta_k(c_{y\,i,k} - y_k) \\ c_z \end{bmatrix} + \begin{bmatrix} w_{x\,k} \\ w_{y\,k} \\ w_{z\,k} \end{bmatrix} \tag{3.18}$$

$$H_{c\,m} = \begin{bmatrix} cos\theta_k & sin\theta_k & 0 \\ -sin\theta_k & cos\theta_k & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.19}$$

$$H_{c\,x_v} = \begin{bmatrix} -cos\theta_k & -sin\theta_k & sin\theta_k(x_k - c_{x\,i,k}) - cos\theta_k(y_k - c_{y\,i,k}) \\ sin\theta_k & -cos\theta_k & cos\theta_k(x_k - c_{x\,i,k}) + cos\theta_k(y_k - c_{y\,i,k}) \\ 0 & 0 & 0 \end{bmatrix} \tag{3.20}$$

3.2.3 State vector definition

The EKF state vector from eq. (2.15) can be built joining the robot's pose x_v defined by eq. (3.6), the set of j point features m_c defined by eq. (3.16) and the set of l plane features m_p defined by eq. (3.14). As stated before, $C_i \in \mathbb{R}^3$ is the centroid of the i-th point feature in $\mathfrak{I}$ and $N_i \in \mathbb{R}^3$ is representation of the i-th plane feature defined in $\mathfrak{I}$.

$$\begin{aligned} \hat{x} &= [x_v^T \quad m_c \quad m_n]^T \\ &= [x_v^T \quad C_1^T \quad ... \quad C_j^T \quad N_1^T \quad ... \quad N_l^T]^T \in \mathbb{R}^{3+3(j+l)\times 1} \end{aligned} \tag{3.21}$$

The covariance matrix from eq. (2.16) can be rewritten as:

$$P = \begin{bmatrix} P_{vv} & P_{vc} & P_{vn} \\ P_{vc}^T & P_{cc} & P_{cn} \\ P_{vn}^T & P_{cn}^T & P_{nn} \end{bmatrix} \in \mathbb{R}^{3+3(j+l) \times 3+3(j+l)} \tag{3.22}$$

where $P_{vv} \in \mathbb{R}^{3\times3}$ is the covariance of the vehicle pose, $P_{cc} \in \mathbb{R}^{3j\times3j}$ is the covariance of point features, and $P_{nn} \in \mathbb{R}^{3l\times3l}$ the covariance of plane features. P_{vc}, P_{vn} and P_{cn} are the correlation between each state. Now, with that definition, the same framework proposed on section 2.1.3.2 can be executed.

3.3 Feature extraction

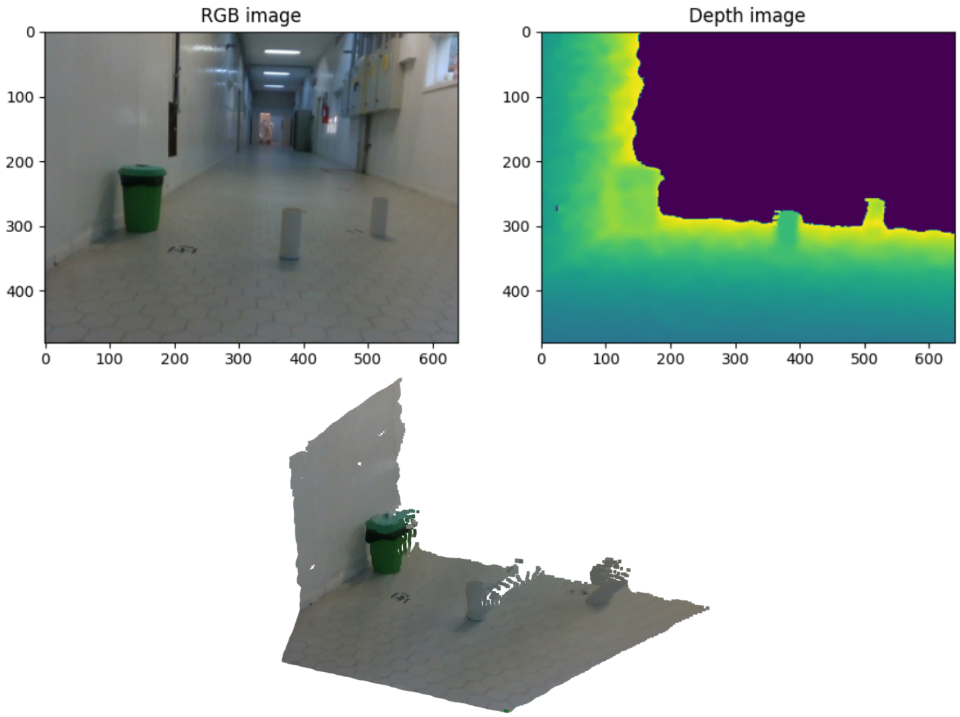


FIGURE 3.5 – Photo from the Laboratório de Máquinas Inteligentes (LMI) corridor at ITA, displaying a trash bin and two PVC tubes. Green is closer points and yellow are points further away. 1 - RGB image, 2 - Depth image, 3 - Point cloud in the 3D space. Darker color are close to the camera. Brighter color are far away. After truncation at 3m, the depth is considered zero.

Now that the features are defined, another important issue should be addressed: feature extraction. This section will discuss methods of segmentation of planar and non-planar objects, clusterization, and parametrization used in this work.

Aiming to acquire 3D information from the environment, one of today's most popular sensors is the Red-Green-Blue-Depth (RDBD) camera, which provides images of color Red-Green-Blue (RGB) along with a depth channel. This type of sensor is subject to

measurement noise based on the distance from the object to the camera (AHN *et al.*, 2019) and can deliver an image with a couple of hundreds of thousands of 3D points.

The feature extraction process is responsible to handle this set of points and simplify it to a higher level environment built with cylinders and plane segments. The plane's equation and cylinder's centroid are also input to the EKF update step, which will correct the robot's pose and the map.

Figure 3.5 shows a sample of a image taken from a Intel RealSense® camera, truncated at 3m of depth. The Intel® library already provides the alignment between the RGB and depth image. For sake of simplicity, we will consider the generated point cloud already in $\mathfrak{B}$. However, coordinate transformations are necessary since the depth information is in the Z-axis of the camera, which is actually $\vec{X}_{\mathfrak{B}}$.

3.3.1 Planar segment extraction

The first step in feature extraction is to discover the main surfaces that constitute the observed image. The plane extraction process aims to label points in the 3D space which are part of planes and isolate them from the others. Also, the extracted points must be parameterized into plane segments.

3.3.1.1 Planar RANSAC

As mentioned before, the depth image contains measurement noise which affects the extraction of characteristics in the image. Thus, the plane extraction should be robust to noise and efficient enough to give fast results for a big number of points. The RANSAC fits all these requirements and has a simple implementation, being a popular choice for this type of task.

To summarize, the RANSAC algorithm is responsible to extract a single plane equation $\boldsymbol{p}$ from a point cloud scene $\boldsymbol{\mathcal{V}} = \{\boldsymbol{v_1}, \boldsymbol{v_2}, ..., \boldsymbol{v_{n_v}}\}$ and the subset of points which belongs to that plane $\boldsymbol{\mathcal{V}_i} \subset \mathcal{V}$. The algorithm chooses randomly three points to create a candidate plane equation $\boldsymbol{p_c}$. Then, we calculate the orthogonal distance $\boldsymbol{d}$ from the plane to all points. The subset of candidate points $\boldsymbol{\mathcal{V}_c} \subset \mathcal{V}$ with distance less then a threshold $\boldsymbol{T_r}$ are inliers and considered part of the plane. This process is repeated for a predefined number of iterations $\boldsymbol{n_i}$ and the plane equation with more inliers is chosen as the best plane. The Algorithm 3 (based on Weingarten (2006)) describes this process and is illustrated in Figure 3.6.

In order to make a good plane segmentation, it is necessary to understand which parameters most affects the performance of this algorithm.

Algorithm 3 Planar RANSAC

Input $\mathcal{V}$
Output p, $\mathcal{V}_i$

$\mathcal{V}_i = \varnothing$
for $i = 1$ to n_i **do**
    $\mathcal{V}_c = \varnothing$
    $\{v_a, v_b, v_c\} \in \mathcal{V}$ are randomly chosen
    $p_c \longleftarrow$ calculatePlaneEq(v_a, v_b, v_c)
    for all v_i in V **do**
        $d \longleftarrow$ calculateOrthogonalDistance(v_i,p_c)
        if $d \leq T_r$ **then**
            $\mathcal{V}_c = \{\mathcal{V}_c, v_i\}$
        end if
    end for
    if $\#(\mathcal{V}_c) > \#(\mathcal{V}_i)$ **then** ▷ $\#(.)$ means length of the set
        $\mathcal{V}_i = \mathcal{V}_c$
        $p = p_c$
    end if
end for

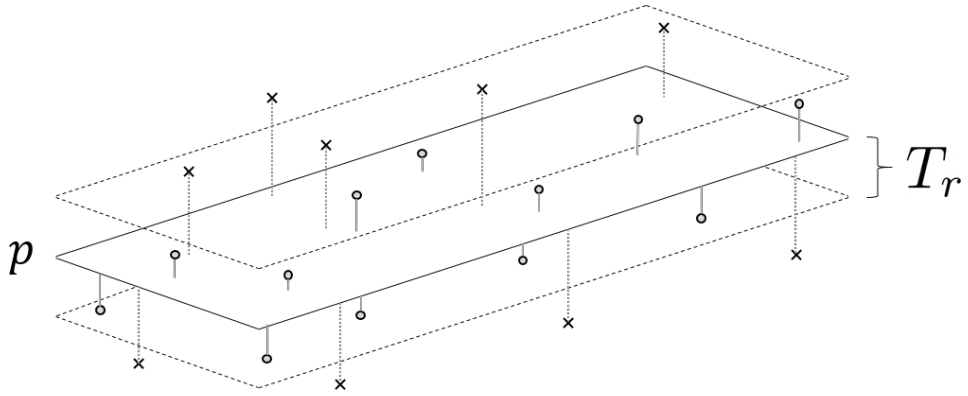


FIGURE 3.6 – Representation of RANSAC algorithm. The inliers $\mathcal{V}_i$ (dot points) are inside a threshold T_r of the plane p. The points outside (x points) are outliers.

T_r is the threshold responsible to select a thickness of the surface. A bigger T_r can be useful in handling environments with increased measurement noise. Also, depending on the application, it can simplify, objects near the walls such as windows, closed doors, electrical panels, switches, and others in being part of the wall. On the other hand, it can diminish the segmentation precision, detecting unwanted points as inliers and generating imprecise plane equations. We found that a good range of values for this parameter is between 4cm and 12cm (used 5cm).

n_i is the number of iterations used in RANSAC. The bigger this parameter is, the more chances it has to find the best plane, but also more time it takes to finish it. For this work, we used $n_i = 1000$ which produced a decent plane estimation in a reasonable amount of time.

3.3.1.2 Plane segment parameterization

Until now, we were only worried about the infinite plane equation. However, it is also necessary to find the region in that plane in which the planar segment exists in the 3D space. This calculation is divided into two steps: 2D projection and calculation of Minimum Area Rectangle (MAR).

Consider that the plane p_i and its inlier points $\mathcal{V}_i$ from previous step are defined in a coordinate frame $\mathfrak{p} = \{\vec{X}_\mathfrak{p}, \vec{Y}_\mathfrak{p}, \vec{Z}_\mathfrak{p}\}$ and are attached to a plane which its normal is the unit vector $\vec{n}$. The plane's 2D projection can be computed by first rotating the plane and aligning its normal to $\vec{Z}_\mathfrak{p}$ and then, suppressing the z values from the points and consequently, projecting all points to $\vec{X}_\mathfrak{p}$ $\vec{Y}_\mathfrak{p}$ plane. The following paragraph explains this process in detail.

We can use the Rodrigues' rotation formula to make an orthogonal basis transformation $\mathbb{R}^3 \rightarrow \mathbb{R}^3$ by calculating a rotation matrix. In other words, $R_p\{\vec{n}, \vec{Z}_\mathfrak{p}\}$ transforms the coordinate frame of the plane so it becomes aligned to its $\vec{X}_\mathfrak{p}$ $\vec{Y}_\mathfrak{p}$ plane.

The minimal rotation between two vectors is done through its orthogonal vector, therefore:

$$\vec{w^r} = \vec{n} \times \vec{Z}_\mathfrak{p} \tag{3.23}$$

where $\vec{w^r} = [w_1^r \quad w_2^r \quad w_3^r] \in \mathbb{R}^3$ is the rotation vector. Equation 3.24 shows the calculation of the Rodrigues' formula, where $W_\times$ is the skew-symmetric cross-product matrix of $\vec{w}$ (BRANNON, 2018).

$$R_p\{\vec{n}, \vec{Z}_\mathfrak{p}\} = I + W_\times + W_\times^2 \frac{1 - \vec{n} \cdot \vec{Z}_\mathfrak{p}}{|\vec{w^r}|^2} \qquad W_\times = \begin{bmatrix} 0 & -w_3^r & w_2^r \\ w_3^r & 0 & -w_1^r \\ -w_2^r & w_1^r & 0 \end{bmatrix}. \tag{3.24}$$

Finally, a point $v_i \in \mathcal{V}_i$ is rotated through $v_i' = R_p\{\vec{n}, \vec{Z}_\mathfrak{p}\}(v_i)$ and projected $\mathbb{R}^3 \rightarrow \mathbb{R}^2$ by removing the z-value resulting in $v_i^{2d} = [v_{i\,x}' \quad v_{i\,y}']^T$. Then, the final step uses a minimum area rectangle (Chelishchev, Petr; Sørby, Knut, 2020) to calculate the parameters of the segment. $\boldsymbol{H_p}$ is the plane's height, $\boldsymbol{W_p}$ is the width, $\boldsymbol{\Theta}$ is the 2D rotation angle and $\boldsymbol{c_{2d}} \in \mathbb{R}^2$ is the coordinates of the center in the plane's 2D projection. Figure 3.7 shows the process above.

3.3.1.3 Complete process

The steps detailed above are supported by preprocessing techniques such as filters, downsample, and validation gates. Figure 3.8 describes the integrated process that will

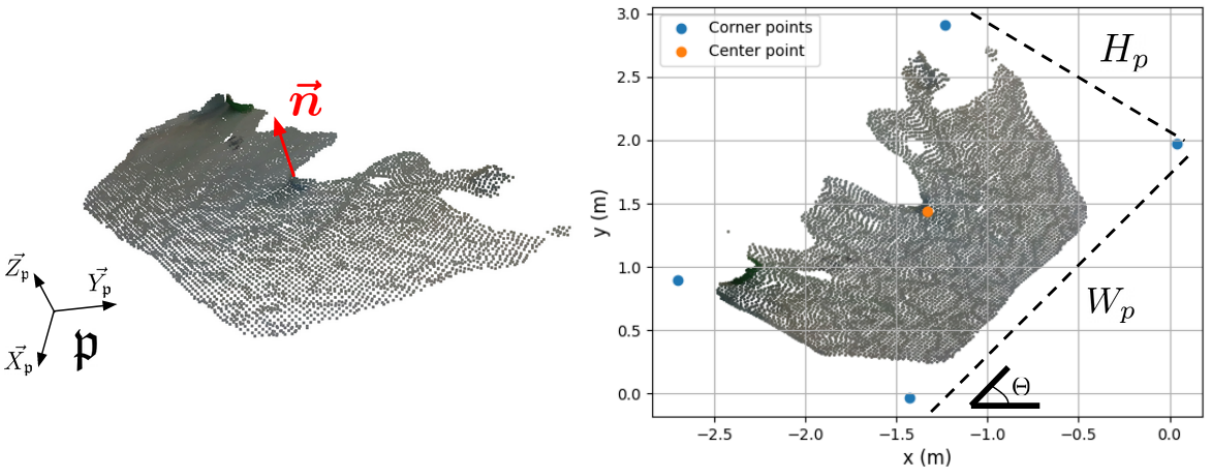


FIGURE 3.7 – Right image shows the floor from Figure 3.5 after downsampled in $\mathbb{R}^3$. Left image show its projection and representation of parameters. In this case $H_p = 1.60$, $W_p = 2.47$, $\Theta = 52.17°$, $c_{2d} = [-1.32 \quad 1.43]$.

be discussed in this section.

The flowchart shows the original scene $\mathcal{V}$ being decomposed into multiples planes $\{\mathcal{V}_{i1}, \mathcal{V}_{i2}, ..., \mathcal{V}_{in}\}$ and the non-planar points $\boldsymbol{\mathcal{V}_o}$. This last subset is created by joining the invalid planes with the remains of points after the full plane extraction. $\mathcal{V}_o$ is the input to the next step of feature extraction, and are the gray points from the last scene. The loop finishes when enough points ($\boldsymbol{N_{\%min}}$) from the original scene were extracted, which can be normalized as a percentage of the original number of points.

Below, a more in-depth explanation of each one of the blocks:

- **Radial filter 1** This type of filter uses two parameters: $\boldsymbol{N_{rf}}$ is the minimum number of neighbors that must exist inside a sphere of radius $\boldsymbol{R_{rf}}$, centered in the analyzed point. The radial filter verifies each point of the scene and if it does not have enough neighbors, $n \leq N_{rf}$, the point is removed from the scene. This filter removes points that are further away from the others which helps remove measurement noise. Specifically, Radial Filter 1 is applied before the RANSAC method and helps to clean up the outlier points from previous plane extraction. The blobs of points removed by the filter are illustrated as red dots in Figure 3.8.

- **Radial filter 2** is applied on the extracted plane $\mathcal{V}_i$. In situations where the camera is close to walls, RANSAC extracts the plane of the wall before the floor. This makes points belonging to the floor understood as a wall, which causes errors in parameterization, especially on corners. The radial filter is applied to only maintain the main portion of the plane, as shown by Figure 3.9. Notice that small values of N_{rf} can cause unwanted effects such as rounding the corners and changing the plane's overall dimensions.

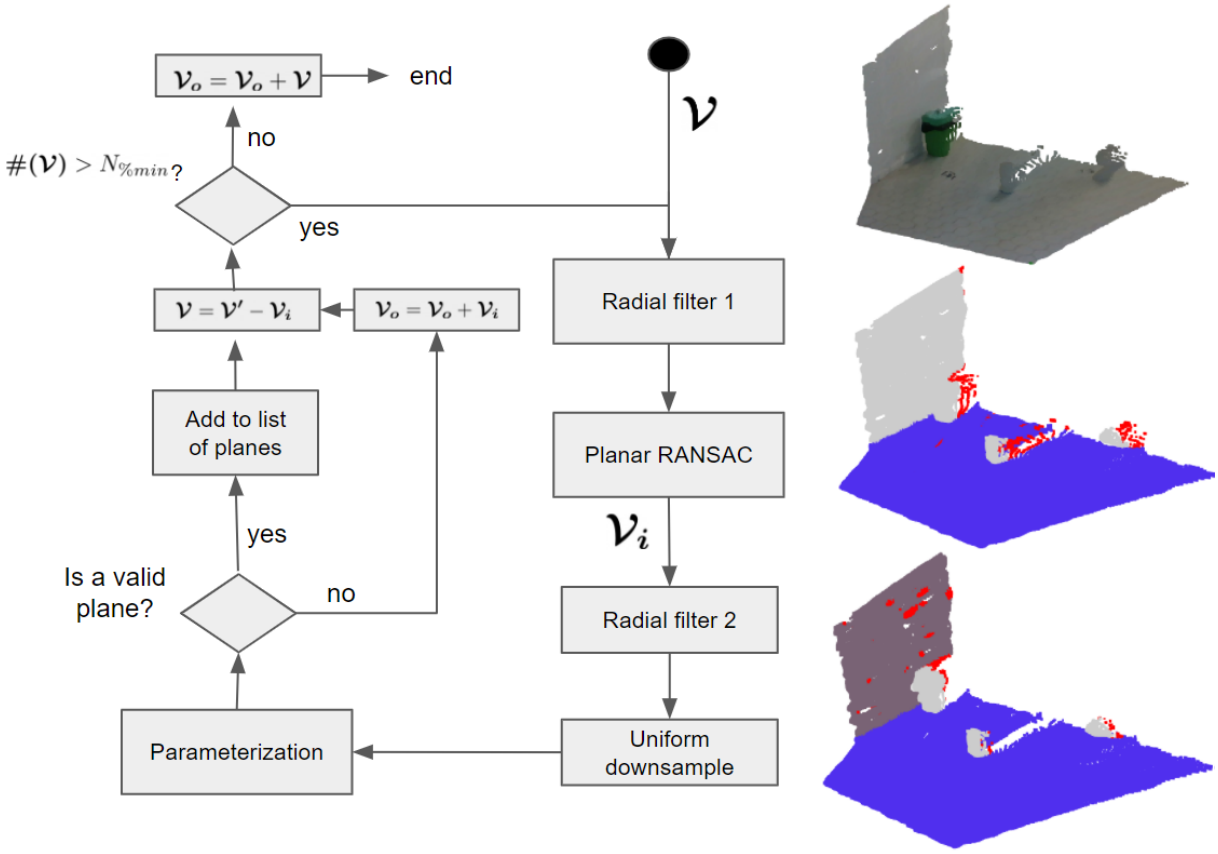


FIGURE 3.8 – Flowchart of the planar segment extraction. On the right, the extraction of two planes (purple and brown) from Figure 3.5. The points filtered by Radial Filter 1 is highlighted in red on each step.

- **Uniform downsample** reduces the number of points in the scene but maintains its geometric form. The downsample creates a grid of size $\boldsymbol{L_{ds}}$ in the point cloud and allows only one point on each voxel. It not only reduces the number of points in the plane but also makes uniform density in the plane.

- The **Validation gate** is responsible to validate the planes in terms of it *size*, *density* and *completeness*:

  1. The **centroid validation** verifies if the centroid of the plane's point cloud are near the parameterized center point $\boldsymbol{c_{3D}}$ (c_{2D} after its inverse transformation back to $\mathbb{R}^3$). An ideal plane extraction expects both points in the same coordinate. However, sometimes a point from distant objects passes through the radial filter and makes parameterization of regions that are not connected. Thus if the distance is more than $\boldsymbol{D_{cv}}$ the plane is invalid, as described in eq. (3.25) where $v_i \in \mathcal{V}_i$.

$$|\mu(\mathcal{V}_i) - R_p\{\vec{n}, \vec{Z}_\mathfrak{p}\}^{-1}(c_{3d})| < D_{cv} \tag{3.25}$$

  2. The **density validation**, described by eq. (3.26), verifies if the plane's points

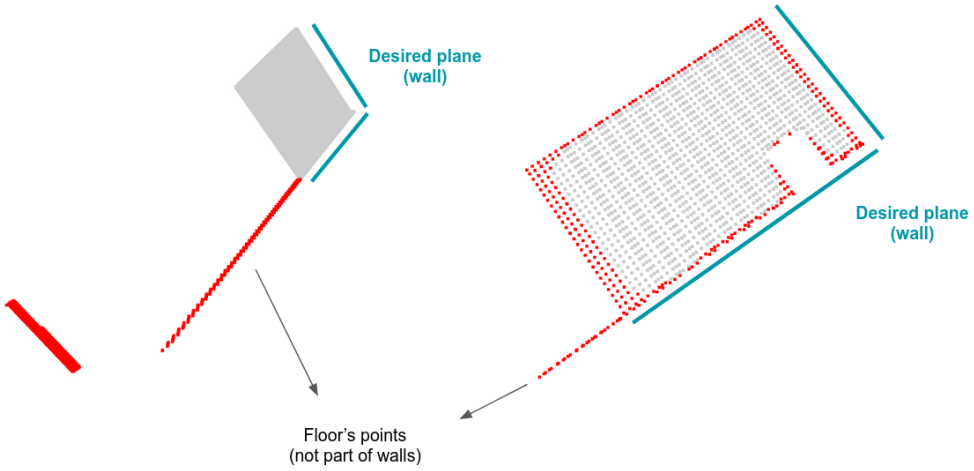


FIGURE 3.9 – Example of points filtered by Radial Filter 2 (red dots).

fill the whole planar segment. Since the points were already downsampled, it is ideally expected to find one point for each L_{ds}^2 of area. Therefore, a parameter $\boldsymbol{\delta_{min\%}}$ describes the percentage of completeness allowed in a valid plane. A plane with low density means there are holes and gaps in its structure, which shouldn't be validated. For instance, the plane from Figure 3.7 has a $\delta_{\%} = 85.7\%$ which is greater then the $\delta_{min\%} = 75\%$ that was used in the project, so it is considered valid.

$$\frac{\#(\mathcal{V}_i)}{H_p W_p} > \delta_{min\%} \frac{1}{L_{ds}^2} \tag{3.26}$$

3. Finally, the **area validation** allows only planes with area greater then $\boldsymbol{A_{min}}$ to be considered valid. For instance, both a wall and a book cover are planar surfaces. However, the smaller the surface is, the harder it is to make correct data association between the poses. This can result in catastrophic erroneous pose correction in the EKF. Thus, walls, floors, and other big surfaces should be considered planes, but not small objects.

$$H_p W_p > A_{min} \tag{3.27}$$

3.3.2 Cylinder extraction

All non-planar objects are simplified as cylinders in the 3D space. The output from the last step, the set of points $\mathcal{V}_o$, is now input to the cylinder extraction. The goal of this step is to find the centroid, height and radius of a cylinder-equivalent of an arbitrary object in the scene. The flowchart in Figure 3.10 shows the steps for the cylinder extraction.

3.3.2.1 DB-SCAN

Now, $\mathcal{V}_o$ contains a set of points clusterized in groups where each group is an object. The Density-based spatial clustering of applications with noise (DB-SCAN) algorithm is responsible to iterate over this set of points and segment them into groups. To do that, the algorithm chooses a core point and analyses the number of neighbors inside a distance of radius D_{eps}. In case this quantity is bigger than N_{db}, the point is considered part of that cluster. This clustering process is illustrated in the first two images in Figure 3.10 where each object is colored differently (red, blue, and pink). We found that reasonable values for D_{eps} are between 5 cm and 20 cm and the N_{db} can be set as a percentage of the total number of points in the scene. For this work we used $N_{db} = \#(\mathcal{V})0.01\%$

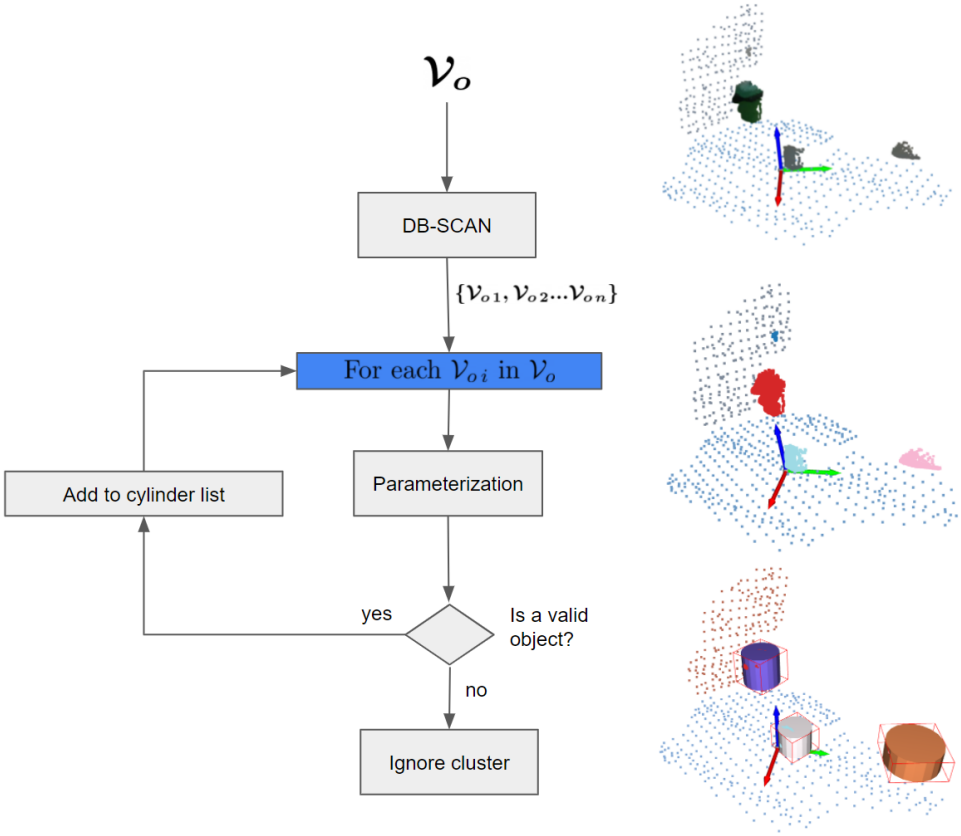


FIGURE 3.10 – Flowchart of the cylinder extraction. On the right, the extraction of three features from Figure 3.5.

3.3.2.2 Cylinder parameterization

Every cluster from the previous step is now a point cloud of a candidate cylinder feature. This section describes how to calculate parameters from that feature, more specifically, the centroid $\boldsymbol{c_{cyl}}$, height $\boldsymbol{H_{cyl}}$, and radius $\boldsymbol{r_{cyl}}$. Some important considerations can be stated:

- This work assumes that the cylinder has its axis aligned with the floor's normal $\vec{n}_{fl}$. No tilted or oblique cylinders are allowed.
- The goal is to simplify arbitrary object with cylinder and its radius is a conservative approximation of the bounds of the real-life object.
- Except for the hidden part of an object which is not seen from the camera, the algorithm assumes that the object is fully inside the bounds of the image.

Finding the radius and centroid of an object is only a trivial task when its point clouds are complete and noiseless. In reality, the acquired point cloud is only a partial view of the object, along with measurement noise. Hence, the algorithm must be capable to estimate the complete object without the noise.

The algorithm proposed here is divided into two steps. First, a rough approximation of the centroid $\boldsymbol{c'}$ and a rough approximation of the radius $\boldsymbol{r'}$ is done by directly analyzing the point cloud. The second part recalculates both parameters estimating a symmetric hidden part. Below is a detailed explanation. The first step is illustrated in Figure 3.11.

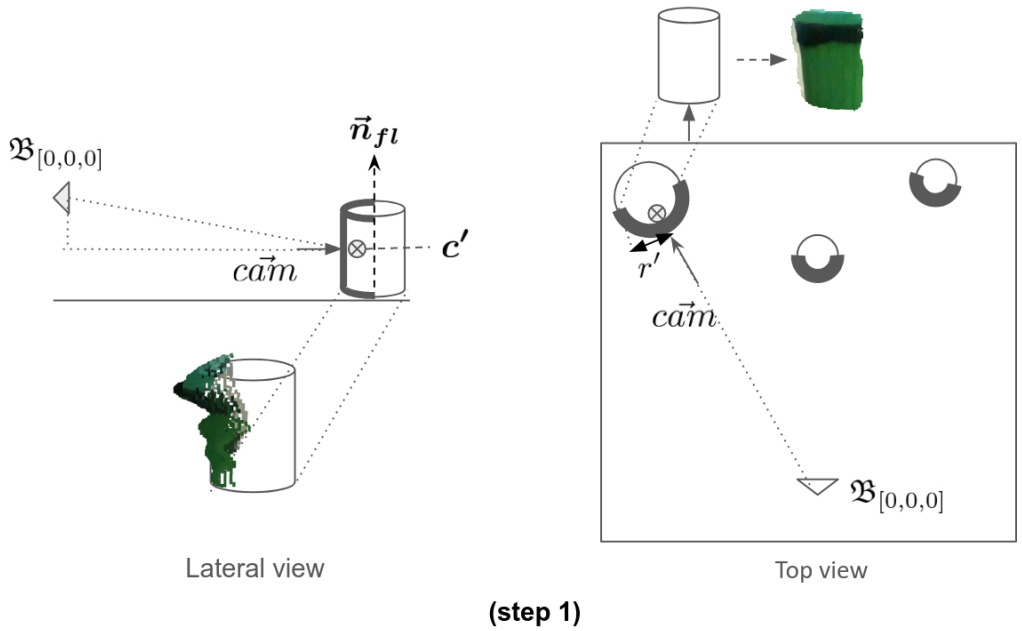


FIGURE 3.11 – First step of cylinder parameterization.

To simplify the explanation, we assume the visible part of the object corresponds to an angle of view of 180°, but the solution can be easily generalized to any angle depending on the camera configuration.

The rough radius r' is computed from the maximum and minimum bounds of the point cloud facing the camera vector (aligned with the radius axis $\vec{r'}$ where $\vec{r'} \perp \vec{n}_{fl} \perp \vec{cam}$).

Therefore:

$$\vec{r'} = n_{fl} \times \vec{cam} \qquad || \qquad \vec{cam} = \frac{proj_{floor}(\vec{c'})}{||proj_{floor}(\vec{c'})||} \tag{3.28}$$

where the vector $\vec{c'}$ is the direction from the orign of $\mathfrak{B}$ (camera) to the centroid of the i-th point cloud cluster $\mathcal{V}_{oi}$ and $\vec{cam}$ is the unit vector of the camera direction parallel to the floor's plane.

$$proj_{floor}(\vec{c'}) = \vec{c'} - \frac{\vec{c'} \cdot \vec{n}_{fl}}{||\vec{n}_{fl}||^2}\vec{n}_{fl} \qquad \vec{c'} = \mu(\ \mathcal{V}_{oi}\ ) \in \mathbb{R}^3 \tag{3.29}$$

The centroid of a semi-circle shell is displaced $\frac{4r'}{3\pi}$ from the centroid of the complete

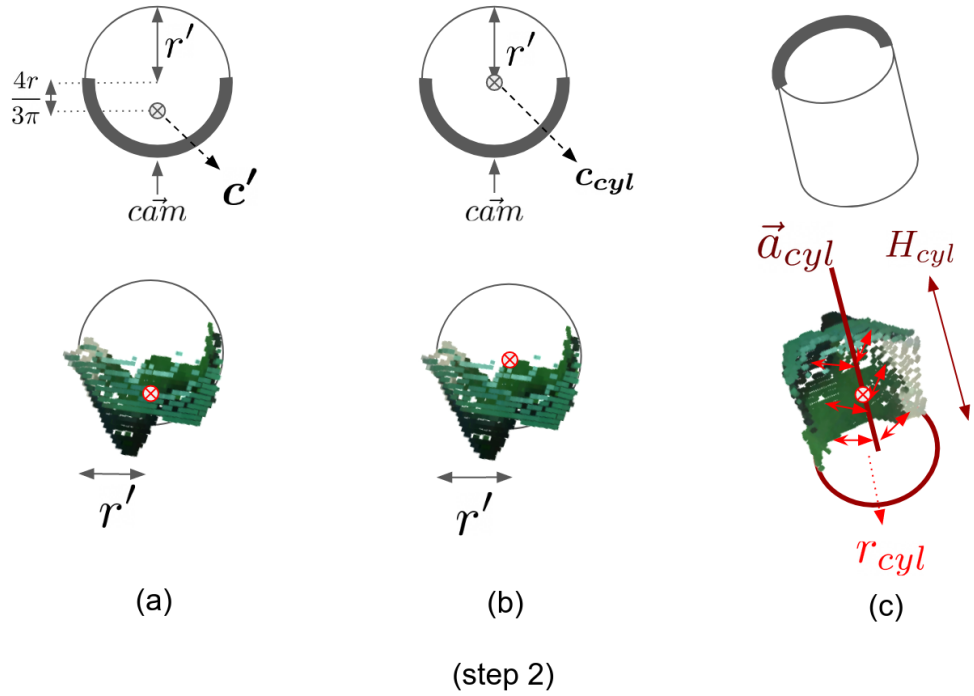


FIGURE 3.12 – Second step of cylinder parameterization.

circle. So, the second step translates the rough centroid to its final position c_{cyl}. Then, it recalculates the radius r_{cyl} based on the distances from each point of the cluster to the cylinder's axis:

$$\mathcal{R}_{cyl} = \{||\vec{n_{fl}} \times (c_{cyl} - v_{oi})||\} \qquad c_{cyl} = c' + \frac{4r'}{3\pi}\vec{cam} \tag{3.30}$$

$$r_{cyl} = \mu(\ \mathcal{R}_{cyl}\ ) + 2\,\sigma_{rcyl} \qquad \sigma_{rcyl} = \sigma(\ \mathcal{R}_{cyl}\ ) \tag{3.31}$$

where, $\mathcal{R}_{cyl}$ is the set of distances from each point of $\mathcal{V}_{oi}$ to cylinder axis $\vec{a}_{cyl} = (\vec{n}_{fl}, c_{cyl})$ and $v_{oi} \in \mathcal{V}_{oi}$. The radius r_{cyl} is calculated with the mean of the radii plus a number of percentile. In this work we choose to use $2\,\sigma_{rcyl}$ to account 95% of points. Finally, H_{cyl} is

calculated by measuring the range of points in direction of $\vec{a}_{cyl}$.

3.3.2.3 Validation gate

A cylinder is considered valid when met the following criteria:

- The **radius validation** bounds the cylinder's radius to a fixed amount $\boldsymbol{R_{max}}$. Is useful to avoid wrong objects, such as part of planes to be considered cylinder.

$$r_{cyl} < R_{max} \tag{3.32}$$

- The **spreadness validation** uses a normalized standard deviation to verify the *cylinderness* of the object. This parameter measures how well the point cloud fits the cylinder shell and seeks to invalidate noise blobs to be considered a valid object. This is a very important parameter for this work since it can be adjusted to be more or less sensitive to different types of objects. If it is greater than a threshold $\boldsymbol{\tau_{max}}$, it is not valid.

$$\tau = \frac{\sigma(\ \mathcal{R}_{cyl}\ )}{\mu(\ \mathcal{R}_{cyl}\ )} < \tau_{max} \tag{3.33}$$

3.4 Data association and feature growth

Until now we discussed how to extract planar segments and cylinders from the point cloud. We also defined how to represent that features inside the EKF framework. We now need to define how to relate new measurements with the older ones and how the feature will behave itself when updated.

3.4.1 Data association

Due to the probabilistic nature of the EKF, it is also interesting to implement a probabilistic approach for data association. Consider a previously observed feature in $\mathfrak{B}$, z_{k-1} and a new measurement $z_k^{\#}$ which we desire to evaluate association. Instead of the classic euclidean distance $d = |z_k^{\#} - z_{k-1}|$ the Mahalanobis Distance (MD) is used to calculate a scale-invariant distance. The MD can be calculated as follows:

$$d_M(z_k^{\#}, z_{k-1}) = \sqrt{(z_k^{\#} - z_{k-1})^T . P^{-1} . (z_k^{\#} - z_{k-1})} \tag{3.34}$$

where P is the covariance in the same coordinate frame as the measurements. Hence an acceptance region can be defined to test the hypothesis of matching two features:

$$d_M(z_k^{\#}, z_{k-1}) < D_{M\,max} \tag{3.35}$$

Having two different types of features helps in the data association process. When a measured feature is a plane, only other planes are possible matches. The same goes for points (or, in this case, cylinders) that only match other points. When no association is found, the feature is added to the map as a new feature. In this work, the loop closure also happens by associating features with the MD.

3.4.1.1 Handling planar equation ambiguity

Consider the scene from Figure 3.13 where an entrance to a corridor is positioned between two walls. Imagine that $z_{p\,2}$ was a new observed plane and $z_{p\,1}$ is a previously known feature. Due to the infinite nature of the plane equation, $\boldsymbol{z_{p1} = z_{p2}}$, and therefore, the association between both walls are inevitable when analysing only it's equation. This

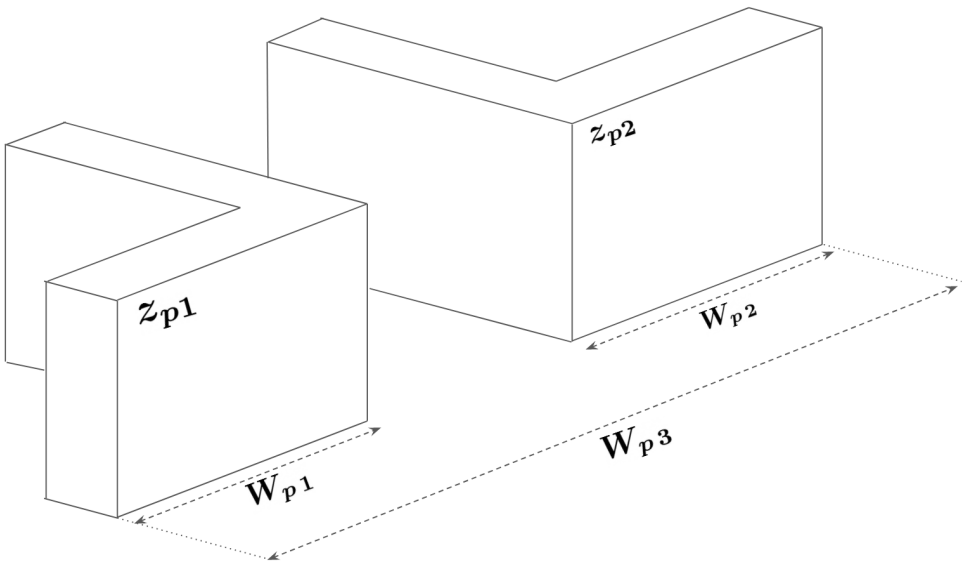


FIGURE 3.13 – Example of a scene where the plane equations $z_{p1} = z_{p2}$ but its associations means closing the corridor's entrance. Therefore both plane should be considered different features even having same equations.

would not be a problem when building the map with point cloud since the projection of the points into the plane would be limited to the segment where the wall exists, resulting in a single third plane $\boldsymbol{z_{p3}}$ with width $\boldsymbol{W_{p3}}$. However, the parameterized plane segment would block the entrance of the corridor.

Having a consistent parameterized plane gives information about a high level simplified environment. This enables the features to be **comprehended independently of its point**

cloud which is essential if the application has limited memory and is not interested in a high fidelity map reconstruction. So, accepting the creation of the z_{p3} would mean lower the scene understanding of the simplified map, and should be avoided.

The workaround for this problem is to invalidate an association that results in a dimension growth of the plane that is much larger than the observed feature. Hence, $\boldsymbol{G_{\%\,max}}$ is the max percentage of the size of $W_{p\,2}$ which a plane is allowed to grow to still be valid. This same criterion is used for the orthogonal growth H_p.

$$\frac{W_{p\,3} - W_{p\,1}}{W_{p\,2}} < G_{\%\,max} \tag{3.36}$$

Consequently, the EKF must couple with features that have equal parameters. The solution to this problem is to vectorize the planes that comply with eq. (3.35) and verify the association of each one in ascending order of $d_M(z_k^{\#}, z_{k-1})$. If no association is valid, the plane is added to the state vector as a new feature.

3.4.2 Feature growth

It is very difficult for a single RGDB image to fully reconstruct an object in the 3D space. This happens due to the hidden part of objects (which are not visible from the camera view) and to the crescent dimensions of a wall when moving through an environment. The centroid position of the cylinder and the plane's equation is updated in the EKF framework when a new measurement is made during the update step from the Algorithm 2. But how should behave other parameters such as H_p, W_p, r_{cyl} and H_{cyl}? How to merge the previously observed point cloud to the measured one? This section describes the solution to these questions.

3.4.2.1 Planar segment growth

After a plane is associated during the process depicted above its equation is updated by the EKF. It is necessary, however, to establish rules to update the other parameters of the plane based on the new measurement. A straightforward procedure can be used to handle that problem.

Consider, the point cloud from the previously observed feature $\mathcal{V}_{i\,k-1}$ and the newly observed plane's point cloud $\mathcal{V}_i^{\#}$. The following steps summarize the plane growth

1. Both $\mathcal{V}_{i\,k-1}$ and $\mathcal{V}_i^{\#}$ are joined into another set of points $\mathcal{V}_i^{'}$.

$$\mathcal{V}_i^{'} = \mathcal{V}_{i\,k-1} + \mathcal{V}_i^{\#} \tag{3.37}$$

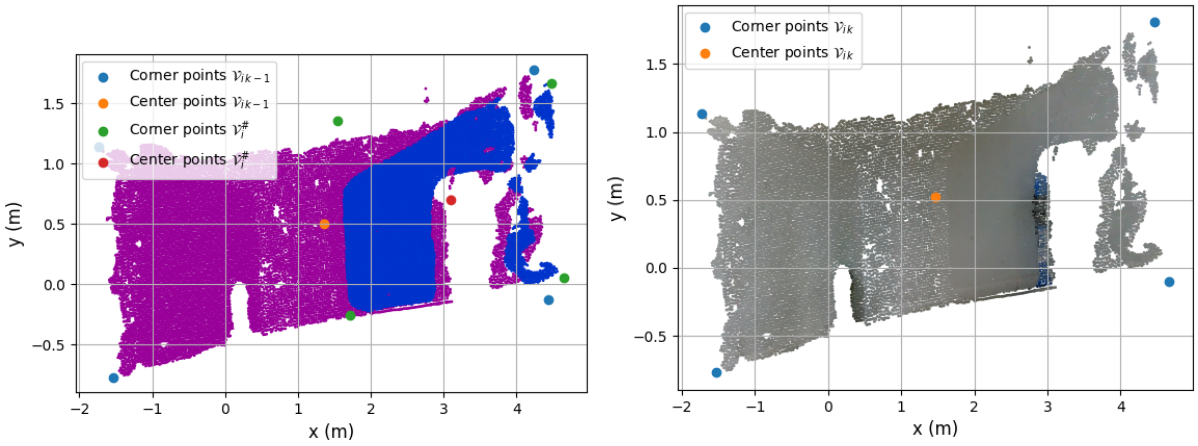


FIGURE 3.14 – On the left image, the purple points are projections from the older feature $\mathcal{V}_{i\,k-1}$ and the blue points are result from the new observation $\mathcal{V}_i^{\#}$. They both are projected into the same plane and the parameters are recalculated. The resulting point cloud is displayed on the right.

2. Then, each point $v_i^{'} \in \mathcal{V}_i^{'}$ is transformed using the Rodrigues Formula from the new plane equation z_k to its z-axis $\vec{Z}_{\mathfrak{p}}$.

$$\mathcal{V}_i^{*} = \underset{\mathbb{R}^3 \to \mathbb{R}^2}{R_p\{\vec{z_k}, \vec{Z}_{\mathfrak{p}}\}}(v_i^{'}) \tag{3.38}$$

3. The MAR algorithm from section 3.3.1.2 is recalculated and the new planar segment's dimension is obtained. Figure 3.14 shows the reparameterization.

4. Finally, the resulting plane is downsampled to the same grid size as before ($\boldsymbol{L_{ds}}$).

The overall result of this algorithm in the 3D space is the direct projection of both point clouds into a new plane as illustrated in Figure 3.15. Notice that for a minimal set, only the corner points (Figure 3.14) of each point cloud are necessary to reparameterize the planar segment. Using this minimal set leads to faster MAR algorithm but also overestimates the size of the plane on each step. For a more optimized (but slower) reparameterization, the whole point cloud can be used.

3.4.2.2 Cylinder growth

Different from the planes, the cylinder is not a planar feature. Assuming that a cylinder was recently updated by the EKF, there are three distinct moments: (I) The cylinder's centroid before the update z_{k-1}, (II) The newly observed cylinder's centroid $z^{\#}$, and (III) the centroid after the update z_k. Since the point cloud is attached to the object whose centroid is being updated, it is also possible to apply the same correction to each point of

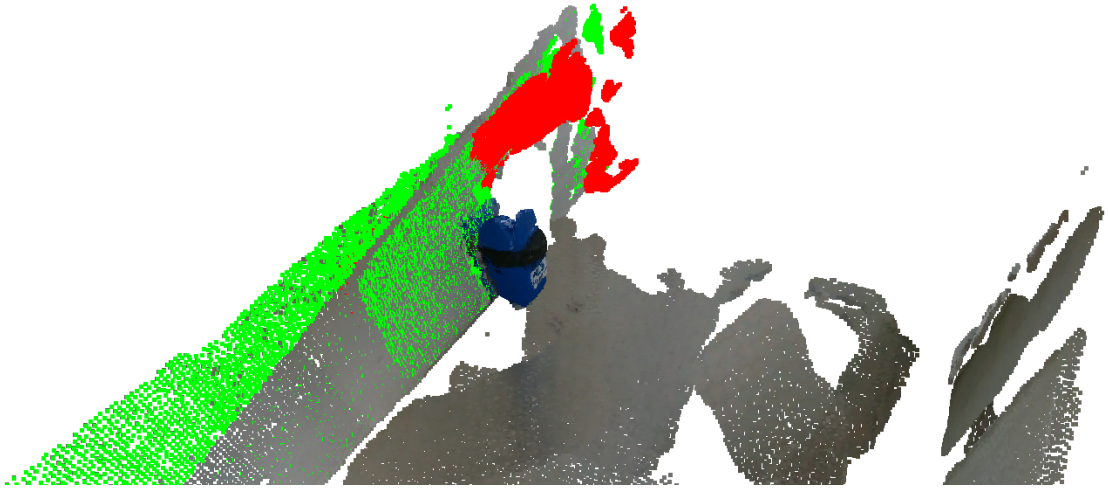


FIGURE 3.15 – Red points are the new observation (tilted inwards to the corridor) and gray is the older feature. Green point cloud is the resulting plane after the update, where the points of both point clouds are projected into.

the point cloud.

$$d_{k-1} = z_k - z_{k-1} \tag{3.39}$$

$$d^{\#} = z_k - z^{\#} \tag{3.40}$$

Let d_{k-1} be the displacement from the last state (eq. 3.39), and $d_{\#}$ the displacement of the observation (eq. 3.40). Each point of its respective point cloud is also corrected by these displacements and appended to the same set.

$$\mathcal{V}_{o\,k} = \{v_o^{\#} + d^{\#}\} + \{v_{o\,k-1} + d_{k-1}\} \tag{3.41}$$

where $v_o^{\#} \in \mathcal{V}_o^{\#}$ and $v_{o\,k-1} \in \mathcal{V}_{o\,k-1}$. Figure 3.16 shows the correction of a feature.

Addressing the parameters H_{cyl} and R_{cyl}, the solution adopted for this work is to average the parameters from the observed feature and from the previous state.

3.5 Map representations

During the sections above, two types of maps were considered: A high level parameterized map comprised of planar segments and cylinders; and a point-cloud-based map. This section aims to discuss additional details from each type of map and present other possible alternatives.

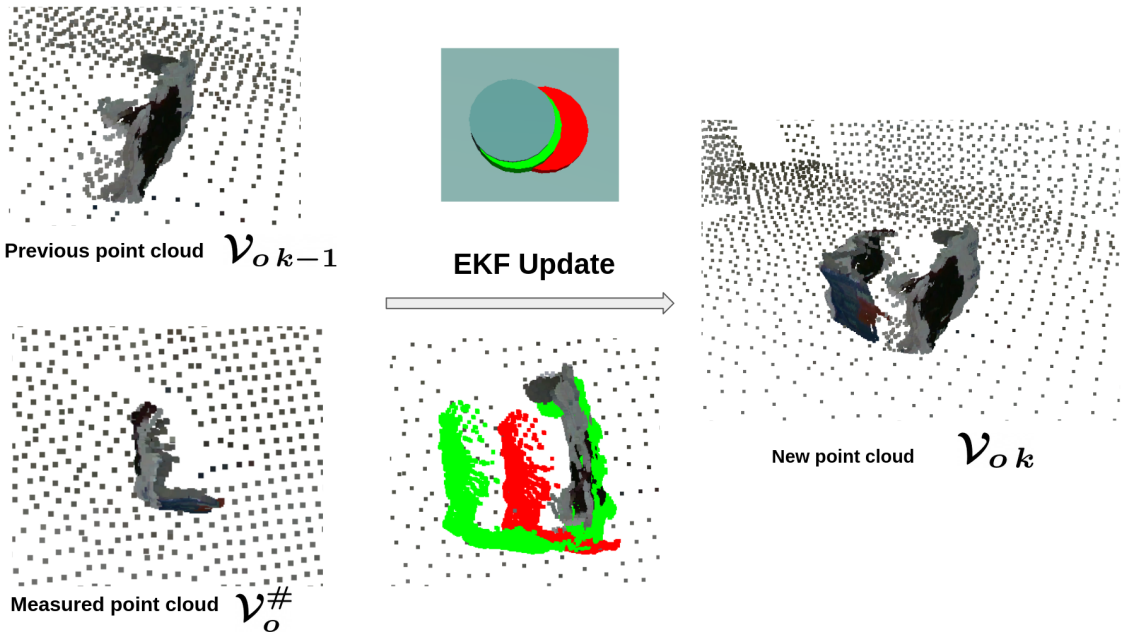


FIGURE 3.16 – Two observations of a box (frontal and back) that resulted in a correction for the EKF. Red is the observation, green is the resultant feature.

3.5.1 High level map and point cloud

The high level map is a representation of the parameterized values from the planar segment and cylinder. It does not just contain the variables that are input to the EKF but also additional values to create a minimal 3D environment.

To generate this map, the point cloud is only needed before the parameterization, making it useful for memory-critical applications, since no point cloud needs to be stored. However, the simplified environment is not able to recreate a visually realistic map.

On the other side, we showed it is also possible to carry the point cloud together with the parameterized feature. The point cloud is a rich representation but is inefficient in memory and time costing. The more observations, the more points need to be processed and corrected on each step.

An interesting tool to compare both types of maps is to create rough memory models. On the high-level map model, the size of each parameter's variable is used. On the point cloud, only its position and color are considered. The resulting model is summarized in Table 3.2

TABLE 3.2 – Summary of rough memory model for the high level map and point cloud

	Bytes
Planar segment	35
Cylinder	23

Single point (from a point cloud)	15

A complete table detailing all variables present in primitive features is attached in Appendix A.1, and the detailed model from the point cloud is presented in Appendix A.2. The estimation of the models presented above is based on a C++ general implementation. Notice that it is a simplified model and does not intend to exactly represent the memory allocation, being used only for comparison purposes. Figure 3.17 shows a scene reconstructed as both types of map.

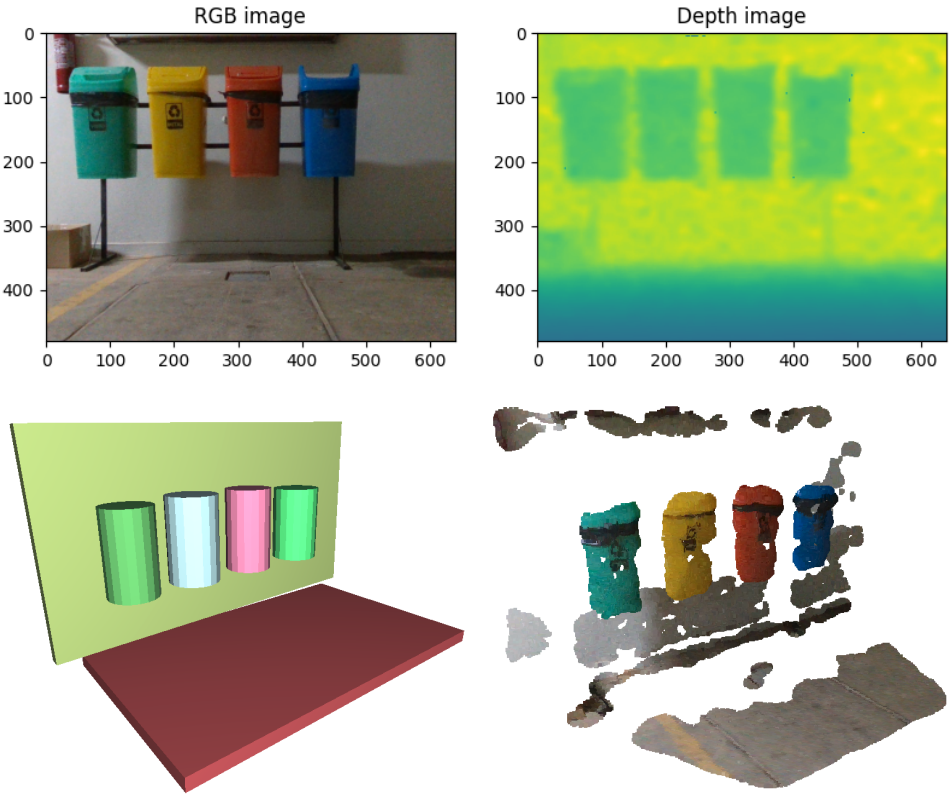


FIGURE 3.17 – The top image is the RGB input and its depth image. The left bottom image is the high level scene, and on right, the point cloud reconstruction

3.5.2 Hybrid map

In some situations, using point cloud is not necessary at all, but only cylinders and planes are a too simplified representation. Another approach is to utilize more types of

primitive features, such as cuboids, cones, spheres, and torus. However, parameterizing some of these features in a local scene is not a trivial task because the RGBD photo is only a partial view of the real-life object.

In a local scene, it is not always possible to be sure what kind of primitive geometry the point cloud can be simplified. For example, a cuboid is defined as a geometric shape that has three perpendicular planes to each other. If the robot sees the cuboid from the front, it may only measure a single face of it. However, when exploring the map, other points belonging to the cuboid can be obtained. Then after a few steps, the three perpendicular planes are observed in this feature.

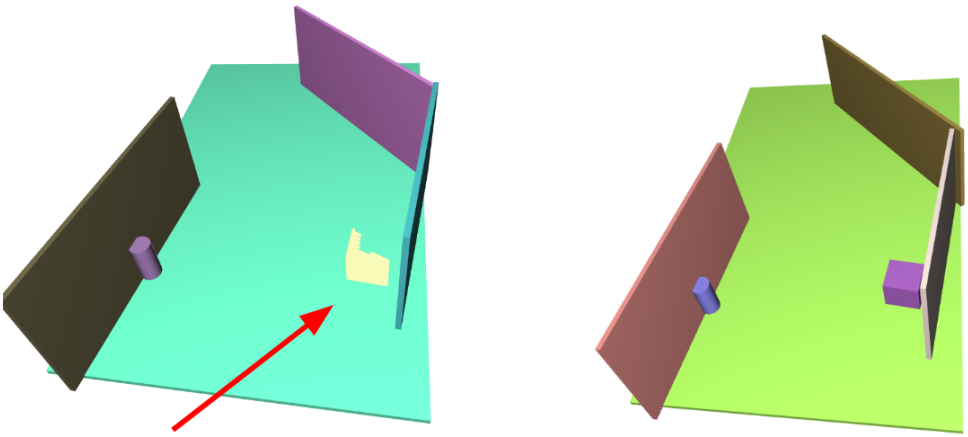


FIGURE 3.18 – Example of a hybrid map. The point cloud of the cuboid is simplified to a parameterized cuboid on the right.

In these cases, it is possible to mix the above techniques and create a hybrid map. In this strategy, the map is primarily reconstructed with primitive geometric shapes (planes, cylinders, cuboids). However, as long as a primitive shape is not defined, the point cloud of that feature will be carried until there is a better understanding of the map.

This way, point clouds very similar to primitive shapes are simplified, reducing memory consumption and processing time. On the other hand, features with irregular or non-standard shapes continued to be carried by the algorithm as a point cloud.

Notice that now, in the hybrid map, when a feature is classified as a cylinder, it is a *defined cylinder*. This means the point cloud was classified and parameterized as this shape, being understood the real-life object is actually similar to a cylinder and not only an arbitrary object.

In this type of strategy, the challenge is to create clear criteria to define the primitives, as well as extract their properties. This work implemented a hybrid map with planes, cylinders, and cuboids as a proof of concept. The classification strategy is explained below:

- Plane features from the high-level world are directly classified as planes in the hybrid representation.
- If a feature is unclassified (represented as a point cloud), the algorithm will try to fit into two categories: cuboid or cylinder:
  1. First, the algorithm tries to fit the point cloud as a cuboid:
     (a) It runs consecutive RANSAC extractions until just a few points are left.
     (b) If at least three planes are detected and they are orthogonal to each other the feature is considered a cuboid.
     (c) The planes are parameterized, and used as the cuboid's height $\boldsymbol{H_{cub}}$, width $\boldsymbol{W_{cub}}$, depth, $\boldsymbol{D_{cub}}$. The centroid from the feature's high-level representation is used as cuboid's centroid $\boldsymbol{c_{cub}}$.
     (d) The normal vector of a plane that are orthogonal to the ground is used to calculate its 2D orientation[2] $\boldsymbol{\Theta_{cub}}$ in $\vec{Z}_{\mathfrak{I}}$.
  2. If the cuboid classification failed (i.e: didn't detect three planes) :
     (a) The $\boldsymbol{\tau}$ parameter from eq.(3.33) is recalculated with the full point cloud, but now using a more sensible value of its *cylinderness* $\boldsymbol{\tau'_{max}}$
     (b) If $\tau < \tau'_{max}$ the feature is considered a cylinder.
     (c) The parameters from the high-level world is passed directly to the cylinder in the hybrid map.
  3. If both criteria failed, the feature will continue undetermined and being mapped a point cloud.

The memory model for each primitive is summarized on Table 3.3.

TABLE 3.3 – Summary of rough memory model for parameterized features in the hybrid map

	Bytes
Plane	35
Cylinder	23
Cuboid	27
Sphere	19
Single point (from a point cloud)	15

[2]As simplification purposes we assume the cuboid has always a face parallel to the floor.

3.5.3 Voxel grid map

There are alternative ways to reconstruct a map other than with a point cloud or simplifications using primitive features. One of these strategies is to discretize the map into voxels, creating a matrix in the 3D space. This map can be constructed from point clouds, by analyzing the existence of the points inside the bounds of a voxel.

In terms of memory, state of art strategies for creating voxel grid maps rely on performing dynamic memory allocation with containers structures. For instance, the structure *unordered map* is used to allocate an occupied cell by the Open3D (ZHOU *et al.*, 2018) library. With this structure the memory model is summarized in Table 3.4 and detailed in Appendix A.3.

TABLE 3.4 – Summary of rough memory model for voxel grid

	Bytes
Grid	144
Bucket container	16
Cell	23

FIGURE 3.19 – Example of a voxel grid scene.

3.5.4 Octree

Another way of representing maps is through sparse voxels. One of these techniques widely used is the octree.

There are two ways to stop a subdivision of an octree:

- **Truncation in a predefined number of subdivisions**: This strategy provides control over the memory since the number of subdivisions will always be the same regardless of the object being represented. In this method, the worst-case number of leaf nodes is 8^N where N is the number of subdivisions. On the other hand, the more the map increases in size, the larger is leaf node's voxel, making the map less and less accurate.

- **Defining a leaf node size:** In this reverse strategy, first the size of the leaf node is defined, and then, using the total size of the map, it calculates both the subdivisions number and size of the root node. This operation is described by eq. (3.42).

$$D_p = \log_2 \frac{M_s}{C_s} \qquad\qquad N_s = C_s.2^{R_p} \mid R_p = \lceil D_p \rceil \tag{3.42}$$

  Where C_s is the desired leaf node size, M_s is the map size, N_s is the resulting root node size, and R_p is the number of subdivisions to achieve the desired precision, being the rounding up integer (operator *ceil*) of D_p.

  This mathematical operation ensures that increasing the map does not cause a decrease in the map's accuracy. In fact, the precision will always remain the same (C_s), and the parameters of root node size and number of subdivisions will vary in order to generate the desired leaf node size.

The memory structure of an octree is divided into its node types. The rough memory model can be summarized in Table 3.5 and the complete model is presented in appendix A.4. The memory model for octree was based on (GEIER, 2014) and (ZHOU *et al.*, 2018).

TABLE 3.5 – Summary of rough memory model for octree

	Bytes
Root node	84
Internal node	64
Leaf node	3

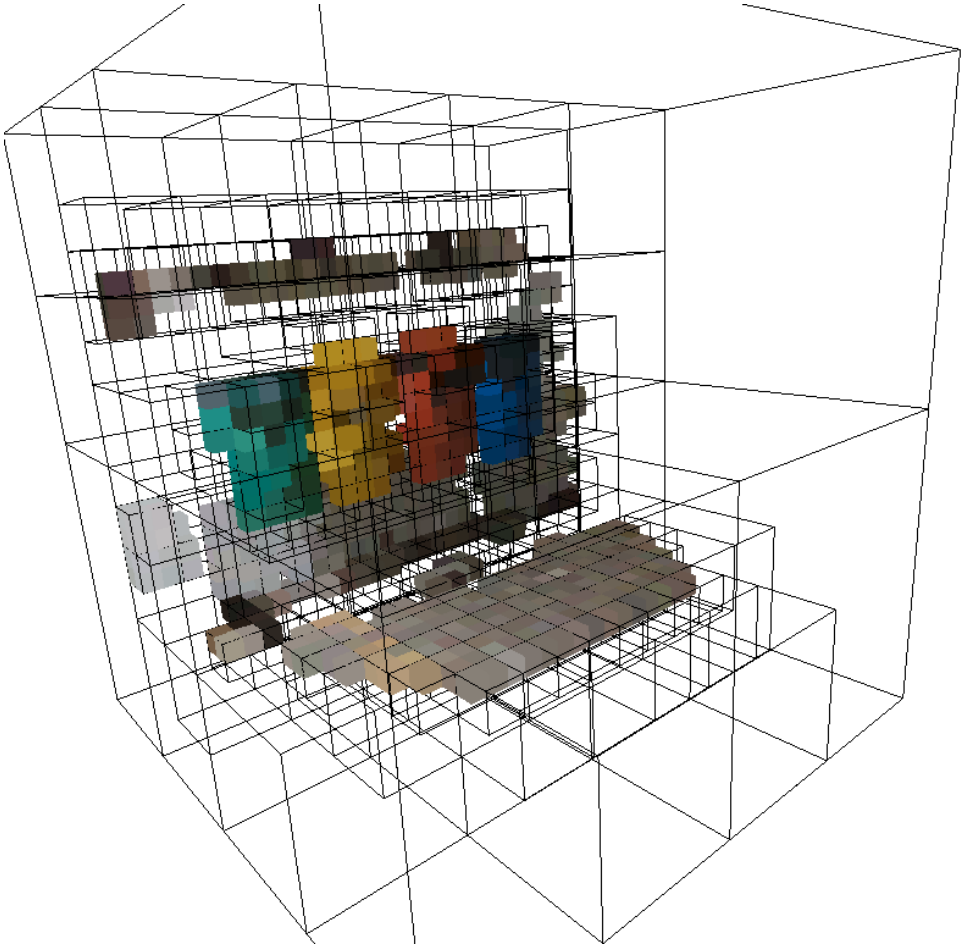


FIGURE 3.20 – Example of an octree scene.

3.6 Integration

Now that each part of the work is explained properly, we can clarify the connection between them. This section details how the previous topics were computationally implemented, focusing on the interconnection and integration.

3.6.1 Execution order

Figure 3.21 shows a flowchart of the execution order of each part of the work. Notice that the exploration is not in the scope of this study, thus the practical implementation and data acquisition can be separated from the main SLAM algorithm. The blocks *Robot's movement* and *Read RGBD image* are readings from odometry and image of stored values of a dataset. If an exploration algorithm were implemented, these blocks can be replaced by on-time measurements of the robot.

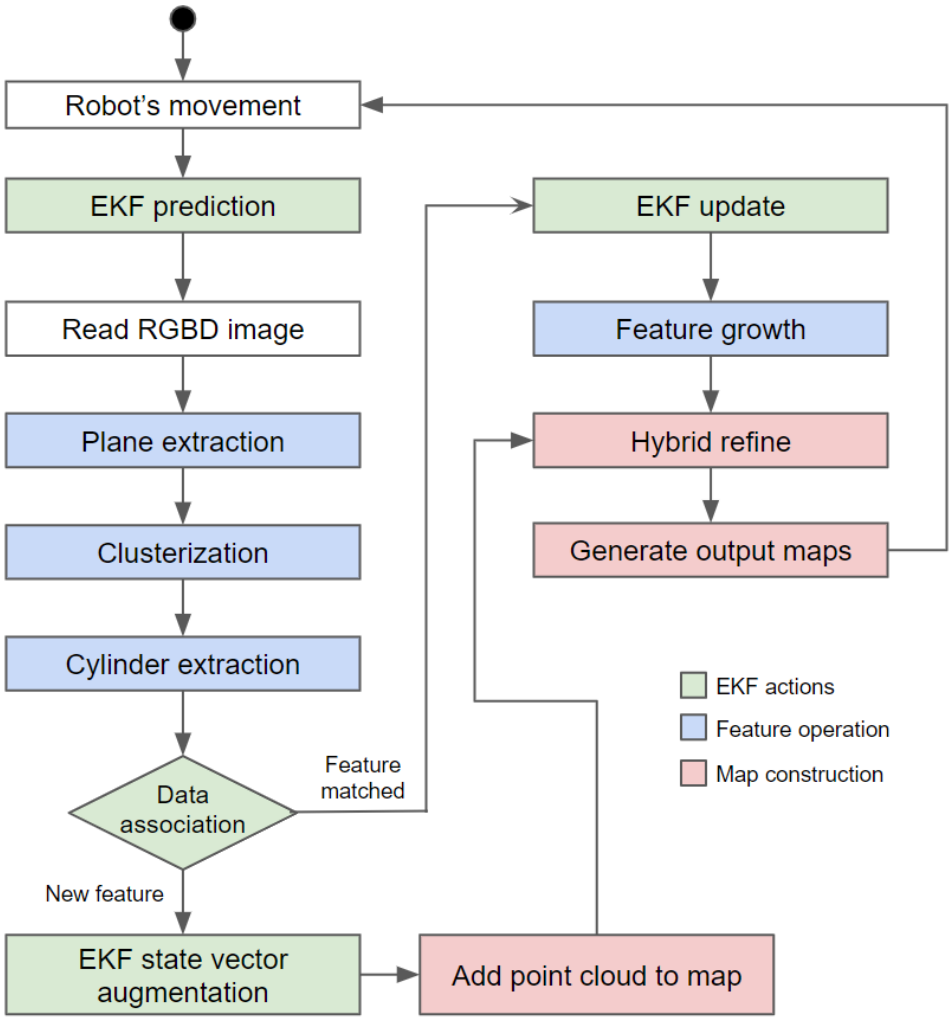


FIGURE 3.21 – Flow chart of the execution order.

3.6.2 Code

The SLAM algorithm proposed here is mainly Python. The most time-consuming tasks use a compiled C++ implementation due to speed gains compared to python. Examples of this category are the low level operations with point clouds such as RANSAC, DB-SCAN, filters, and downsample. For instance, a python planar RANSAC implementation was around ten times slower than the C++ . Other examples of C++ uses are the construction of octrees, voxel grid, point cloud, and 3D display on the screen.

The main code is written in Python and works as a "glue-code" to call the above functions by C++ wrappers. The python implementation also includes the full EKF framework, point cloud transformations, parameterizations, and general linear algebra calculations.

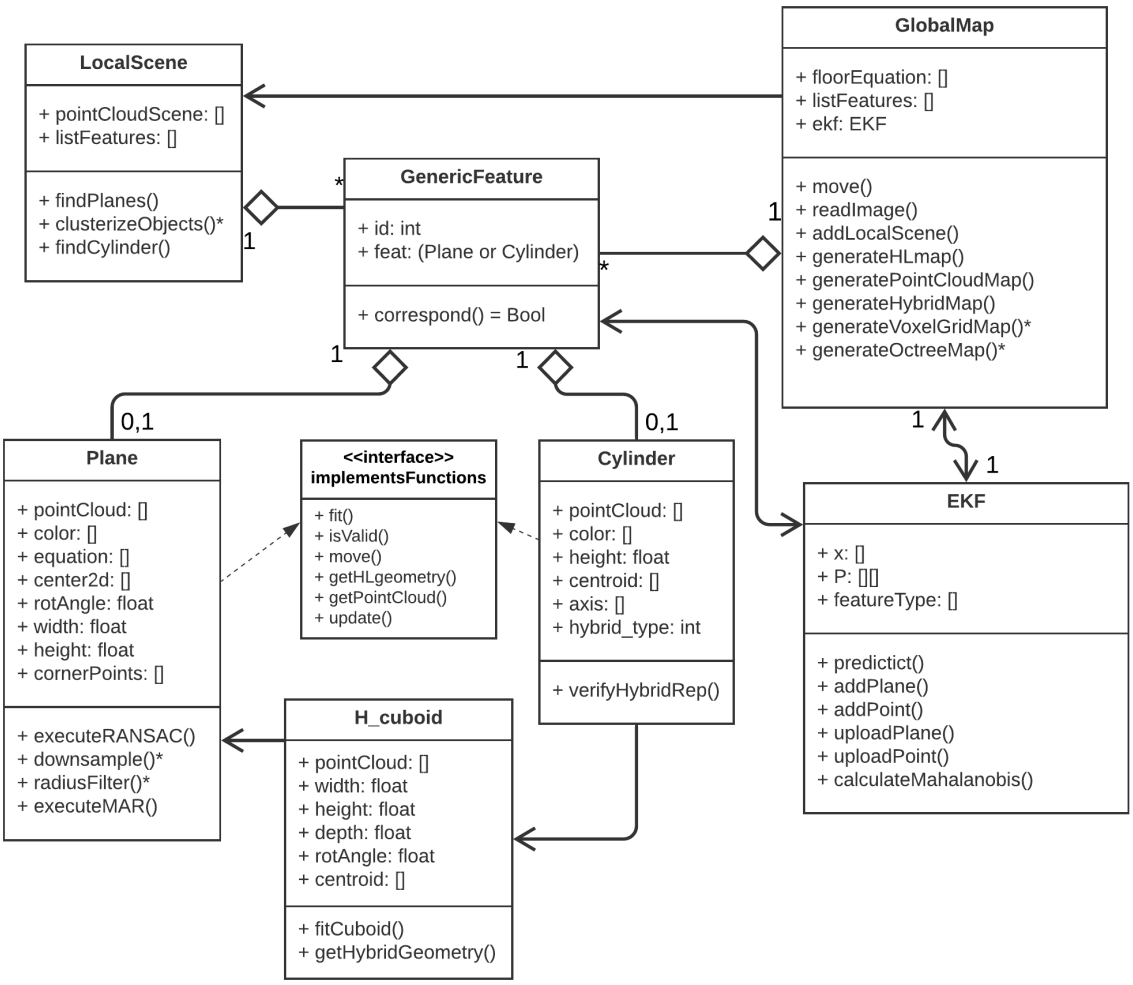


FIGURE 3.22 – Class diagram for the SLAM algorithm

The code uses open source libraries, standing out the Open3D (ZHOU *et al.*, 2018) library, which is a collection of tools to handle 3D data transformations and visualizations. We use its data structure for point cloud, octrees, and voxel grid. Also the C++ implementation of radial filtering, uniform downsample DBSCAN, and the point cloud generation from RGBD image. Also, all 3D visualization is generated by this library.

Other libraries include Numpy for general math operations, Matplotlib for plotting graphs, and minor uses of Pandas and OpenCV.

The code is object-oriented and its class diagram is illustrated in Figure 3.22. Notice that it is only a simplification of the real code, and many internal variables and functions are omitted.

The `GlobalMap` is the main class and contains the global map with all features. It calls `LocalScene` to handle most of the feature extraction methods. After obtaining the features from a local scene they are associated and updated with help of an instance of

the `EKF`, a class that contains all functions related to the EKF, and stores the state vector and covariance matrix.

The `Plane` and `Cylinder` are objects that contains methods for its respective transformation, parameterization, and validation. Since it is a python code, the block `ImplementsFunction` is not a formal interface object, but common functions that both objects implements differently. Both Plane and Cylinder are encapsulated by `GenericFeature` which create connections between the EKF state vector and the feature objects, directing the point cloud growth after an EKF update. The `_cuboid` is only called when using the hybrid map and has rules to create a cuboid based on Cylinder's point cloud.

Aside from the mentioned above, the code has a global class called `Settings` that contains a list of the main parameters and settings. Finally, a set of global functions in file `aux.py` implements a collection of linear algebra functions that are used throughout the work. Other classes such as data structures for point cloud, voxel grid, and octree are implemented in the Open3D library and were omitted in the diagram above.

4 Simulated results

This chapter describes the simulated results for the proposed SLAM method. It describes (I) the simulation setup for software configurations, (II) the map creation in the simulated environment, (III) two simulation results using different process noises. The performance is measured by analyzing the path error and feature position error.

4.1 Setup

The selection of a simulation software for this work must attend a set of requirements:

- The simulation engine must be able to create indoor scenes with walls and objects on the ground, similar to the real environment.
- The simulation shall be able to use a robot to move and acquire RGBD images from its surroundings.
- It must supply simulation meta-data, such as the feature's position and real robot's pose, so this value can be used as ground truth for validation.

A popular engine that complies with all requirements above is the Gazebo software (KOENIG; HOWARD, 2004). It uses Robot Operating System (ROS), a middleware for communication between devices, to send and receive signals. A great advantage of using ROS is the possibility to create multiple nodes for communication. Each node talks with other nodes through topics, which are messages transmitted at a pre-determined rate.

It is also possible to install ROS in embedded platforms. This way, switching from simulation to the real robot is an easy task, since we can standardize the topics' names and rates. Creating node is also possible from within the same computer, gazebo creates topics that are accessible via UDP.

Figure 4.1 shows the setup for the simulation developed in this work. A python code starts the ROS communication with Gazebo and enables to send velocity commands through `cmd_vel` topic. The simulated robot in Gazebo reads these commands

and moves in the environment transmitting back the RGB image `color/image_raw` and depth `depth/image_raw`. The python code reads both images and saves them with the command data.

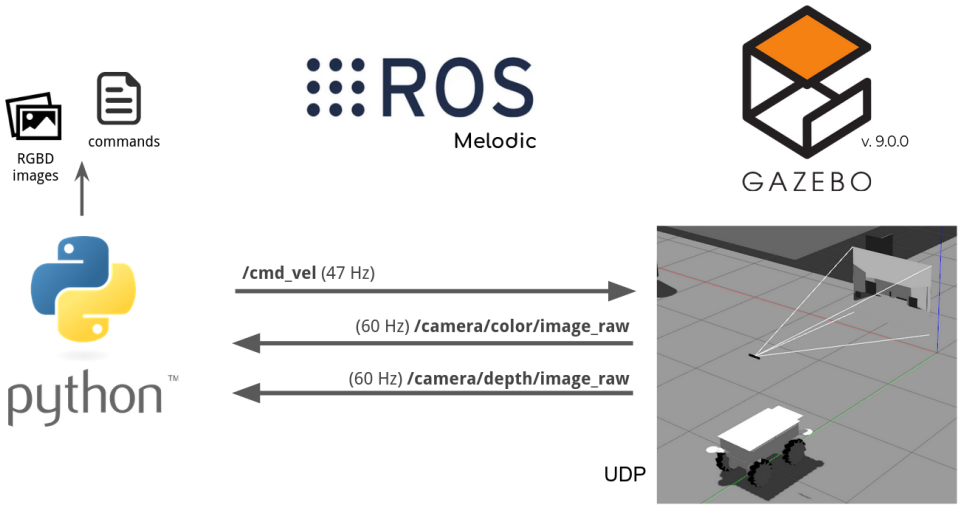


FIGURE 4.1 – Simulation setup

We used the building editor from the Gazebo to create the indoor environment with walls, boxes, and cylinders. The simulation uses a robot model from The Construct (2019) and the Intel gazebo camera plugin from Intel (2013) which already gives the same ROS topic structure as the real camera. Figure 4.2 shows the Gazebo simulation.

The python code creates rotation and translation commands that output a rotation rate and translation velocity for the robot. The code maintains these rates and velocity during a calculated time, based on the desired displacement. The default step size is $0.8m$ and the rotation angle $45°$, nevertheless, smaller displacements were also used for path corrections.

The starting pose is $\hat{x} = [1.440 \quad 0 \quad 0.627]^T$ since this was the pose of the robot in the Gazebo inertial coordinate axis. The covariance matrix starts as zero in all its terms.

This simulation aims to test the capability of the robot in correcting its path based on noisy odometry[1] so we add the process noise $v_k \approx \mathcal{N}(0, V)$ in each step to simulate the displacement error. The measurement noise $w_k \approx \mathcal{N}(0, W)$ only accounts for the error in the feature extraction process, and no noise was added to the point cloud.

[1]The robot does not have an odometry sensor on its wheels. What we call "odometry" is actually a command sent to the robot as a replacement of odometry sensor.

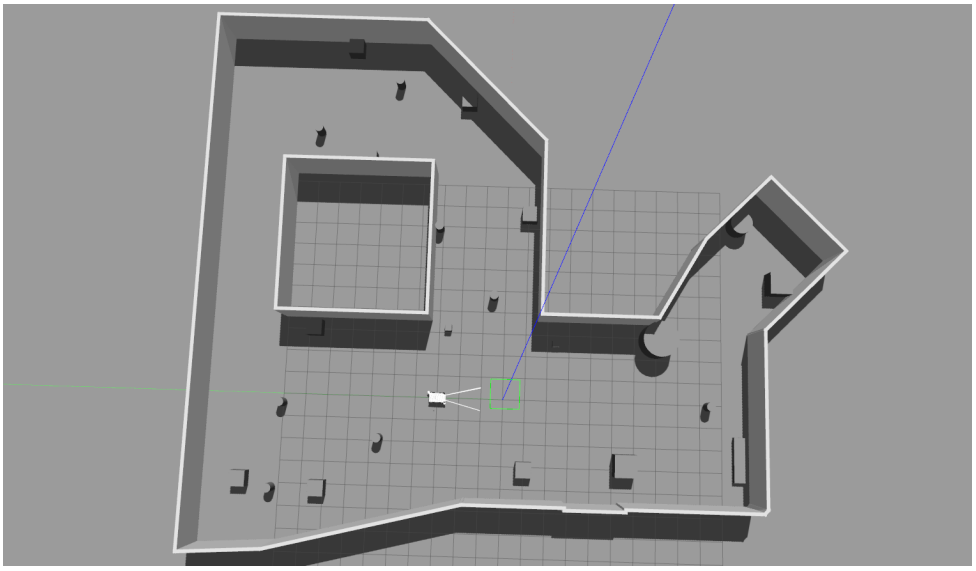


FIGURE 4.2 – Gazebo simulation

4.2 Results

The results are divided into three sections, first, we compare the real map with the zero-noise odometry. Then we test the simulation in two different levels of process noise and discuss the results.

4.2.1 Zero process noise map

The goal of the section is to understand which map would be generated in perfect conditions, based on the current feature extraction method. The zero process noise map is used as ground truth for the next sections and exposes the feature extraction limitation.

Figure 4.3 shows the overlap between two maps. The yellow map is generated by directly appending the extracted point cloud into Gazebo's perfect robot's position, not being parameterized in any way. The blue map is the reconstruction using the SLAM algorithm from this work but without process noise.

The comparison highlights some important points.

- The cuboid marked as **1** wasn't detected because it is partially covered by the shadow of the cylinder next to it, which distorted its shape and was excluded by the filters.
- The number **2** is a shallow box, of a height of 10cm, which is approximately the value for the RANSAC threshold $\boldsymbol{T_r}$. Consequently, many points of that feature were considered part of the floor, or part of the wall next to it. The remaining points were invalidated as a feature.

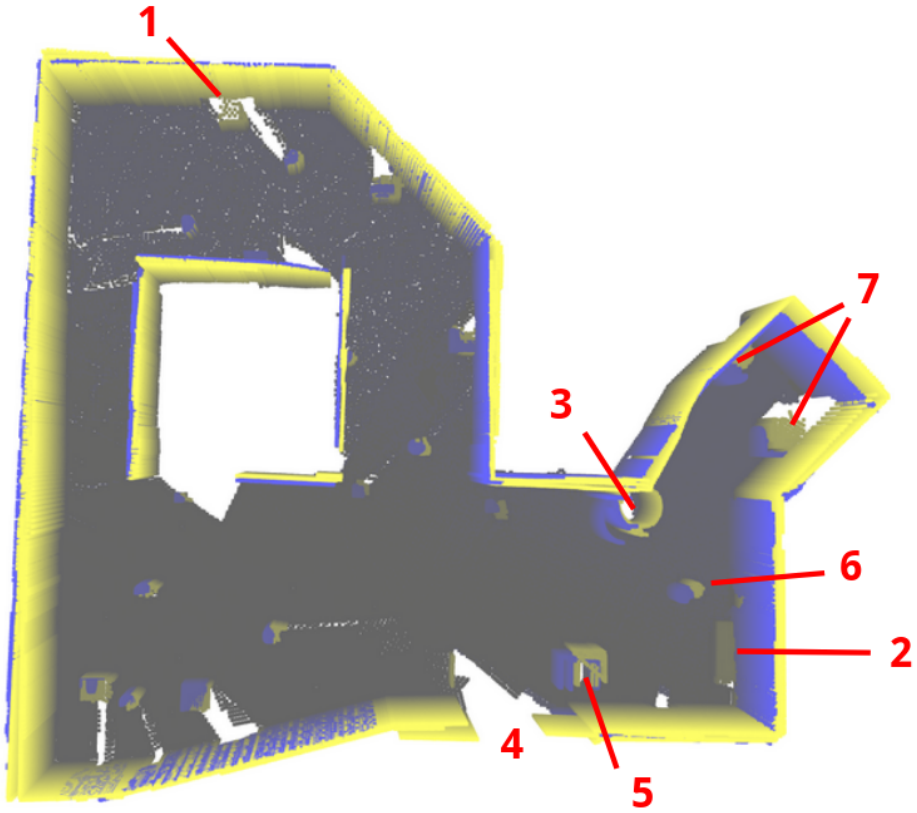


FIGURE 4.3 – Overlap comparison between the zero-noise map (blue) and the real map (yellow).

- The cylinder shell from **3** presented distortions caused by a parameterization of objects partially out of bounds of images.
- The section of the wall marked as **4** wasn't observed even in yellow map.
- Features **5**, **6** and **7** presented small displacements error by the same reason as **3**.

Aside from the points above, the map is consistent to the wall sizes and overall objects position. In the next sections the experiment will be executed adding the process noise to the vehicle odometry.

4.3 Normal process noise ($\sigma_x = 0.067\,m$ and $\sigma_\theta = 1.66°$)

The first test simulates a process noise similar to the real robot. The robot now receives an information of odometry with an addition of a random gaussian noise $v_k \approx \mathcal{N}(0, V)$ which $V = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_\theta^2 \end{bmatrix}$. The estimated process noise is $\hat{V} = V$ and the estimated measurement covariance matrix is $\hat{W} = \begin{bmatrix} \sigma_X^2 & 0 & 0 \\ 0 & \sigma_Y^2 & 0 \\ 0 & 0 & \sigma_Z^2 \end{bmatrix}$ where $\sigma_X = \sigma_Y = \sigma_Z = \frac{0.1}{3}\,m$

which was obtained empirically.

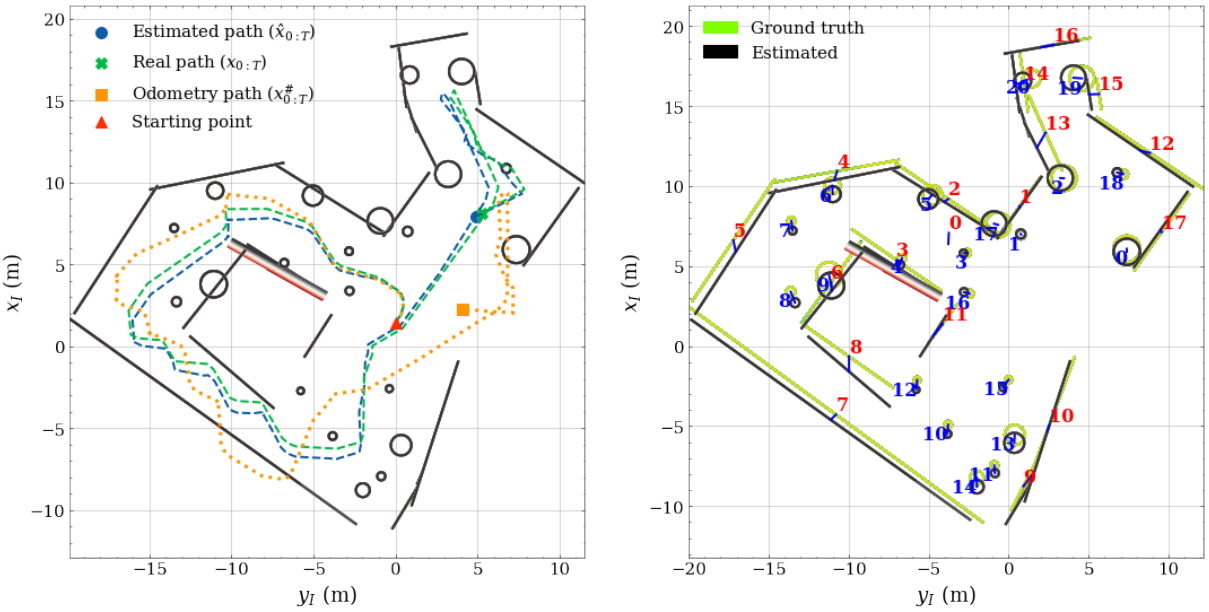


FIGURE 4.4 – 2D projection of the map.

The left image from Figure 4.4 shows the estimated robot path and compares it with odometry path and real path. The right image is the 2D overlap between the ground truth and the estimated map. The blue numbers are the cylindrical features and the red are the walls.

The algorithm successfully preserved the overall shape of the map and its feature's position. Furthermore, the vehicle path presented much less error than the odometry path. Table 4.2 and 4.3 shows the displacement error for each feature in the map.

Figure 4.5 shows the evolution of the path error for the odometry and the estimated path. The odometry path error is the error related to the concatenation of commands sent to the robot and the estimated path error is obtained from comparing the proposed algorithm with the real path. We can see two loop closures happening around step 70 and step 96 which marked the robot revisiting the starting scene. In spite of the odometry error slowly increasing and reaching 7 m, the estimated trajectory error is always within bounds of 1 m. As a quantification tool, we can use the Integral Absolute Error (IAE) criteria (eq. 4.1) to measure the overall error during the whole SLAM process. Table 4.1 shows the IAE criteria for this case.

$$IAE = \int |e| dt \tag{4.1}$$

TABLE 4.1 – IAE criteria for the trajectory

	IAE criteria
Estimated path error	**49.82**
Odometry path error	**304.28**

Figure 4.6 shows the final reconstructed point cloud map, where each feature is colored differently. Some unwanted behavior happened in walls **3** and **9**, which were duplicated. Bigger displacements happened in walls **8**, **14** and **15**. Furthermore, cylinder **17** is actually a corner of two walls but was interpreted as a non-planar object which can be observed in the high-level map in Figure 4.7.

The algorithm also successfully simplified 13 features to its parameterized form in the hybrid map (Figure 4.8), with 7 other features being represented as a point cloud. Notice that it is not expected for all features to be simplified since the robot does not have enough information to be sure of the primitive type of that object. For example, the ground truth only manages to simplify 14 features.

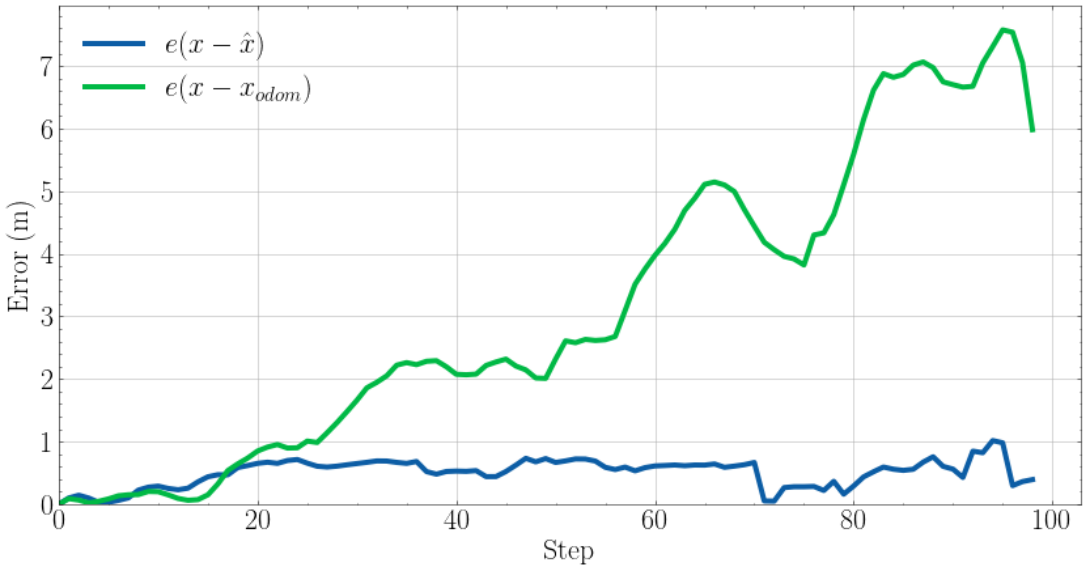


FIGURE 4.5 – Path error for each time step.

Another interesting visualization of the effect of the SLAM correction is making a 3D overlapping. Figure 4.9 shows the overlapping between the odometry-based reconstruction and the output from our SLAM algorithm. Also, the comparison with the ground-truth is illustrated in Figure 4.10.

FIGURE 4.6 – Reconstructed point cloud map where each color represents a feature.

TABLE 4.2 – Cylinder position error.

	c_x	c_y	c_z	$\hat{c}_x$	$\hat{c}_y$	$\hat{c}_z$	Error
0	6.07	7.38	0.59	5.89	7.38	0.59	0.18
1	6.96	0.78	0.91	7.00	0.75	0.92	0.05
2	10.49	3.45	0.16	10.50	3.23	0.15	0.22
3	5.83	-2.61	0.65	5.79	-2.82	0.65	0.22
4	5.51	-6.66	0.65	5.08	-6.78	0.65	0.44
5	9.44	-4.78	0.74	9.19	-5.03	0.75	0.35
6	10.02	-11.02	0.73	9.49	-10.99	0.75	0.53
7	7.82	-13.58	0.65	7.21	-13.50	0.64	0.62
8	3.40	-13.62	0.66	2.71	-13.35	0.66	0.74
9	4.42	-11.22	0.74	3.78	-11.09	0.76	0.65
10	-4.91	-3.79	0.65	-5.49	-3.82	0.65	0.58
11	-7.46	-0.90	0.66	-7.94	-0.86	0.66	0.48
12	-2.12	-5.72	0.73	-2.73	-5.78	0.75	0.61
13	-5.56	0.38	0.77	-6.05	0.35	0.78	0.49
14	-8.24	-1.98	0.78	-8.79	-1.98	0.77	0.55
15	-2.11	-0.01	0.66	-2.60	-0.37	0.66	0.61
16	3.27	-2.44	0.94	3.37	-2.79	0.93	0.37
17	7.57	-0.66	-0.20	7.64	-0.91	-0.17	0.26
18	10.74	7.18	0.65	10.84	6.78	0.65	0.42
19	16.73	4.58	0.57	16.76	4.05	0.57	0.53
20	16.73	1.43	0.59	16.55	0.88	0.59	0.58

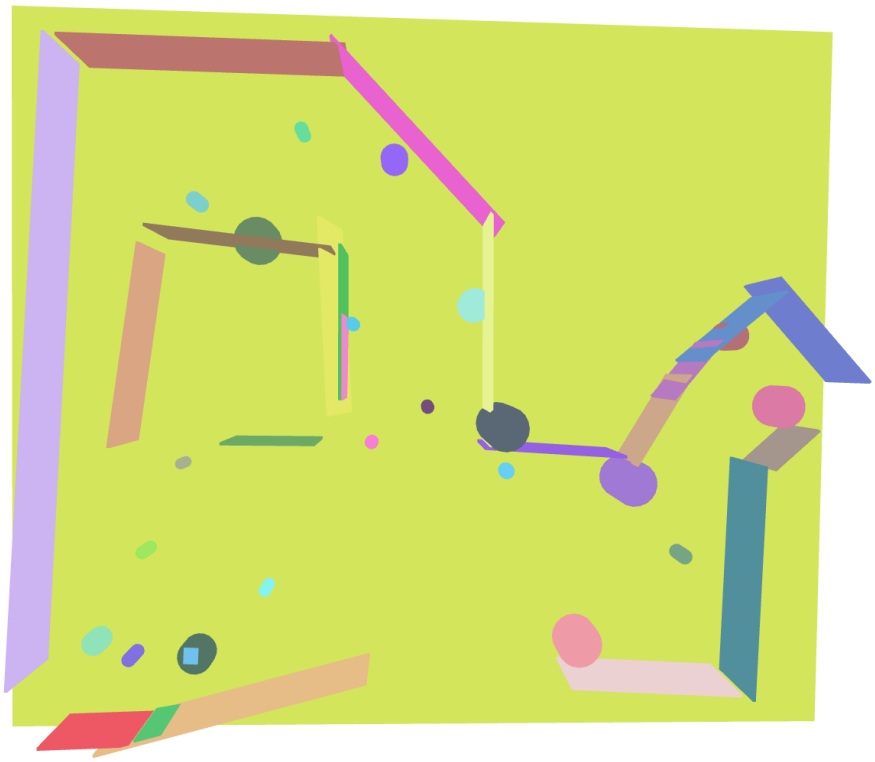


FIGURE 4.7 – Parameterized high level map.

TABLE 4.3 – Plane feature error.

	n_a	n_b	n_c	$\hat{n}_a$	$\hat{n}_b$	$\hat{n}_c$	Error
0	-0.00	0.00	-1.20	-0.00	0.00	-1.20	0.00
1	-2.62	3.67	0.00	-2.63	3.66	0.00	0.02
2	-4.53	-3.08	-0.00	-4.68	-2.92	0.00	0.22
3	-0.37	-0.25	0.00	-0.35	-0.22	0.01	0.03
4	-12.46	2.24	-0.01	-12.07	2.41	-0.01	0.42
5	-10.06	14.87	-0.00	-9.55	14.68	-0.00	0.55
6	-6.69	9.11	0.03	-6.73	8.46	0.02	0.66
7	7.93	5.81	-0.01	8.31	5.96	-0.01	0.40
8	5.13	3.60	0.02	5.85	5.01	0.03	1.58
9	2.19	-4.28	-0.06	1.80	-4.18	-0.00	0.40
10	1.43	-3.85	-0.01	1.21	-3.75	-0.01	0.24
11	0.00	0.00	-0.45	0.00	0.00	-0.45	0.00
12	-2.47	3.32	-0.00	-2.33	3.63	-0.00	0.34
13	-12.41	-8.32	-0.04	-12.16	-8.31	-0.03	0.25
14	-3.08	-6.89	0.03	-3.20	-6.37	0.04	0.53
15	-0.47	-3.28	0.00	-0.33	-2.54	0.00	0.75
16	-1.30	-8.00	-0.01	-1.25	-7.46	-0.01	0.54
17	-17.69	3.29	-0.02	-17.78	3.08	-0.03	0.23
18	2.11	-2.95	-0.00	2.04	-2.86	-0.00	0.11

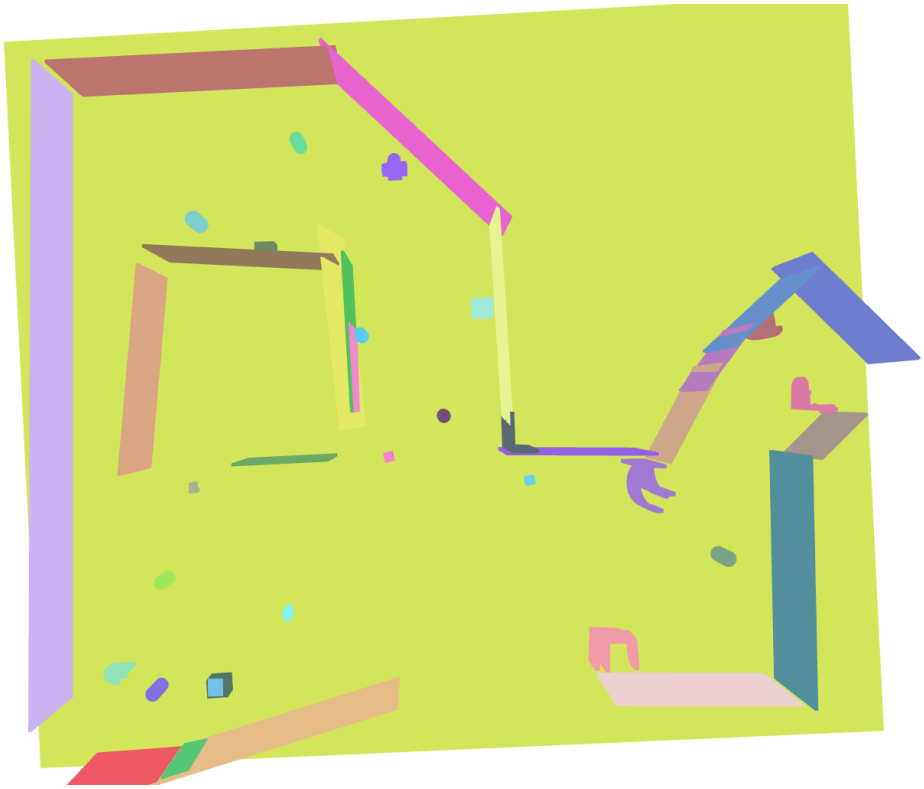


FIGURE 4.8 – Hybrid map.

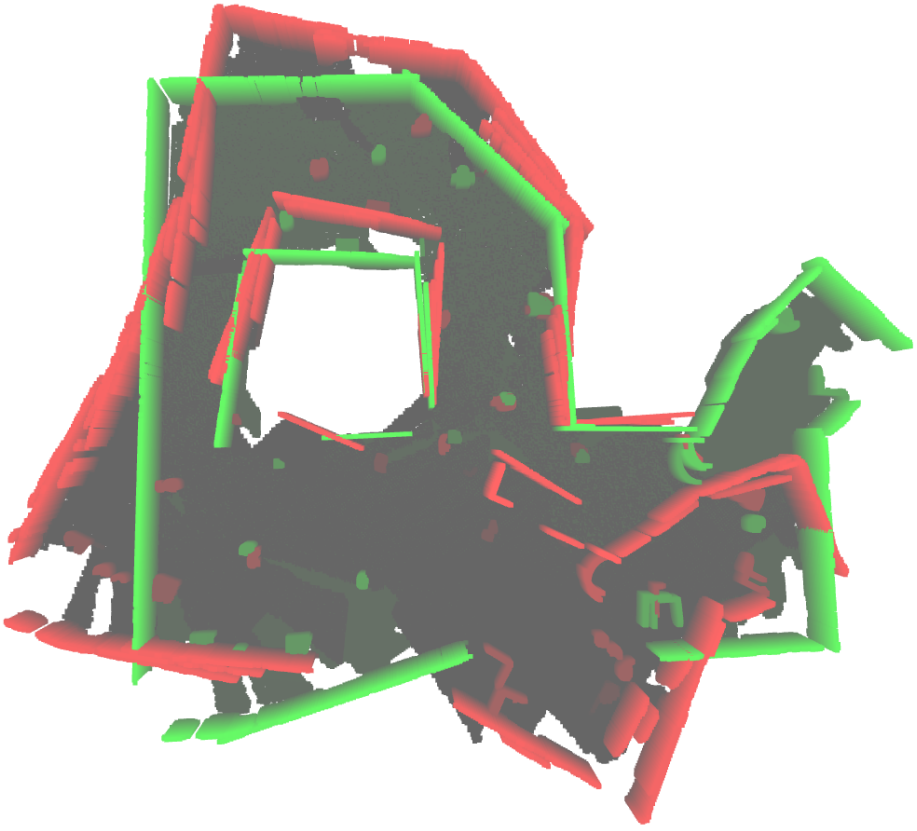


FIGURE 4.9 – The odometry-based map is illustrated in red. The green map is our SLAM estimation.

FIGURE 4.10 – The blue map is the ground-truth and the green is our map reconstruction.

4.4 Increased process noise ($\sigma_x = 0.167m$ and $\sigma_\theta = 3.333°$)

Now we increase the process noise to $\sigma_x = 0.167$ and $\sigma_\theta = 3.333°$ simulating an extreme case to test the SLAM capabilities. Just as before, Figure 4.11 shows that the map was built successfully.

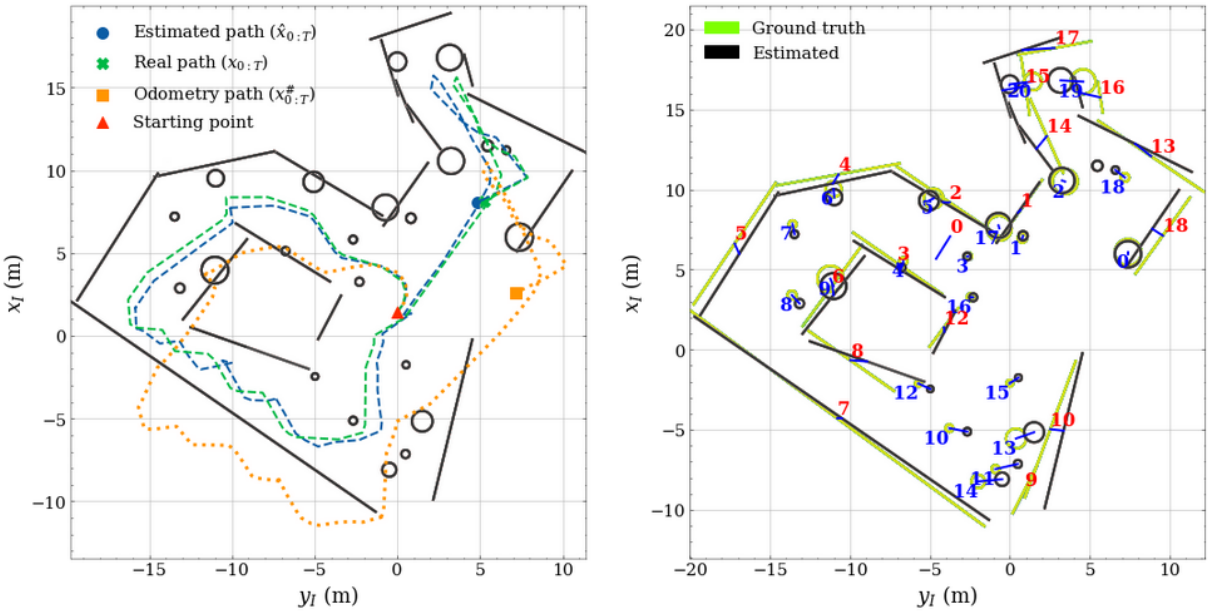


FIGURE 4.11 – 2D projection of the map.

The robot always is looking ahead. Consequently, when turning in corners, the next captured frame does not have a reference from behind nor from previously seen features, which leaves it vulnerable to the process noise. This is the main cause for the wrong angle of plane **8** and is a contributing factor for the error in the corridor on the top right corner of the map.

Another challenge for the algorithm is to differentiate the slight changes in the wall's direction when being affected by noise. An example of this vulnerability is the union of walls **10** and **9** at the bottom of the map. Another problem is the creation of intermediary walls in their intersections. This happens because RANSAC threshold not being able to differentiate these changes and considers part of both walls as inliers. We can see this problem in a third wall created in the intersection of walls **14** and **15** on the top right corner. To handle this problem, parameters $\boldsymbol{T_r}$ and $\boldsymbol{D_{M\,max}}$ must be tuned accordingly.

Figure 4.12 shows the evolution of the path error. The odometry error rapidly increases to around 6 m, the estimated trajectory is always lower than 2 m. The IAE criteria from this case are presented in table 4.4. In both simulation, the ratio between the estimate IAE and the odometry IAE maintained practically the same, about 15%.

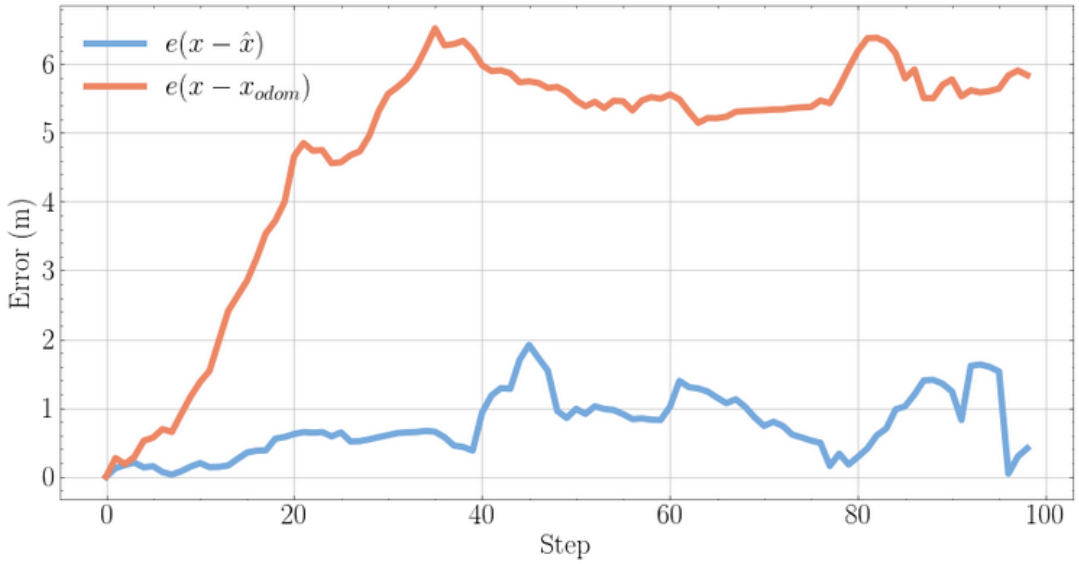


FIGURE 4.12 – Path error for each time step.

	IAE criteria
Estimated path error	**73.5**
Odometry path error	**472.3**

TABLE 4.4 – IAE criteria for the trajectory

Figure 4.13 shows the point cloud reconstruction and Figure 4.14 the high level parameterized features. On this simulation, 15 features were parameterized in the hybrid map, one more than the ground truth. The only reason for that is the duplicated cylinder number **18**.

Figure 4.16 shows the overlapping between the map estimation and the odometry-based reconstruction, and Figure 4.17 the comparison with the ground-truth.

FIGURE 4.13 – Reconstructed point cloud map where each color represents a feature.

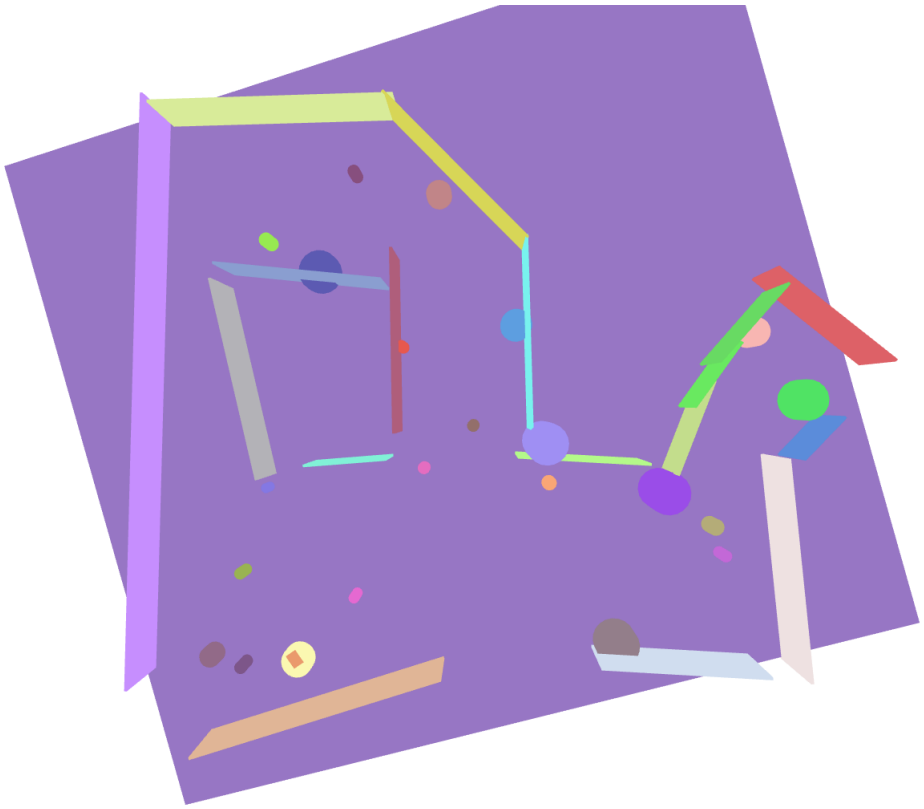


FIGURE 4.14 – Parameterized high level map.

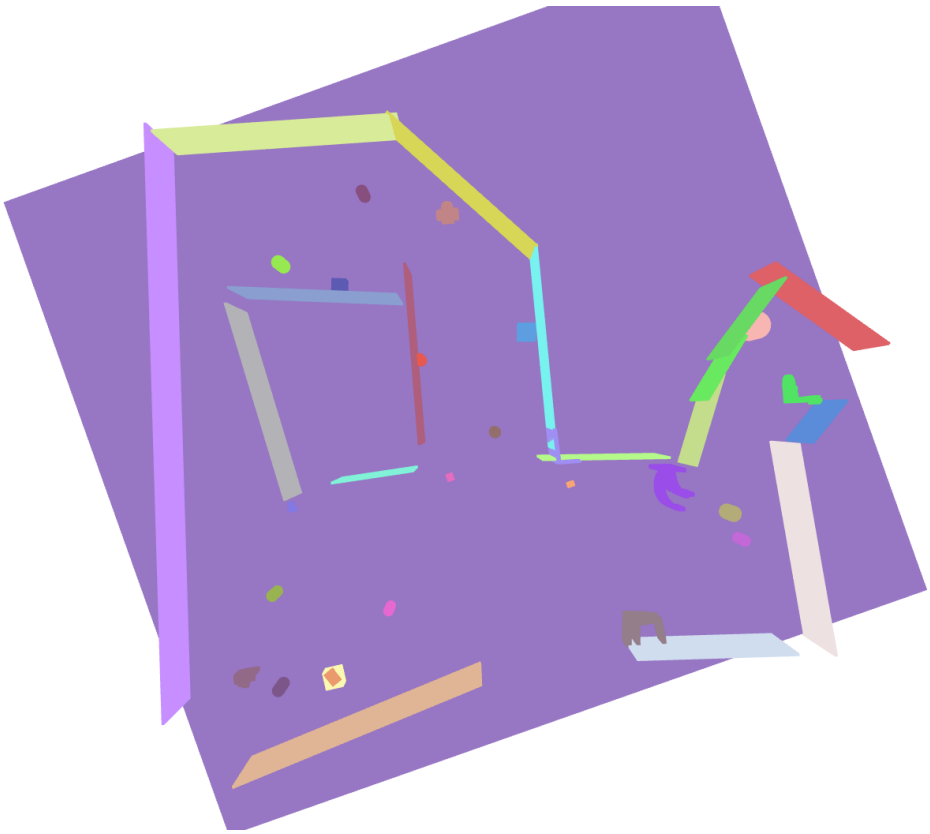


FIGURE 4.15 – Hybrid map.

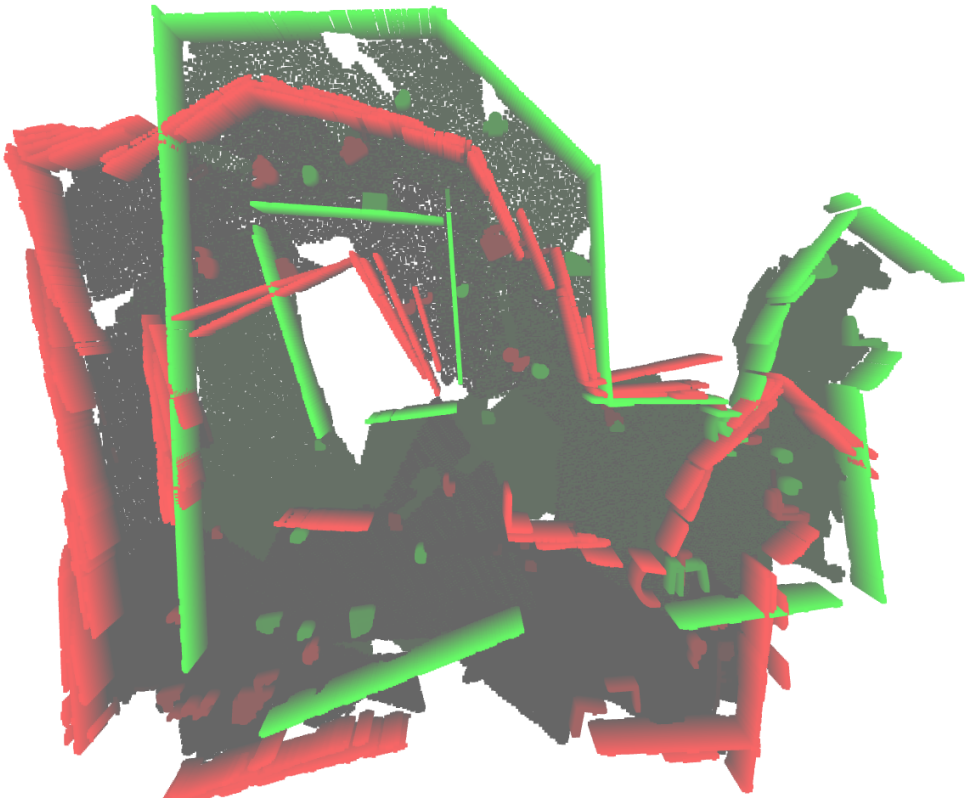


FIGURE 4.16 – The odometry-based map is illustrated in red. The green map is our SLAM estimation.

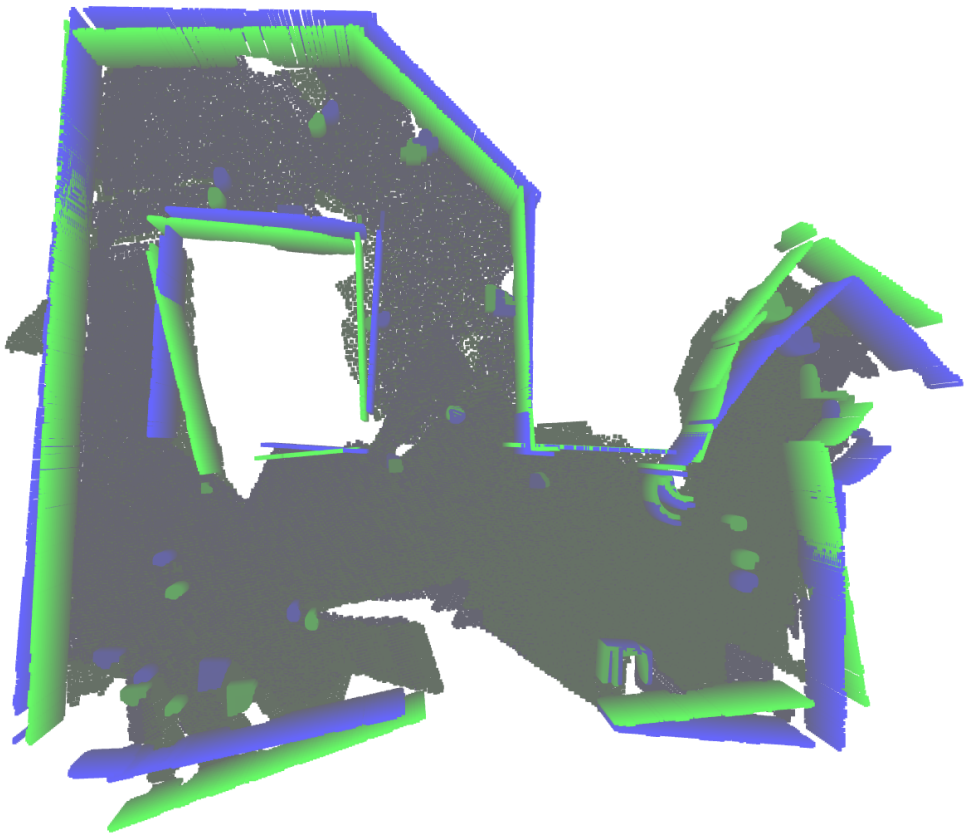


FIGURE 4.17 – The blue map is the ground-truth and the green is our map reconstruction.

TABLE 4.5 – Cylinder position error.

	c_x	c_y	c_z	$\hat{c}_x$	$\hat{c}_y$	$\hat{c}_z$	Error
0	6.07	7.38	0.59	5.95	7.38	0.58	0.12
1	6.96	0.78	0.91	7.10	0.82	0.92	0.15
2	10.49	3.45	0.16	10.56	3.25	0.14	0.21
3	5.83	-2.61	0.65	5.81	-2.67	0.65	0.06
4	5.51	-6.66	0.65	5.12	-6.76	0.65	0.41
5	9.44	-4.78	0.74	9.30	-5.06	0.75	0.32
6	10.02	-11.02	0.73	9.51	-10.98	0.74	0.51
7	7.82	-13.58	0.65	7.20	-13.46	0.65	0.63
8	3.40	-13.62	0.66	2.88	-13.14	0.66	0.71
9	4.42	-11.22	0.74	3.96	-11.03	0.76	0.50
10	-4.91	-3.79	0.65	-5.12	-2.67	0.65	1.14
11	-7.46	-0.90	0.66	-7.14	0.50	0.66	1.43
12	-2.12	-5.72	0.73	-2.46	-4.98	0.73	0.82
13	-5.56	0.38	0.77	-5.16	1.51	0.77	1.20
14	-8.24	-1.98	0.78	-8.10	-0.49	0.78	1.50
15	-2.11	-0.01	0.66	-1.75	0.53	0.66	0.64
16	3.27	-2.44	0.94	3.26	-2.29	0.93	0.15
17	7.57	-0.66	-0.20	7.75	-0.72	-0.19	0.19
18	10.74	7.18	0.65	11.21	6.59	0.64	0.75
19	16.73	4.58	0.57	16.80	3.17	0.57	1.41
20	16.73	1.43	0.59	16.56	-0.00	0.58	1.44

TABLE 4.6 – Plane feature error.

	n_a	n_b	n_c	$\hat{n}_a$	$\hat{n}_b$	$\hat{n}_c$	Error
0	-0.00	0.00	-1.20	-0.00	0.00	-1.19	0.00
1	-2.62	3.67	0.00	-2.62	3.70	-0.00	0.02
2	-4.53	-3.08	-0.00	-4.97	-2.95	-0.00	0.46
3	-0.37	-0.25	0.00	-0.56	-0.34	0.00	0.21
4	-12.46	2.24	-0.01	-12.17	2.57	-0.00	0.44
5	-10.06	14.87	-0.00	-9.38	14.74	-0.00	0.69
6	-6.69	9.11	0.03	-6.71	8.38	0.03	0.73
7	7.93	5.81	-0.01	7.82	5.39	-0.00	0.43
8	5.13	3.60	0.02	3.46	1.22	0.01	2.91
9	2.19	-4.28	-0.06	–	–	–	–
10	1.43	-3.85	-0.01	1.06	-4.33	-0.01	0.61
11	-2.47	3.32	-0.00	-1.93	3.66	-0.00	0.64
12	-12.41	-8.32	-0.04	-13.43	-6.64	-0.04	1.97
13	-3.08	-6.89	0.03	-5.35	-7.08	0.04	2.27
14	-0.47	-3.28	0.00	-1.24	-4.08	0.00	1.11
15	-1.30	-8.00	-0.01	-2.16	-8.03	-0.01	0.86
16	-17.69	3.29	-0.02	-16.65	5.46	-0.03	2.41
17	2.11	-2.95	-0.00	1.74	-2.54	-0.00	0.55

4.5 Memory and time analysis

4.5.1 Hybrid map performance

Figure 4.18 exemplifies the construction of a hybrid map in terms of memory and processing time. In step 5, the robot sees the partial view of the cuboid (see Figure 3.18). However, it cannot interpret what kind of primitive shape it is, so it carries and grows the point cloud up to step 17. At this point, there is a peak in the processing time of almost 1 second, caused by the algorithm parameterizing the cuboid. In the following steps, memory usage drops to levels equal to high-level features.

It is important to mention that the time shown in this graph is only referent to the process of building the hybrid map and not the entire algorithm. As the algorithm does not need to process the point cloud, the time spent processing the map drops to a plateau smaller than it was before the parameterization.

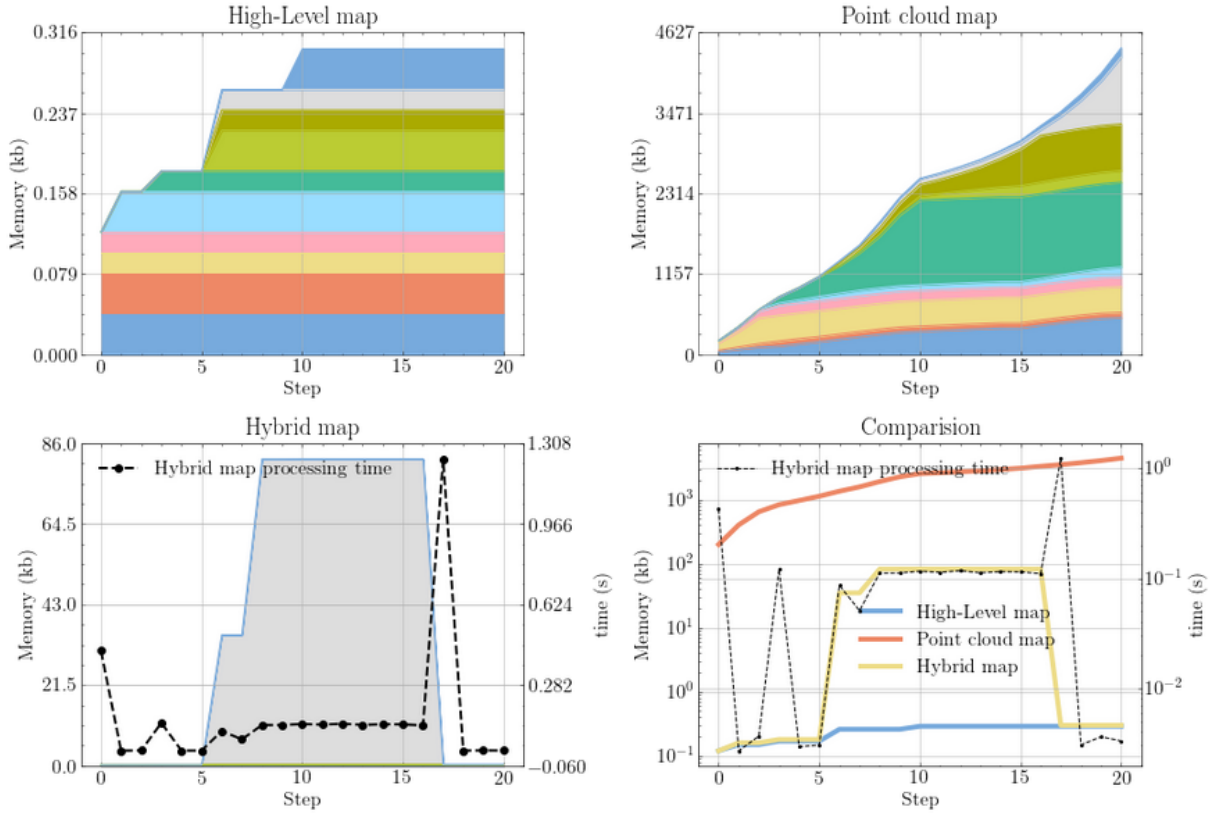


FIGURE 4.18 – A comparison of memory allocated for each feature on different types of maps when creating Figure 3.18. The hybrid map processes the point cloud and parameterizes the feature, reducing it to the same level as the high-level map. The colored layers are the memory usage of each feature.

4.5.2 Voxel grid and Octree

Until now, we only presented the high-level, point cloud, and hybrid maps. However, voxel-based maps are also options to decrease memory usage. Figure 4.19 shows examples of the simulated environment as octree and voxel grid.

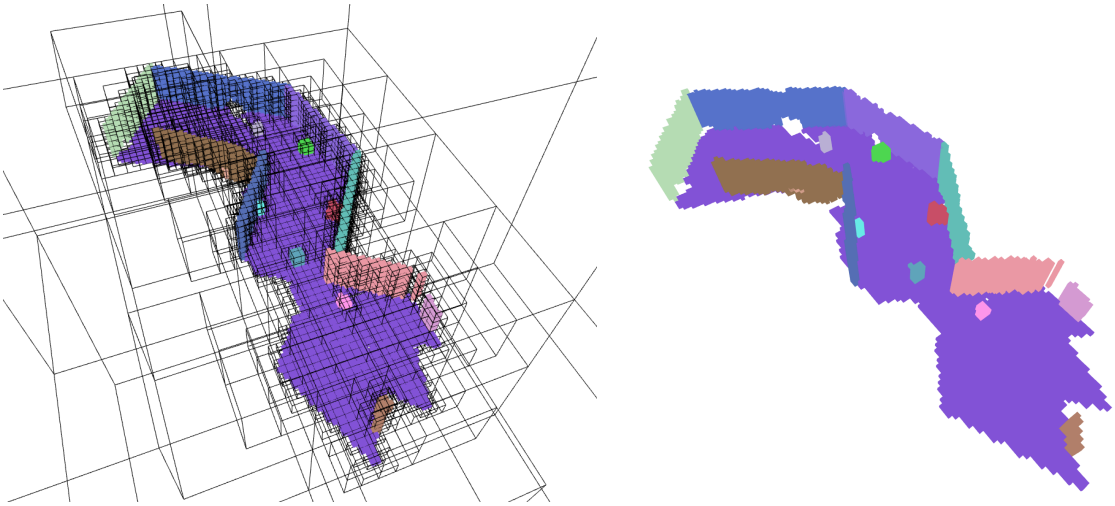


FIGURE 4.19 – Partial map from section 4.4 built in octree (left) and voxel grid (right) with a voxel size of 20cm

Figure 4.20 compares the two voxel-based strategies. We see that the memory usage of an octree map is about 60% of the voxel grid. However, when looking at the time taken to create the map, it is clear that the voxel grid is much faster. While the octree needs to be created in a tree-like structure, the voxel grid is created by inserting points in a hash table, of approximate linear-complexity points of the cloud.

Notice that the processing time in this analysis represents the time to transform the whole point cloud into a global voxel-based map. Thus, these maps are dependent on the point cloud, different from the hybrid or high-level map. For independent octree algorithm, Octomap from Hornung *et al.* (2013) made big advances in implementing grid mapping approaches, providing data structures and mapping algorithms.

4.5.3 Global vs Feature-wise octree - Truncation effect

We can ask the following question: is it better to create an octree representation of each feature individually and place them separately on a continuous map, or create a global octree map without feature distinction?

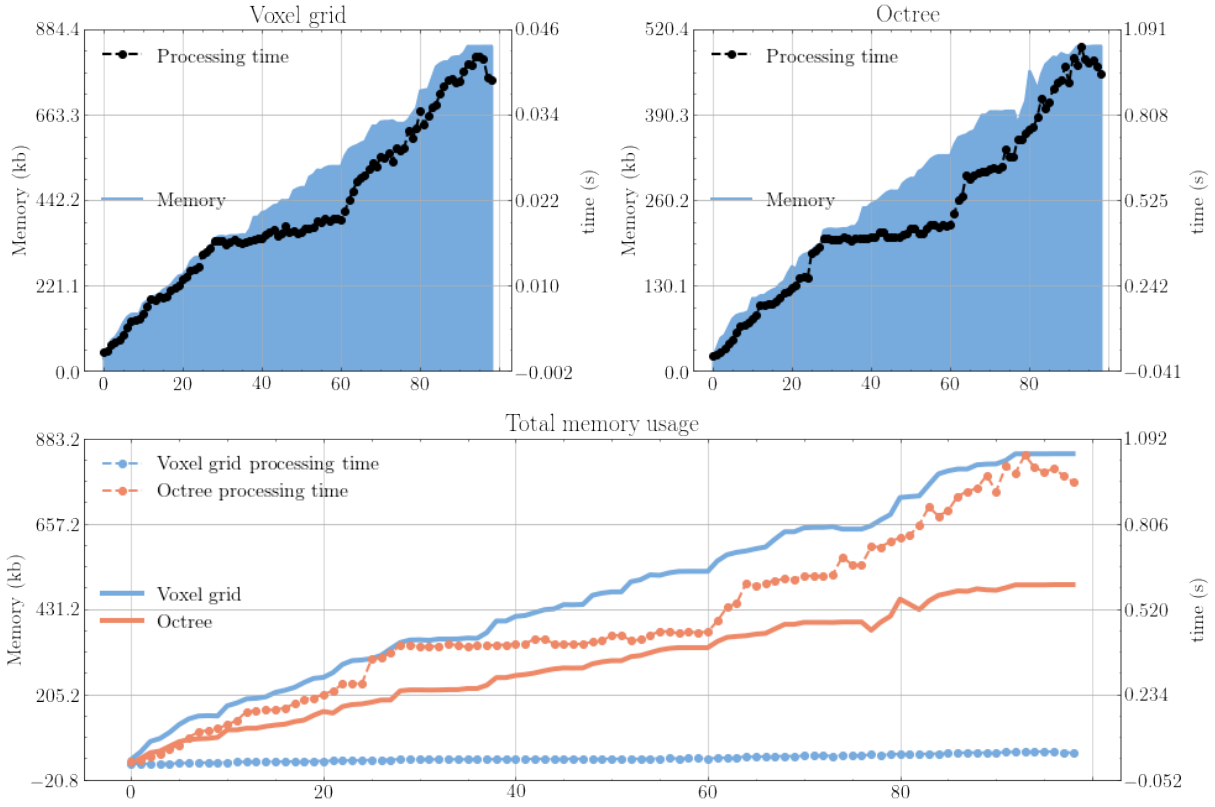


FIGURE 4.20 – Time and memory comparison for creating octree and voxel grid with same precision (20cm of voxel size) from map of section 4.4.

4.5.3.1 Fixed leaf node size

When building an increasingly large octree map, the depth of that tree must be increasingly greater to achieve the desired precision. Thus, comparing the feature-wise and the global octree, we see that the memory usage of both cases is roughly the same. The main difference is in processing time. The feature-wise map is faster to build because the time of traversing many smaller trees is less than traversing a tree with many nodes. Figure 4.21 illustrates this effect.

4.5.3.2 Truncated tree

We can use truncation to limit the memory usage of an octree. Truncating individual features ensures maximum usage of memory for each object, and truncating a global map means limiting the whole map to a certain level of memory usage. However, truncation of maps has undesired effects.

Truncating a global map causes a compromise with precision since the leaf node will increase in size as the map grows (left image from Figure 4.23). In the global octree, adding and growing features does not necessarily cause an increase in memory, only a rearrangement in the tree structure.

Analyzing the feature-wise truncated octrees, smaller features will have much more accurate representations than large features (right image from Figure 4.23). The bigger the number of features in the map, the larger is the memory usage. However, growing each individual feature does not necessarily increase memory usage, but can decrease its precision.

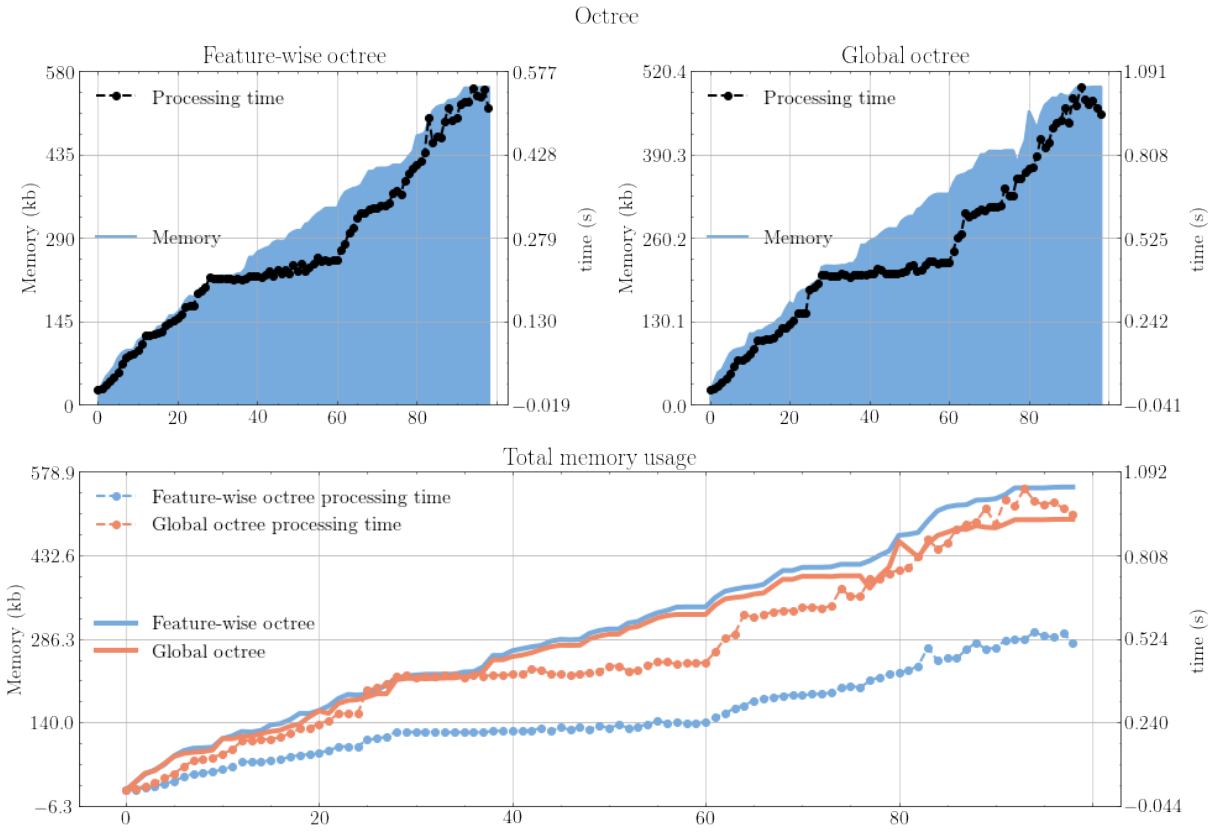


FIGURE 4.21 – Building an octree of fixed-size (20cm) leaf node. Comparison of global map and feature-wise representation

4.5.4 Feature growth and map augmentation

Understanding the dynamic of a map construction over time can give interesting insights in order to choose a method for a determined application. Each one of the five map options presented in this work has a different profile when increasing the number of features or growing the existent ones. By looking at Figure 4.24, it is clear that there are 3 levels of memory usage.

The first one is the high-level map, in which all features are parameterized in planar segments or cylinders. Adding a new feature to the map stacks its size in the memory, but growing it in dimensions does not provoke a memory increase.

The second level is where lies the octree and voxel-grid maps. In this test, we compare the feature-wise fixed leaf node size for an octree. Both leaf node size and grid size of

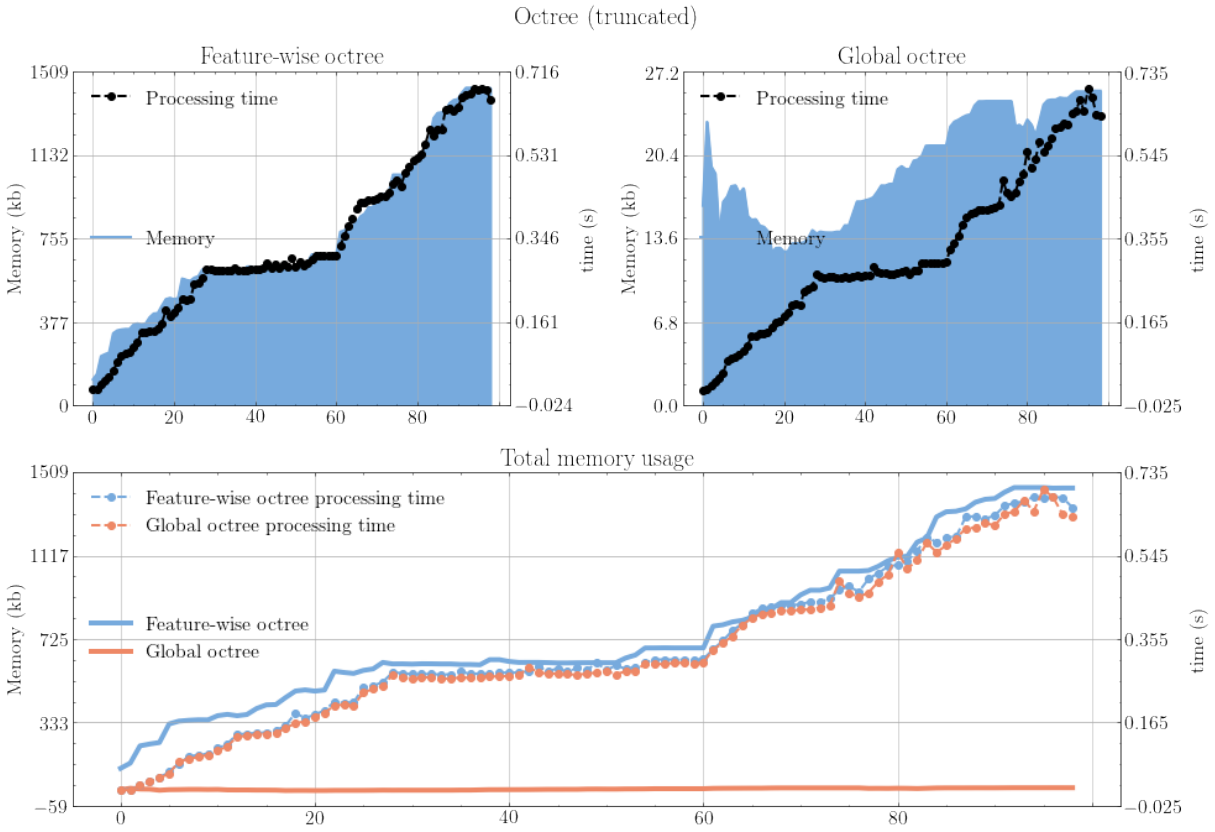


FIGURE 4.22 – Building a truncated octree. Comparison of global map and feature-wise representation

voxel grid are 20cm. The physical growth of the feature causes an increase in its use in memory which is demonstrated by the floor plane (bigger blue area). The growth pattern of octree and voxel grid is similar, but octree is more efficient in this regard.

The last level, the point cloud is least efficient in terms of memory. The growth of the features also causes an increase in memory as a greater amount of points are now describing the feature. Furthermore, multiple observations of the same feature also add points to it, even if there is no increase in its size. Such behavior can be modified with filtering if desired.

Finally, the behavior of the hybrid map moves between levels, always seeking to optimize the scene and minimize the memory usage.

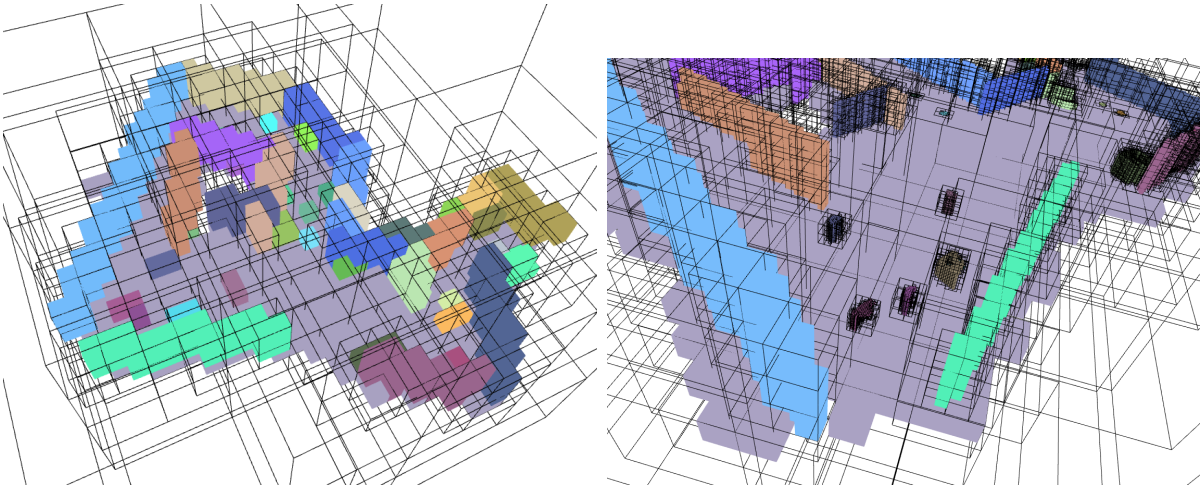


FIGURE 4.23 – Undesired effects of octree truncation. On the left, the global octree truncation causes a loss of precision. On the right image, the feature-wise truncation causes each feature to have different leaf node sizes. In both cases, the octree was truncated in 5 subdivision.

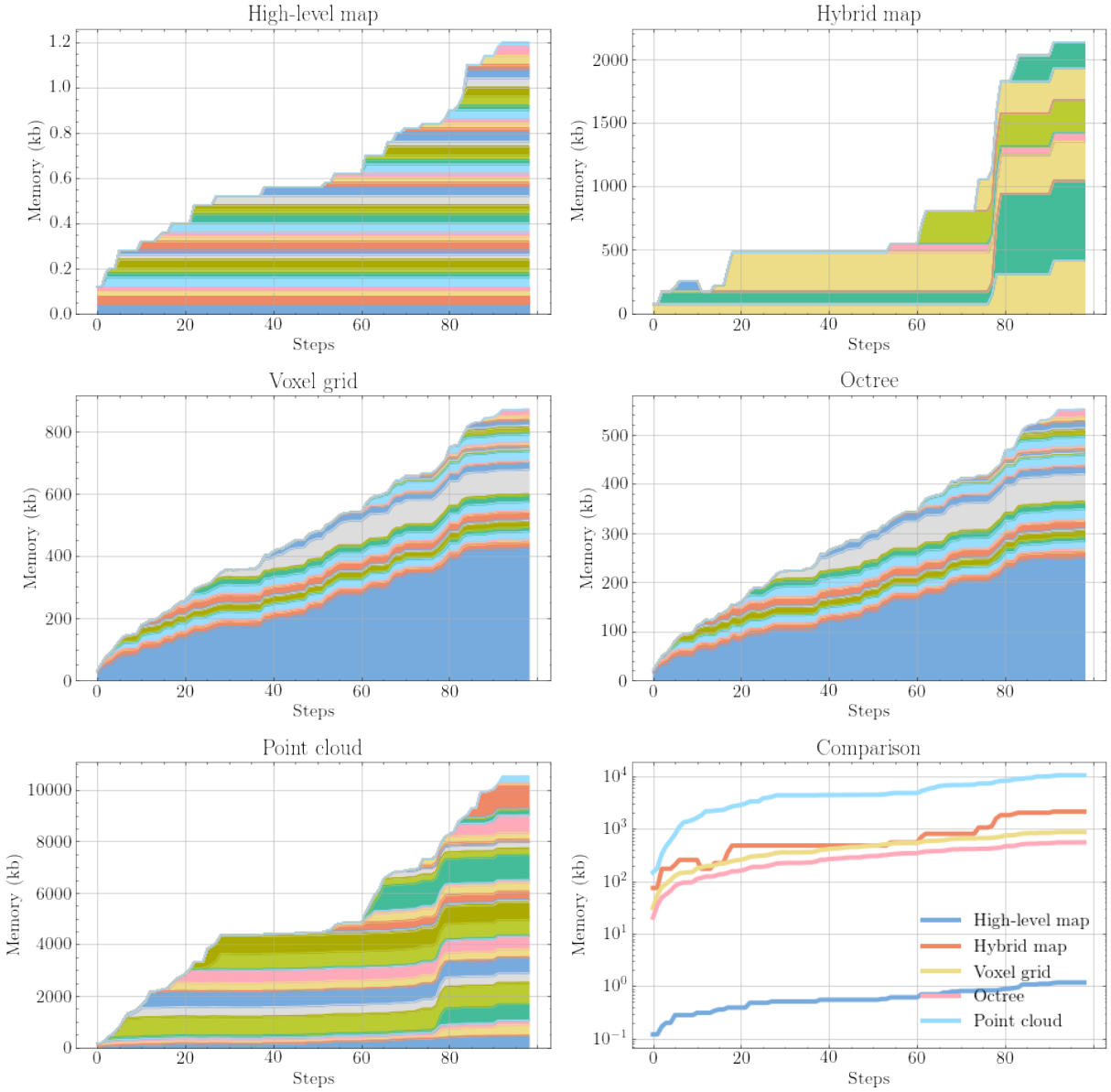


FIGURE 4.24 – Memory usage for each type of map from section 4.4

5 Experimental Results

In this section we discuss the experiments in the real environment. Following the same structure of simulation, we first define the setup for the experiment, describing hardware, software and location. Then, we analyse the results by comparing the reconstructed environment with real measurements from the scene.

5.1 Experimental Setup

For the experiment, we used the Omni robot available in the Intelligent Machines Laboratory (LMI). This robot was built by past laboratory members and was further improved for this work. The robot is described in Figure 5.1. A summary of the components in the robot is described in Table 5.1.

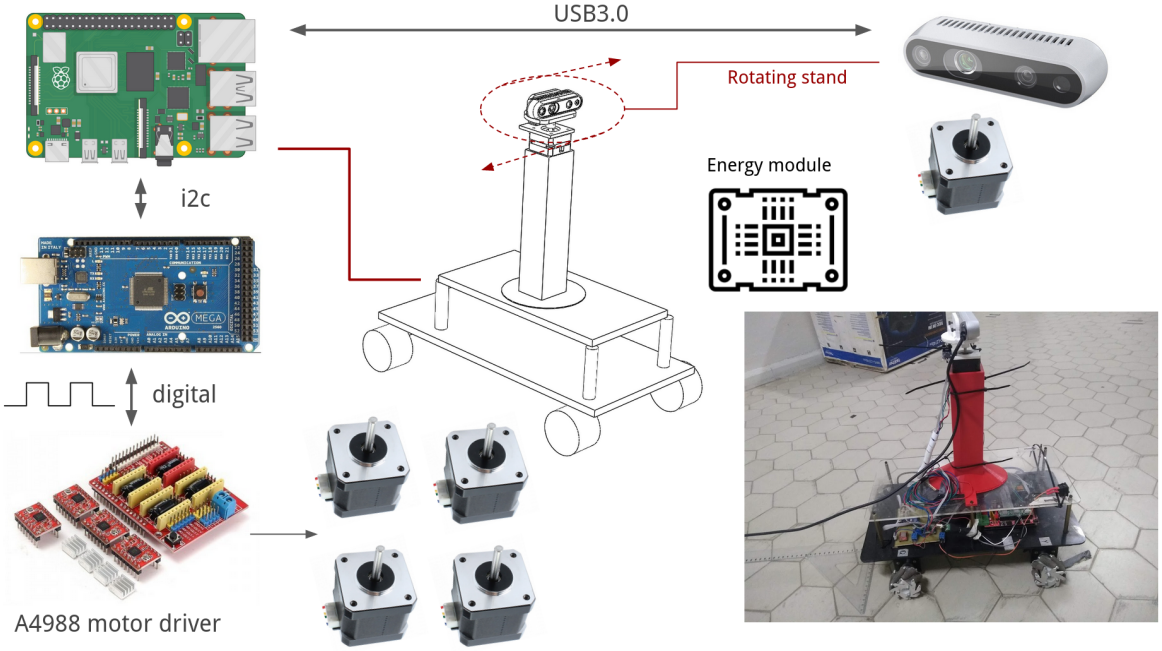


FIGURE 5.1 – Omni robot description.

The Omni robot contains a Raspberry pi as a central computer and connects to an Arduino Mega via i2c connection which is responsible for low level commands to the

TABLE 5.1 – Summary of hardware description of Omni Robot

	Specification
RGBD camera	**Intel Realsense D435i** Depth range: 0.3 to 3 m Depth resolution: up to 1280 x 720 Depth accuracy: $< 2\%$ at 2 m
Processing board	Raspberry Pi 4B Arduino Mega 250
Motor driver	4x A4988 (+1 for camera rotation)
Motor type	4x Stepper motor (+1 for camera rotation)

robot. By sending predetermined commands to the Arduino, the robot moves in different levels of speed and directions through four A4988 motor drivers and four stepper motors connected to it. The rotation logic from each wheel that was implemented inside the Arduino is the classical omnidirectional model (OLIVEIRA *et al.*, 2008) and is abstracted from this work since we only use the unidirectional and rotational movements.

For this work, we added other functionalities: A 3D-printed tower necessary to create height for the photos. We also added another stepper motor to enable the camera to rotate without moving the robot. Finally, we renewed the energy distribution in the robot by designing and manufacturing a printed circuit board with voltage regulators and other general connections which eliminated the old jumper-based connections that caused connection problems. These changes were shared with other students that were also using the robot for their research.

The communication with the robot has the same architecture as the simulation. The only difference is the replacement of the Gazebo node for the Raspberry node as explained in Figure 5.2. The computer used to execute the SLAM algorithm is described in Table 5.2.

TABLE 5.2 – System specifications of the computer used to run the SLAM algorithm. (simulation and real experiment)

	Specification
CPU	Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz x 8
Memory	15,6 GiB
Operating System	Ubuntu 18.04.5 LTS
Gnome	3.28.2
OS type	64-bit

This work uses an RGBD Intel Realsense D435i camera which is a stereo depth camera. It has an open-source and cross-platform SDK with ROS support and many tools available for pre and post-processing depth images.

The movement command consists of a rotation followed by a frontal displacement of

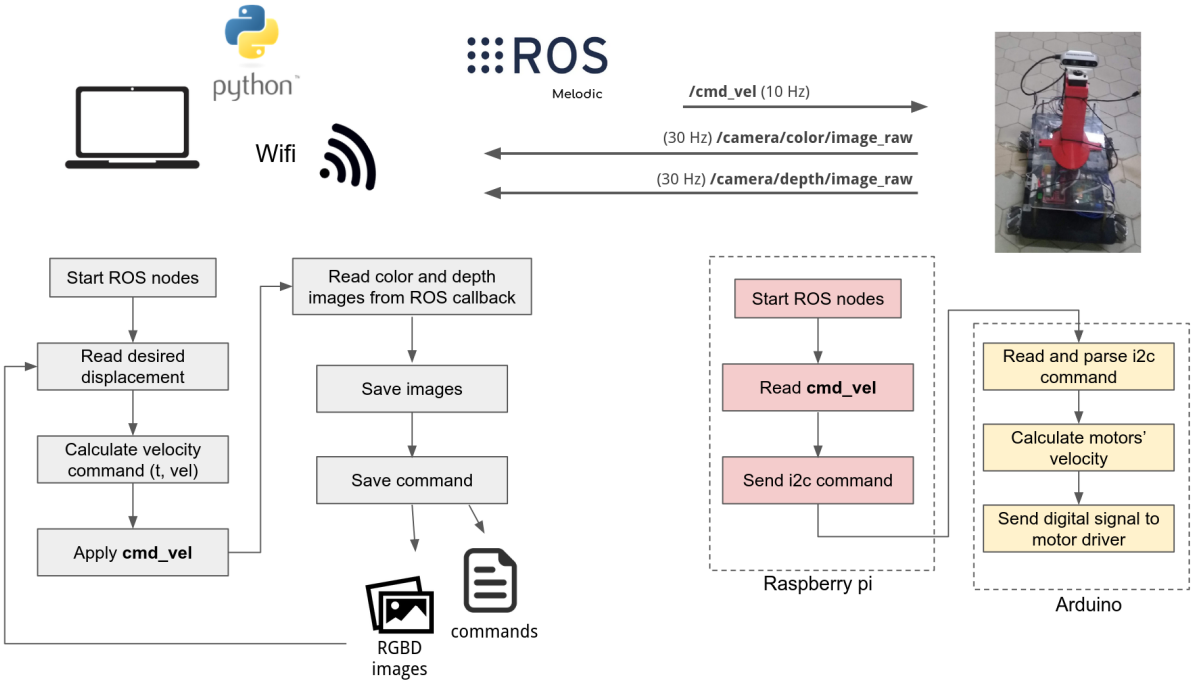


FIGURE 5.2 – Connection used to acquire dataset

the robot's body. The maximum value of rotation is 45° and the maximum value of translation is 0.71 m.

We can also send camera rotation movements to rotate the camera with the robot's body steady to make quicker sequences of 4 photos rotating 90° each.

5.1.1 Process noise estimation

Just as in simulation, in the experiment the robot does not have a odometry sensor. The measurement of displacement is the command sent to the robot by a python code. We initially estimated covariance matrix $\hat{\boldsymbol{V}}$ based on the following experiment: We repeated a set of movement commands, alternating in clockwise and counterclockwise rotation using the maximum step values. The result achieved was $\hat{\sigma}_x = 6$ cm and $\hat{\sigma}_\theta = 1.5°$. After running the experiments some times we changed it to slightly pessimist value: $\hat{\boldsymbol{\sigma}}_{\boldsymbol{x}} = \mathbf{0.0667}\,\boldsymbol{m}$ and $\hat{\boldsymbol{\sigma}}_{\boldsymbol{\theta}} = \mathbf{1.667°}$ A undershoot of movement was corrected by directly increasing the command duration.

5.1.2 Measurement noise estimation

The measurement covariance matrix $\hat{\boldsymbol{W}}$ is based on two components. The first is related to the camera's intrinsic point cloud noise that is generated during the capture of the stereo image. A work from Ahn *et al.* (2019) found that it can be modeled as a

Gaussian distribution based on the distance z from the point to the camera, given the following simplified equation:

$$\sigma_i(z) = 0.001063 + 0.0007278\ z + 0.003949\ z^2 \tag{5.1}$$

Notice that the noise increases with the square of the depth which causes a rapidly deterioration for values above 3m:

$$\sigma_i(3) = 0.03878 \qquad \sigma_i(4) = 0.06715 \tag{5.2}$$

To avoid unreliable measurements, we limited the point cloud to a depth of 3m, which is also a standard recommendation from the manufacturer. This fact restricts the number of features that can be observed during each measurement and put up a big challenge for this work.

The second component is the noise introduced during the feature extraction process and is difficult to be calculated. Therefore, the final value for the measurement noise was obtained empirically during many tests using samples from the camera. This value is the same for each axis of both plane and cylinder feature $\hat{\boldsymbol{\sigma}} = \mathbf{0.0333}$ m.

5.2 Experiment 1 - The LMI corridor

The first test is an indoor environment built using mostly structured objects in a corridor as shown Figure 5.3. The only exception is a chair that was put as a "mapping challenge" to study the robustness of the feature extraction parameters. Then, we planned a desired trajectory in the ground. In the test, we calculate the commands to send to the robot, aiming to correct and follow the desired path as best as possible.

FIGURE 5.3 – The experiment was conducted in the corridor in front of the LMI.

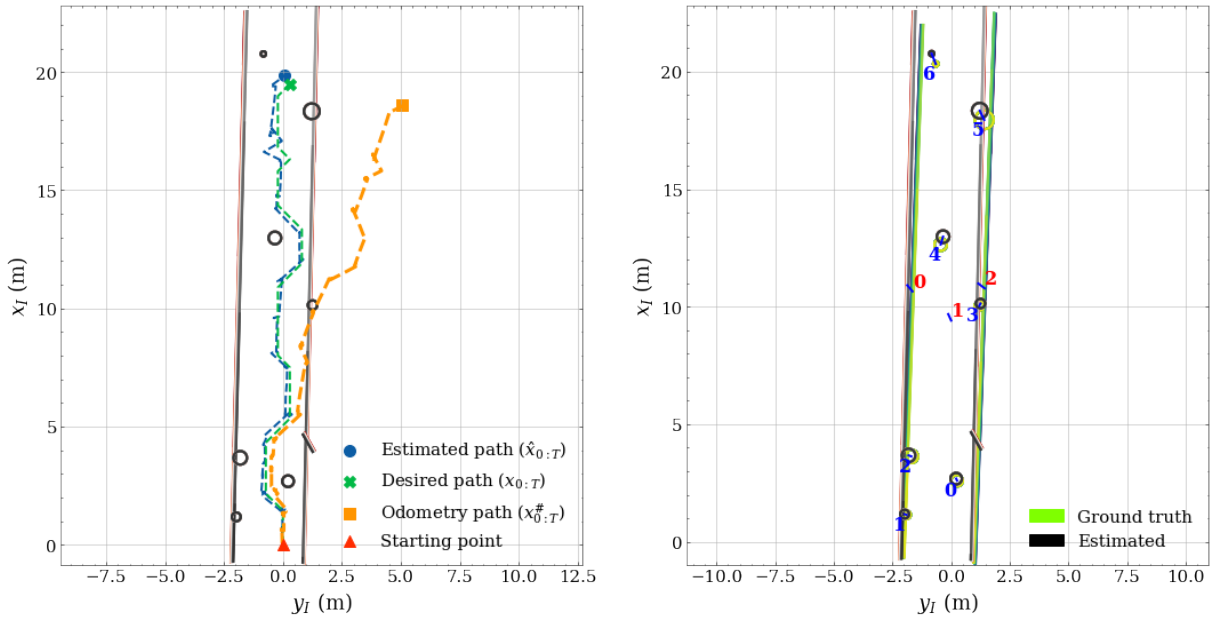


FIGURE 5.4 – 2D projection of the experiment.

Figure 5.4 shows the result for the corridor experiment and Figure 5.8 shows the overlap with the odometry-based map in a 3D point cloud environment.

We see that the algorithm manage to successfully reconstruct the map, even though the first third part of the map presented a higher amount of noise. The maximum error for the cylinder feature was 47 cm and the normal distance error from the wall was 13 cm, mostly due to a slight wrong orientation. The error for each feature is displayed in Tables 5.3 and 5.4. The uncertainty of the robot's pose is illustrated in Figure 5.5.

TABLE 5.3 – Cylinder position error.

	c_x	c_y	c_z	$\hat{c}_x$	$\hat{c}_y$	$\hat{c}_z$	Error
0	2.60	0.26	0.41	2.68	0.22	0.41	0.09
1	1.15	-1.90	0.37	1.18	-1.98	0.37	0.08
2	3.63	-1.69	0.17	3.67	-1.81	0.17	0.13
3	10.04	1.21	-0.00	10.14	1.26	-0.01	0.11
4	12.65	-0.44	0.12	12.98	-0.34	0.12	0.35
5	17.95	1.42	0.27	18.34	1.23	0.28	0.43
6	20.33	-0.65	0.26	20.77	-0.83	0.26	0.47

TABLE 5.4 – Plane feature error.

	n_a	n_b	n_c	$\hat{n}_a$	$\hat{n}_b$	$\hat{n}_c$	error
0	0	2.01	-0.10	-0.06	2.13	-0.10	0.13
1	0	0.00	-0.49	0.00	-0.01	-0.49	0.01
2	0	-1.03	-0.04	0.02	-0.90	-0.03	0.13

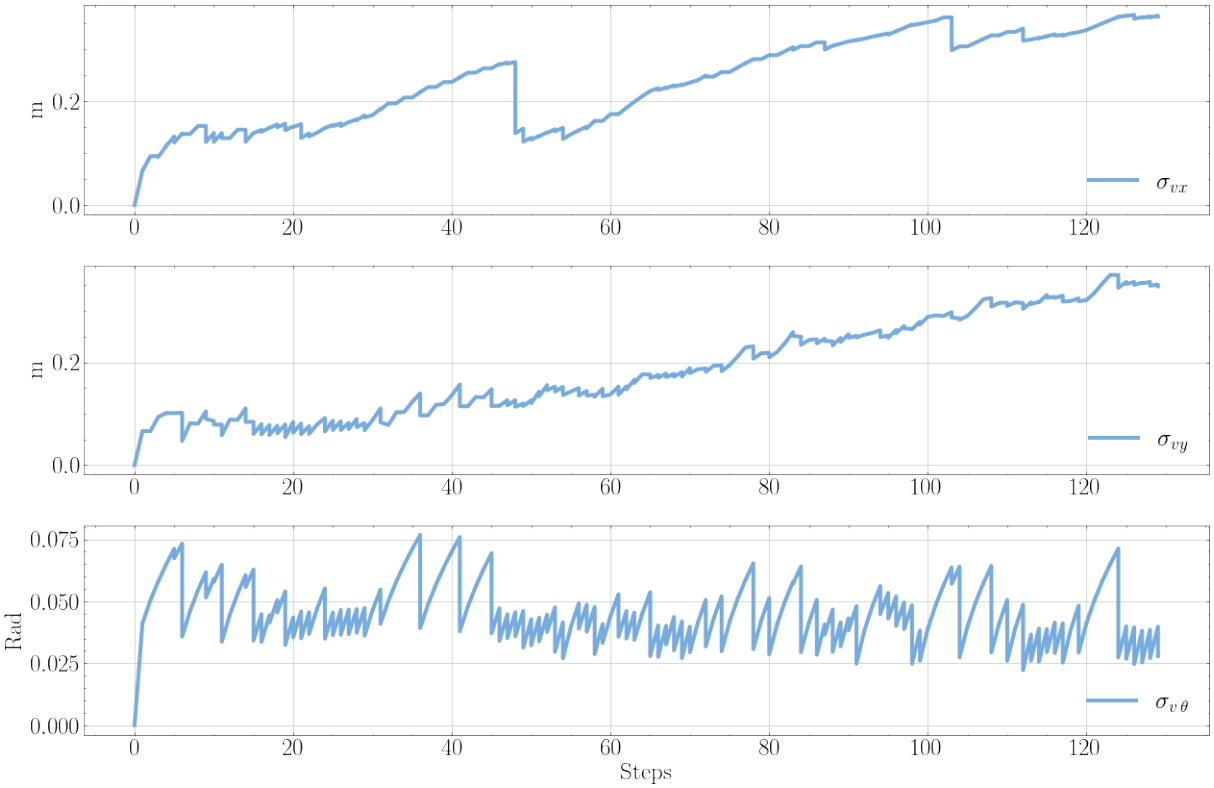


FIGURE 5.5 – Uncertainty of the robot's pose on each axis (σ_{vx}, σ_{vy} and $\sigma_{v\theta}$ respectively) on each step.

Figure 5.7 shows the top view of the corridor. In the high-level map, we see a plane highlighted as **a** that was created because of a door hole. The cylinder **b** appears to be outside of the map. However, this feature is actually a plumbing tube that is close to the wall. Due to its shape, the parameterization made its centroid outside the wall.

The hybrid map manages to simplify 4 of 7 features. In the map, the cylinders are marked as **I**, the cuboid as **II** and the undefined as **III**. 5.4 shows different views of the map reconstruction.

Different from many other SLAM techniques, an interesting aspect of this algorithm is that it enables us to easily access the point cloud of each feature. Table 5.5 shows a comparison between our point cloud reconstruction and the real object.

The more cylindrical the object is, the more precise is the estimated dimensions, as demonstrated by features **2**, **3** and **5**. However, when parameterizing an elongated object, the feature growth method tends to *squarify* the objects' base dimensions, causing the effect illustrated in object **1**, where the width and depth tend to become equal. Finally, when the object's dimensions are out of the boundaries of the image, will certainly cause the wrong parameterization. This happens in object **6** which was partially cut by camera depth range.

In this experiment, we used $\tau_{max} = 0.3$ and obtained a considerable clean map, but the chair in the middle of the corridor wasn't detected at all. By using a $\tau_{max} = 0.45$ it

is possible to detect the chair and also more views of the big box, as shown Figure 5.9. However, many other undesirable point clouds were parameterized as cylinder features. Though it didn't affect much in the estimated path, this can be dangerous since facilitates wrong data associations.

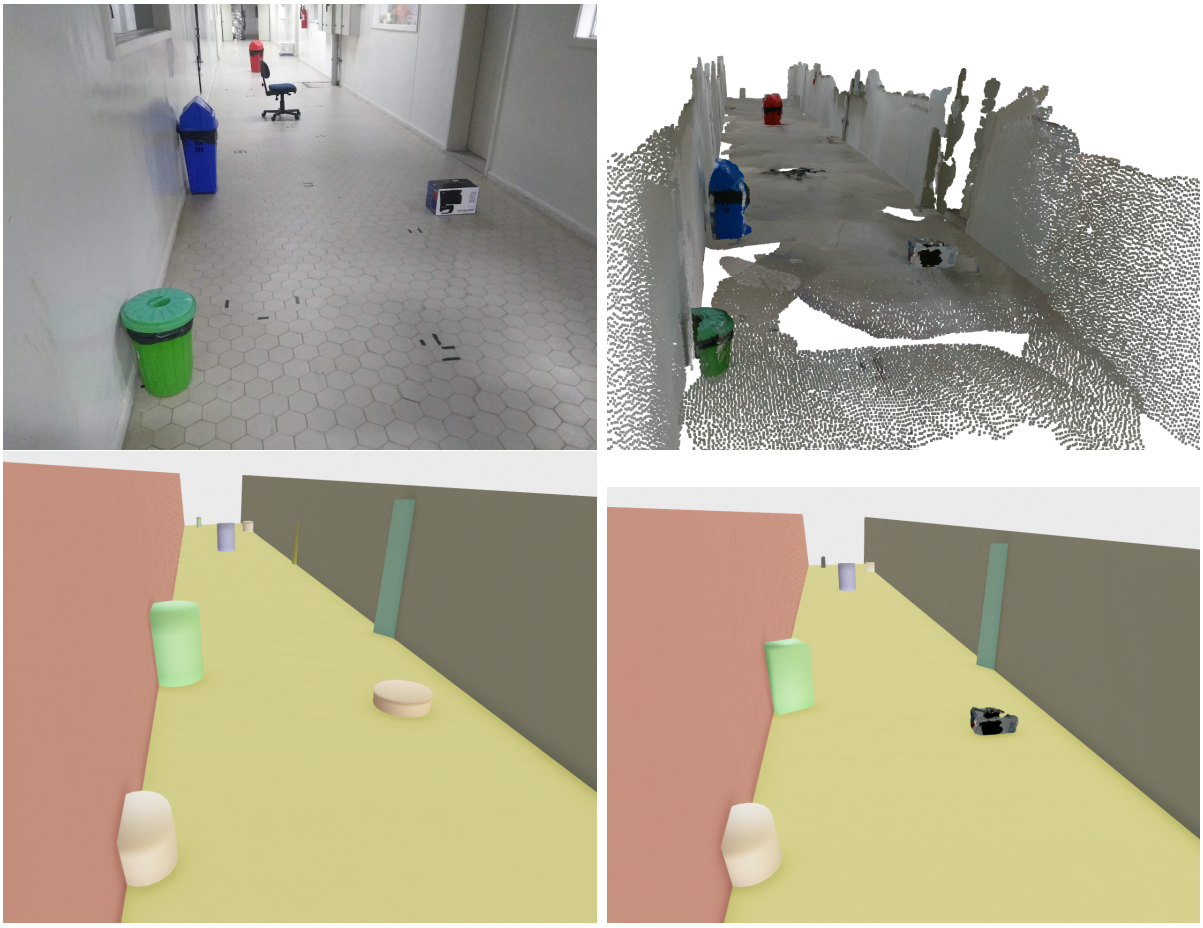


FIGURE 5.6 – Front image of LMI corridor reconstruction. (Top left) shows real image of the experiment, (Top right) illustrate the point cloud reconstructed scene. (Bottom left) shows the high-level environment and (Bottom right) illustrates the hybrid representation.

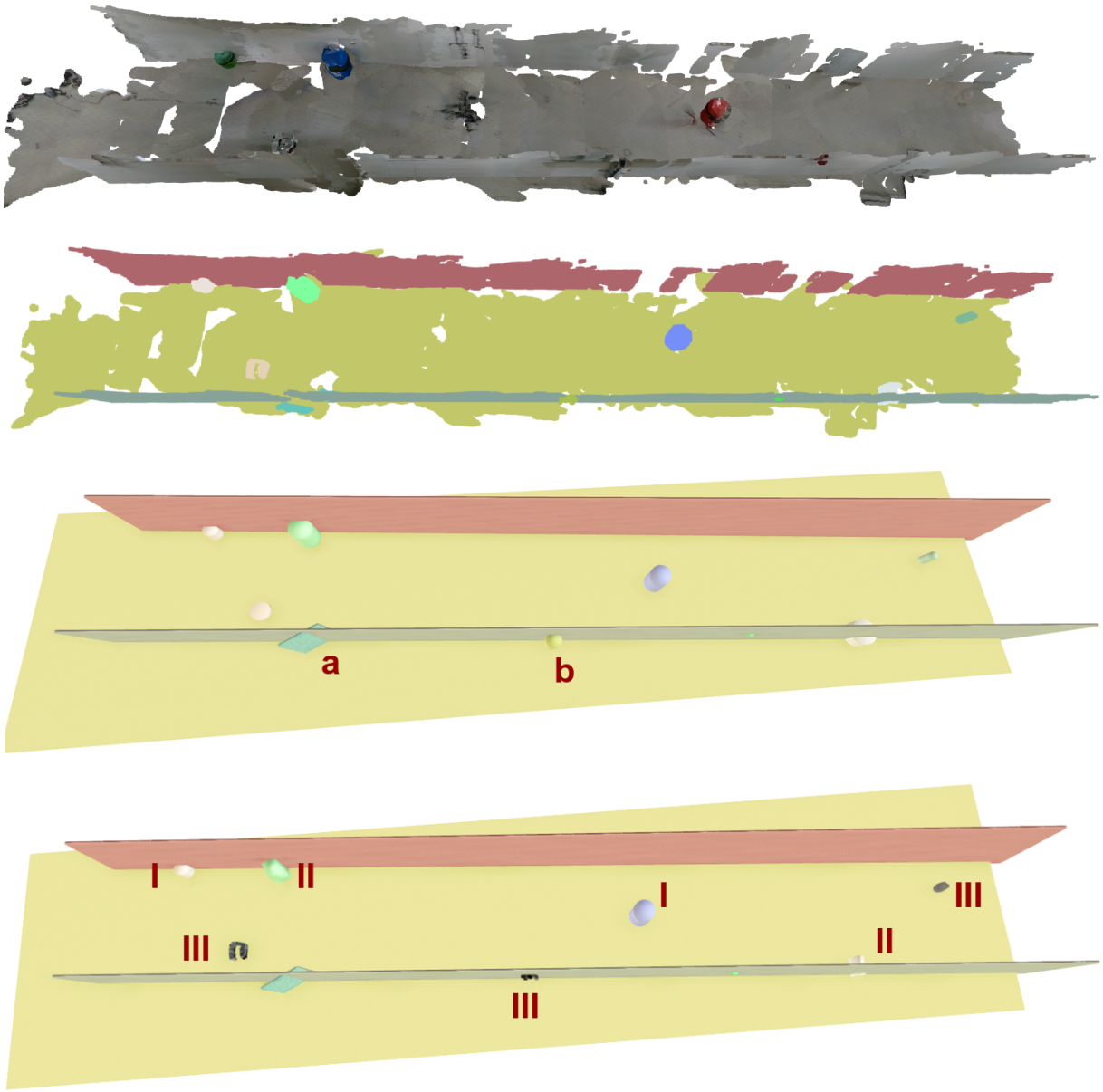


FIGURE 5.7 – Top view of of LMI corridor reconstruction. (First image) shows the point cloud reconstruction, (Second image) illustrates the point cloud reconstruction, but colored each feature individually. (Third image) shows the high-level environment (Fourth image) illustrates the hybrid representation.

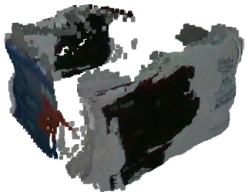
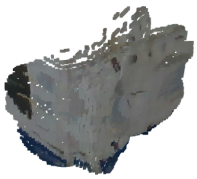
Object	Real object	Reconstruction
1 - Small box 35 x 20 x 15 cm Num. images: 6 **Estimate:** r_{cyl}: 0.244 m H_{cyl}: 0.196 m		
2 - Green trash bin 30 x 30 x 40 cm Num. images: 3 **Estimate:** r_{cyl}: 0.186 m H_{cyl}: 0.397 m		
3 - Blue recycle bin 40 x 30 x 85 cm Num. images: 4 **Estimate:** r_{cyl}: 0.286 m H_{cyl}: 0.848 m		
4 - Plumbing tubes 15 x 10 x 125 cm Num. images: 1 **Estimate:** r_{cyl}: 0.198 m H_{cyl}: 1.255 m		
5 - Red recycle bin 40 x 35 x 85 cm Num. images: 4 **Estimate:** r_{cyl}: 0.267 m H_{cyl}: 0.873 m		
6 - Big box 80 x 80 x 60 cm Num. images: 2 **Estimate:** r_{cyl}: 0.332 m H_{cyl}: 0.459 m		
7 - PVC tube 9 x 9 x 4.5 cm Num. images: 2 **Estimate:** r_{cyl}: 0.112 m H_{cyl}: 0.502 m		

TABLE 5.5 – A side-by-side comparison from the real world object and the estimated feature

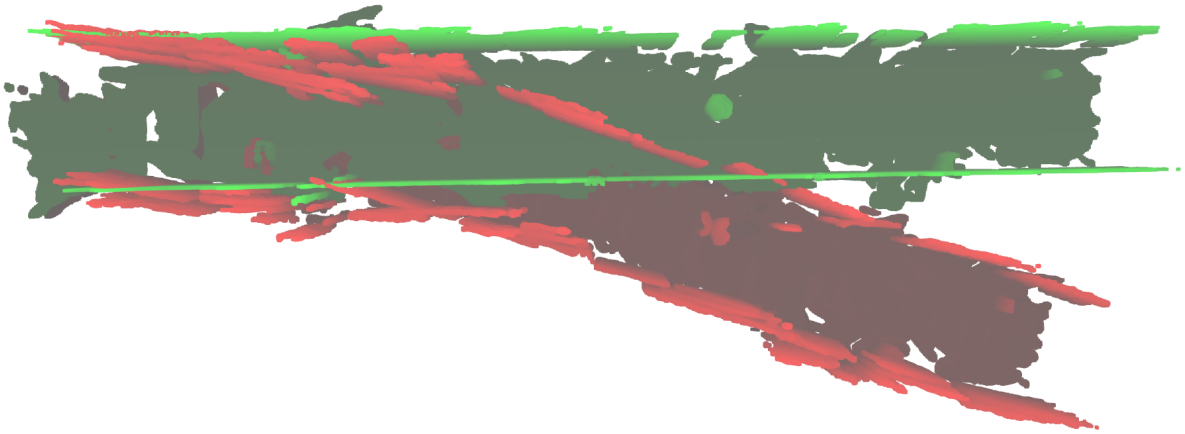


FIGURE 5.8 – The red map is the odometry-based reconstruction. Green map is our SLAM correction.

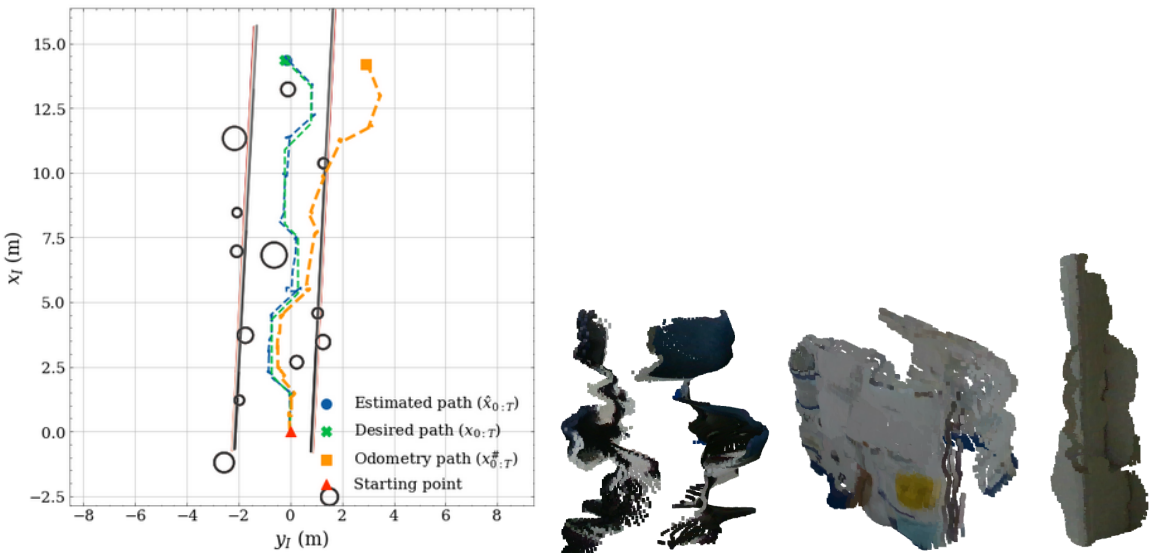


FIGURE 5.9 – By adjuting $\tau_{max} = 0.45$ we detect the chair and more pieces of the big box. However, parts of the walls that should be filtered during the feature extraction are also considered as a cylinders. The map in the left shows the many undesirable features created with this amount of sensitivity.

5.3 Other experiments

We conducted three other less formal experiments in other parts of the department, by moving the robot without a pre-defined reference path.

5.3.1 LMI corridor part 2

Figure 5.10 shows a semi-colored reconstruction of the second part of the LMI corridor. The robot moves through doors and other less structured objects. The image also shows samples of images used to create this reconstruction. The approximate position and orientation of the camera are marked as red squares and the robot's approximate path is marked as cyan.

By looking at the overlap of the odometry map and the estimation in Figure 5.11 we see a small angle error in the corner right of the corridor. This same problem happened in the simulation and is caused by the robot losing reference from previously observed features during turns. Additionally, we see in the high-level map that some panels that were hanging in the right wall caused points blobs that were parameterized as planes and cylinders.

5.3.2 The recycle bin map

We did another experiment in a spacious environment, by moving the robot near trash bins as shown in Figure 5.12. In this wider environment, the robot successfully corrected its pose by having as reference two perpendicular walls and four pillars, reconstructing the map in Figure 5.13. The trash bins were also parameterized as features, but due to their close proximity, the red and yellow bins were associated as the same feature as shown in Figure 5.14.

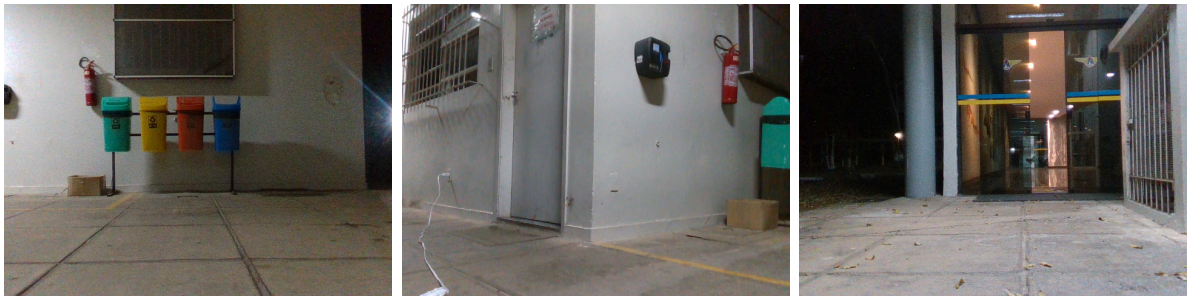


FIGURE 5.12 – Sample images from the recycle bin map.

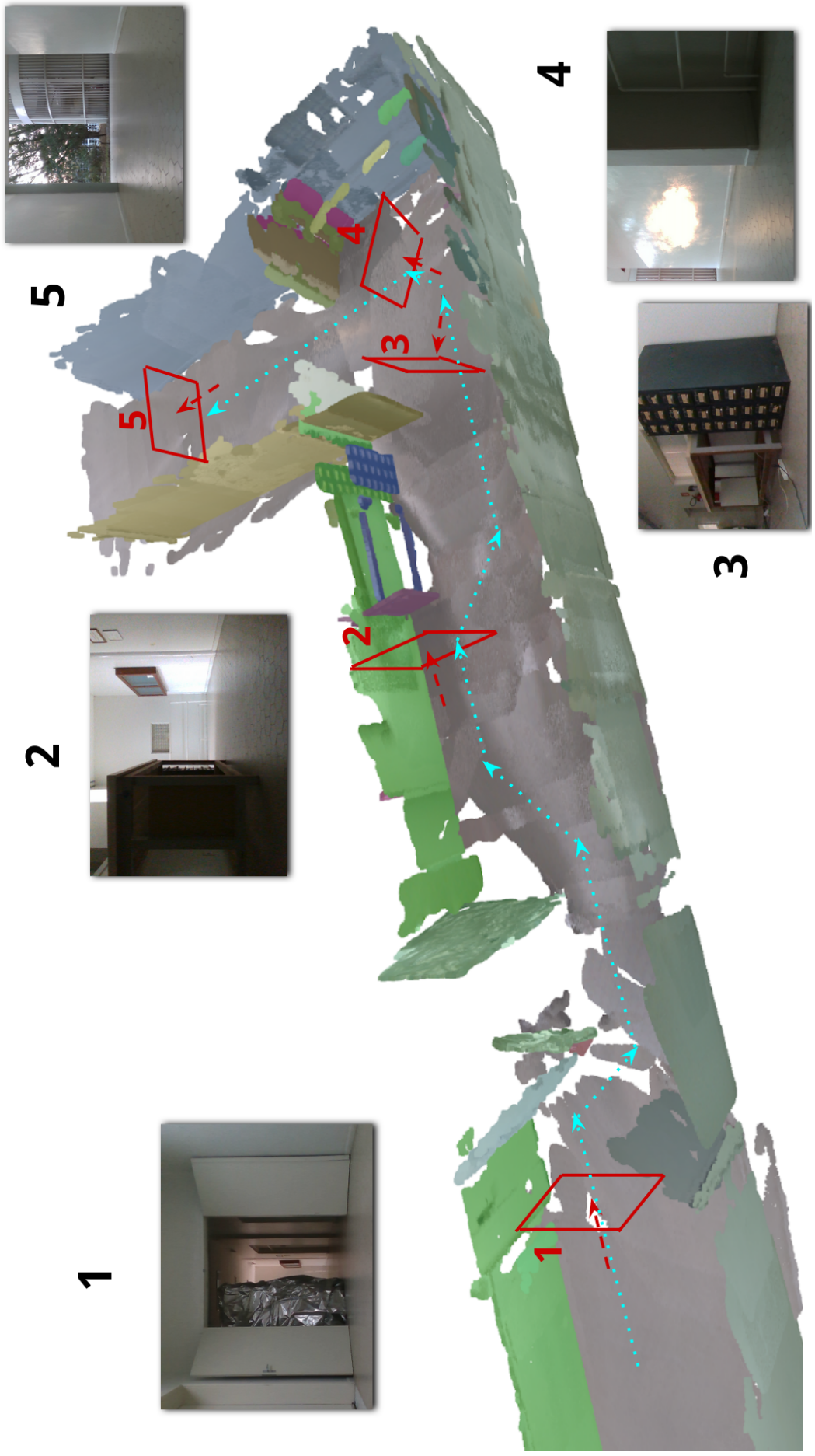


FIGURE 5.10 – Semi-colored point cloud reconstruction for second part of LMI corridor where each feature has its own individual color overlay. The figure also shows some of the images used to reconstruct this scene, where the red frames illustrates the place where each image was taken. The cyan shows an approximate path of the robot.

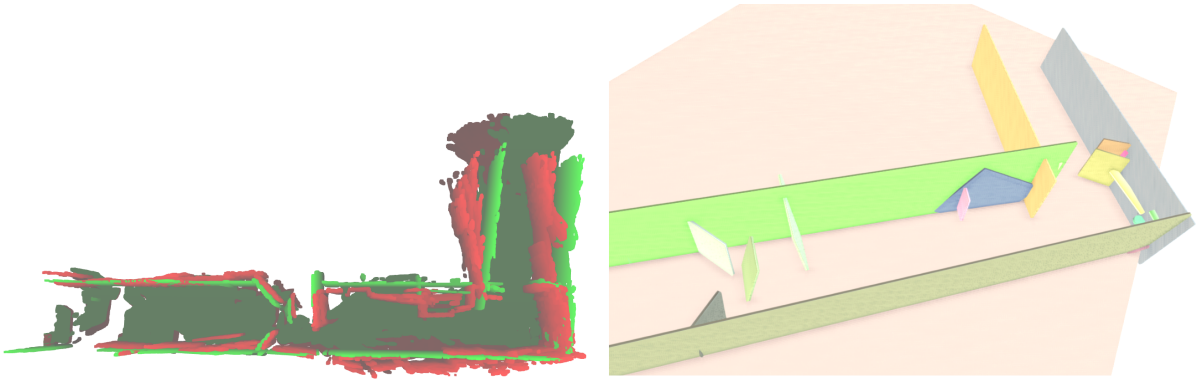


FIGURE 5.11 – Left image is the overlap between the odomentry map and our estimation. Right image is the high-level map.

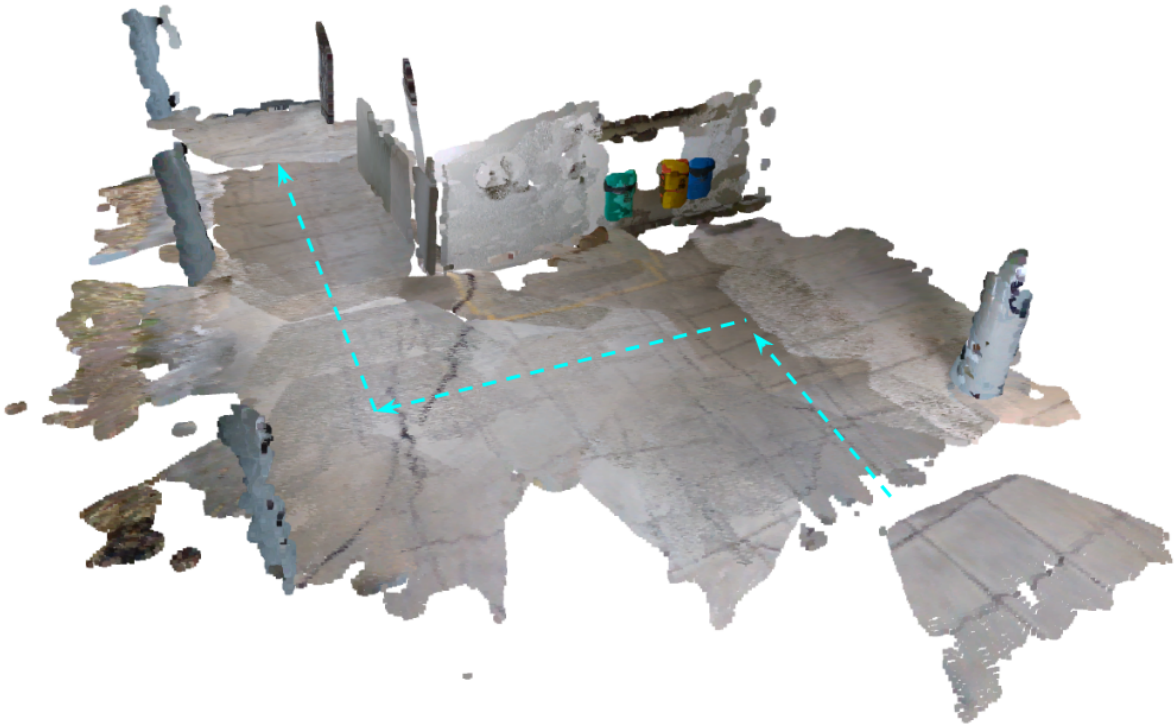


FIGURE 5.13 – Point cloud map of recycle bin scene. The cyan arrows represents the robot's trajectory.

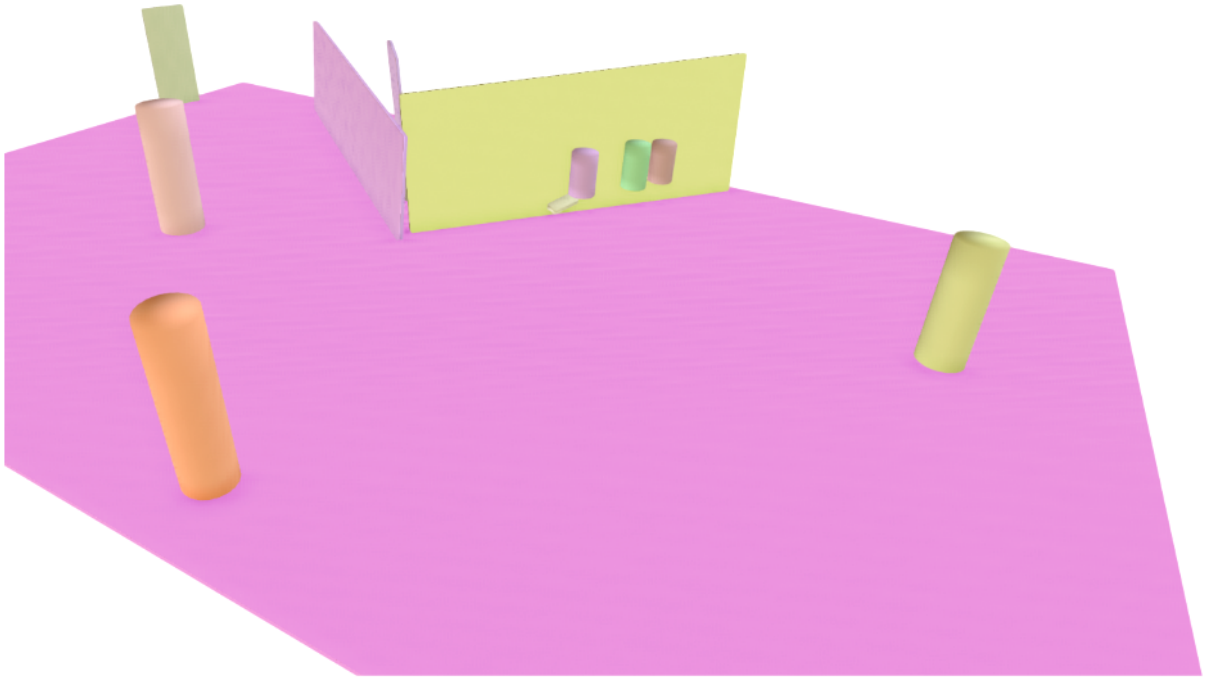


FIGURE 5.14 – High level map of recycle bin scene.

6 Conclusion

The algorithm implemented in this work proved to be successful in its goal of creating concise maps of indoor scenes. The cylinder-and-plane combination allowed the robot to create a parameterized environment, and at the same time, correct its position.

Simulation results showed a considerable reduction in the IEA criteria when comparing the estimated and odometry paths. For both simulations, the estimated path IAE error remained approximately 15% of the odometry path IAE error.

Different from point cloud registration techniques, the proposed algorithm does not match consecutive frames on each other. Actually, it interprets the scene and classifies each point cloud to an EKF feature. Consequently, our solution is not only a basic SLAM technique but also a strategy for scene understanding and structural segmentation.

Though the point cloud can be used to reconstruct a colored realistic environment, the algorithm only requires the parameterized environment to work. Consequently, it can operate in machines with different memory configurations. In this regard, we compared the memory usage for different mapping representations: high-level, point cloud, octree, voxel-grid. A hybrid approach was proposed as a way to intelligently simplify the environment and reduce memory consumption.

The main limitations and difficulties of this solution are summarized as follows: (I) The cylinder representation of a feature does not reconstruct well elongated objects; (II) The cylinder growth algorithm expects that object seen in the camera is completely inside the bounds of the image. (III) A big challenge in this work (and other works in this area) is to differentiate noise blobs from objects of interest. For this work, the parameter τ_{max} worked well for structured objects, but for a more complex object (chair in the experimental result) it could not differentiate from noise.

For future works, we recommend experimental tests in bigger environments, sensitivity tests for other parameters, and experimenting with other data association algorithms. We also could enrich the results by making a comparison with other SLAM techniques (such as graph SLAM).

Many studies are using neural networks to help comprehend the scene. For example,

we could overcome some limitations mentioned above could using semantic segmentation neural network (PHAM *et al.*, 2018), (JABLONSKY *et al.*, 2018) (RAMBACH *et al.*, 2019) or 3D structure prediction (HAN *et al.*, 2020). The addition of these neural networks in our solution would certainly help in building a more realistic environment and making better estimates.

Bibliography

AHN, M. S.; CHAE, H.; NOH, D.; NAM, H.; HONG, D. Analysis and noise modeling of the intel realsense d435 for mobile robots. *In*: **2019 16th International Conference on Ubiquitous Robots (UR)**. **Proceedings** [...]. [*S.l.: s.n.*], 2019. p. 707–711.

BACK, J.-H.; KIM, S.; HO, Y.-S. High-precision 3d coarse registration using ransac and randomly-picked rejections. *In*: SCHOEFFMANN, K.; CHALIDABHONGSE, T. H.; NGO, C. W.; ARAMVITH, S.; O'CONNOR, N. E.; HO, Y.-S.; GABBOUJ, M.; ELGAMMAL, A. (Ed.). **MultiMedia Modeling**. **Proceedings** [...]. Cham: Springer International Publishing, 2018. p. 254–266. ISBN 978-3-319-73603-7.

BAY, H.; TUYTELAARS, T.; GOOL, L. V. Surf: Speeded up robust features. *In*: LEONARDIS, A.; BISCHOF, H.; PINZ, A. (Ed.). **Computer Vision – ECCV 2006**. **Proceedings** [...]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 404–417. ISBN 978-3-540-33833-8.

BESL, P.; MCKAY, N. D. A method for registration of 3-d shapes. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 14, n. 2, p. 239–256, 1992.

BRANNON, R. M. Rates and other derivatives of rotation. *In*: **Rotation, Reflection, and Frame Changes**. [*S.l.*]: IOP Publishing, 2018, (2053-2563). p. 15–1 to 15–21. ISBN 978-0-7503-1454-1.

CALONDER, M.; LEPETIT, V.; STRECHA, C.; FUA, P. Brief: Binary robust independent elementary features. *In*: DANIILIDIS, K.; MARAGOS, P.; PARAGIOS, N. (Ed.). **Computer Vision – ECCV 2010**. **Proceedings** [...]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 778–792. ISBN 978-3-642-15561-1.

CASELLA, G.; ROBERT, C. P. Rao-Blackwellisation of sampling schemes. **Biometrika**, v. 83, n. 1, p. 81–94, 03 1996. ISSN 0006-3444. Available at: https://doi.org/10.1093/biomet/83.1.81.

CASTELLANOS, J.; MONTIEL, J.; NEIRA, J.; TARDOS, J. The spmap: a probabilistic framework for simultaneous localization and map building. **IEEE Transactions on Robotics and Automation**, v. 15, n. 5, p. 948–952, 1999.

Chelishchev, Petr; Sørby, Knut. Estimation of minimum volume of bounding box for geometrical metrology. **Int. J. Metrol. Qual. Eng.**, v. 11, p. 9, 2020. Available at: https://doi.org/10.1051/ijmqe/2020007.

CHOI, S.; ZHOU, Q.-Y.; KOLTUN, V. Robust reconstruction of indoor scenes. *In*: **2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. **Proceedings** [...]. [*S.l.: s.n.*], 2015. p. 5556–5565.

CHOI, Y.-H.; LEE, T.; OH, S.-Y. A line feature based slam with low grade range sensors using geometric constraints and active exploration for mobile robot. **Autonomous Robots**, v. 24, p. 13–27, 2008.

CONNETTE, C.; MEISTER, O.; HäGELE, M.; TROMMER, G. Decomposition of line segments into corner and statistical grown line features in an ekf-slam framework. *In*: . **Proceedings** [...]. [*S.l.: s.n.*], 2007. p. 3884 – 3891.

CORKE, P. **Robotics, Vision and Control: Fundamental Algorithms in MATLAB**. 1st. ed. [*S.l.*]: Springer Publishing Company, Incorporated, 2013. ISBN 3642201431.

C++. **CPP Reference - Unordered Map**. 2021. `https://en.cppreference.com/w/cpp/container/unordered_map`. Last accessed 2015-11-01.

DELLAERT, F.; KAESS, M. Square root sam: Simultaneous localization and mapping via square root information smoothing. **The International Journal of Robotics Research**, v. 25, n. 12, p. 1181–1203, 2006. Available at: https://doi.org/10.1177/0278364906072768.

ECK, M. Simultaneous 2d localization and 3d mapping on a mobile robot with time-of-flight sensors. 10 2013.

ELIAZAR, A.; PARR, R. Dp-slam: Fast, robust simultaneous localization and mapping without predetermined landmarks. *In*: **Proceedings of the 18th International Joint Conference on Artificial Intelligence**. **Proceedings** [...]. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003. (IJCAI'03), p. 1135–1142.

ESTER, M.; KRIEGEL, H.-P.; SANDER, J.; XU, X. A density-based algorithm for discovering clusters in large spatial databases with noise. *In*: **Proceedings of the Second International Conference on Knowledge Discovery and Data Mining**. **Proceedings** [...]. [*S.l.*]: AAAI Press, 1996. (KDD'96), p. 226–231.

FISCHLER, M. A.; BOLLES, R. C. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 24, n. 6, p. 381–395, jun. 1981. ISSN 0001-0782. Available at: https://doi.org/10.1145/358669.358692.

FUKUNAGA, K.; HOSTETLER, L. The estimation of the gradient of a density function, with applications in pattern recognition. **IEEE Transactions on Information Theory**, v. 21, n. 1, p. 32–40, 1975.

GARULLI, A.; GIANNITRAPANI, A.; ROSSI, A.; VICINO, A. Mobile robot slam for line-based environment representation. *In*: **Proceedings of the 44th IEEE Conference on Decision and Control**. **Proceedings** [...]. [*S.l.: s.n.*], 2005. p. 2041–2046.

GEIER, D. **Advanced Octrees 2: node representations**. Aug 2014. Available at: https://geidav.wordpress.com/2014/08/18/advanced-octrees-2-node-representations/.

GHARATAPPEH, S.; GHORBANIAN, M.; KESHMIRI, M.; TAGHIRAD, H. D. Modified fast-slam for 2d mapping and 3d localization. *In*: **2015 3rd RSI International Conference on Robotics and Mechatronics (ICROM)**. **Proceedings** [...]. [*S.l.: s.n.*], 2015. p. 108–113.

GORDON, N. Novel approach to nonlinear/non-gaussian bayesian state estimation. **IEE Proceedings F (Radar and Signal Processing)**, v. 140, p. 107–113(6), April 1993. ISSN 0956-375X. Available at: https://digital-library.theiet.org/content/journals/10.1049/ip-f-2.1993.0015.

GRISETTI, G.; KüMMERLE, R.; STACHNISS, C.; FRESE, U.; HERTZBERG, C. Hierarchical optimization on manifolds for online 2d and 3d mapping. *In*: **2010 IEEE International Conference on Robotics and Automation**. **Proceedings** [...]. [*S.l.: s.n.*], 2010. p. 273–278.

GRISETTI, G.; KüMMERLE, R.; STACHNISS, C.; BURGARD, W. A tutorial on graph-based slam. **IEEE Intelligent Transportation Systems Magazine**, v. 2, n. 4, p. 31–43, 2010.

HAN, J.; WANG, F.; GUO, Y.; ZHANG, C.; HE, Y. An improved ransac registration algorithm based on region covariance descriptor. *In*: **2015 Chinese Automation Congress (CAC)**. **Proceedings** [...]. [*S.l.: s.n.*], 2015. p. 746–751.

HAN, S.; GU, J.; MO, K.; YI, L.; HU, S.; CHEN, X.; SU, H. Compositionally Generalizable 3D Structure Prediction. **arXiv preprint**, 2020.

HARTLEY, R.; ZISSERMAN, A. **Multiple View Geometry in Computer Vision**. 2. ed. USA: Cambridge University Press, 2003. ISBN 0521540518.

HORNUNG, A.; WURM, K. M.; BENNEWITZ, M.; STACHNISS, C.; BURGARD, W. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. **Autonomous Robots**, software available at `https://octomap.github.io`, 2013. Available at: https://octomap.github.io.

HOUGH, P. V. C. Method and means for recognizing complex patterns. **U.S. Patent, no.3069654**, 1962. Available at: https://ci.nii.ac.jp/naid/10016796402/en/.

INTEL, R. **Gazebo RealSense Plugin**. [*S.l.*]: GitHub, 2013. `https://github.com/intel/gazebo-realsense`.

JABLONSKY, N.; MILFORD, M.; SüNDERHAUF, N. **An Orientation Factor for Object-Oriented SLAM**. 2018.

JAFRI, S. R. U. N.; IQBAL, J.; KHAN, H.; CHELLALI, R. A unified slam solution using partial 3d structure. **Elektronika ir Elektrotechnika**, v. 20, p. 3–8, 11 2014.

JENSFELT, P.; KRISTENSEN, S. Active global localisation for a mobile robot using multiple hypothesis tracking. 08 2000.

JO, H.; JO, S.; KIM, E.; YOON, C.; JUN, S. 3d fastslam algorithm with kinect sensor. *In*: **2014 Joint 7th International Conference on Soft Computing and Intelligent Systems (SCIS) and 15th International Symposium on Advanced Intelligent Systems (ISIS)**. **Proceedings** [...]. [*S.l.: s.n.*], 2014. p. 214–217.

JULIER, S.; UHLMANN, J. New extension of the kalman filter to nonlinear systems. *In*: **Defense, Security, and Sensing**. **Proceedings** [...]. [*S.l.: s.n.*], 1997.

KAESS, M. Simultaneous localization and mapping with infinite planes. **2015 IEEE International Conference on Robotics and Automation (ICRA)**, p. 4605–4611, 2015.

KALMAN, R. E. A new approach to linear filtering and prediction problems. **Transactions of the ASME–Journal of Basic Engineering**, v. 82, n. Series D, p. 35–45, 1960.

KOENIG, N.; HOWARD, A. Design and use paradigms for gazebo, an open-source multi-robot simulator. *In*: **2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)**. **Proceedings** [...]. [*S.l.: s.n.*], 2004. v. 3, p. 2149–2154 vol.3.

KWAK, N.; KIM, I.-K.; LEE, H.-C.; LEE, B.-H. Analysis of resampling process for the particle depletion problem in fastslam. *In*: **RO-MAN 2007 - The 16th IEEE International Symposium on Robot and Human Interactive Communication**. **Proceedings** [...]. [*S.l.: s.n.*], 2007. p. 200–205.

KüMMERLE, R.; GRISETTI, G.; STRASDAT, H.; KONOLIGE, K.; BURGARD, W. G2o: A general framework for graph optimization. *In*: **2011 IEEE International Conference on Robotics and Automation**. **Proceedings** [...]. [*S.l.: s.n.*], 2011. p. 3607–3613.

LABORATORY, R. P. I. I. P.; MEAGHER, D. **Octree Encoding: a New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer**. [*S.l.: s.n.*], 1980. Available at: https://books.google.com.br/books?id=CgRPOAAACAAJ.

LEI, X.; FENG, B.; WANG, G.; LIU, W.; YANG, Y. A novel fastslam framework based on 2d lidar for autonomous mobile robot. **Electronics**, v. 9, p. 695, 04 2020.

LEONARD, J.; DURRANT-WHYTE, H. Mobile robot localization by tracking geometric beacons. **IEEE Transactions on Robotics and Automation**, v. 7, n. 3, p. 376–382, 1991.

LEUTENEGGER, S.; CHLI, M.; SIEGWART, R. Brisk: Binary robust invariant scalable keypoints. **2011 International Conference on Computer Vision**, p. 2548–2555, 2011.

LI, J.; HU, Q.; AI, M. Point cloud registration based on one-point ransac and scale-annealing biweight estimation. **IEEE Transactions on Geoscience and Remote Sensing**, p. 1–14, 2021.

LI, Y.; OLSON, E. B. Ipjc: The incremental posterior joint compatibility test for fast feature cloud matching. *In*: **2012 IEEE/RSJ International Conference on Intelligent Robots and Systems**. **Proceedings** [...]. [*S.l.: s.n.*], 2012. p. 3467–3474.

LIU, J.; WU, Z.-k. An adaptive approach for primitive shape extraction from point clouds. **Optik - International Journal for Light and Electron Optics**, v. 125, 05 2014.

LLOYD, S. Least squares quantization in pcm. **IEEE Transactions on Information Theory**, v. 28, n. 2, p. 129–137, 1982.

LU, F.; MILIOS, E. Globally consistent range scan alignment for environment mapping. **Autonomous Robots**, v. 4, p. 333–349, 1997.

MACHINERY, A. for C. **2014 SIGKDD Test of Time Award**. [*S.l.*]: Special Interest Group on Knowledge Discovery and Data Minin, Aug 2013. `https://www.kdd.org/News/view/2014-sigkdd-test-of-time-award`.

MAHALANOBIS, P. C. On the generalized distance in statistics. **Proceedings of the National Institute of Sciences (Calcutta)**, v. 2, p. 49–55, 1936.

MAHMOUD, D.; SALEM, M. A.-M. M.; RAMADAN, H.; ROUSHDY, M. 3d graph-based vision-slam registration and optimization. **International Journal of Circuits, Systems and Signal Processing**, v. 8, p. 123–, 06 2014.

MAKARENKO, A.; WILLIAMS, S.; BOURGAULT, F.; DURRANT-WHYTE, H. An experiment in integrated exploration. *In*: **IEEE/RSJ International Conference on Intelligent Robots and Systems**. **Proceedings** [...]. [*S.l.: s.n.*], 2002. v. 1, p. 534–539 vol.1.

MARSHALL, D.; LUKACS, G.; MARTIN, R. Robust segmentation of primitives from range data in the presence of geometric degeneracy. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 23, n. 3, p. 304–314, 2001.

MASÓ, A.; JOSEP, M. Selective submap joining slam for autonomous vehicles. *In*: . **Proceedings** [...]. [*S.l.: s.n.*], 2011.

MONTEMERLO, M.; THRUN, S.; KOLLER, D.; WEGBREIT, B. Fastslam: A factored solution to the simultaneous localization and mapping problem. *In*: . **Proceedings** [...]. [*S.l.: s.n.*], 2002.

MUGLIKAR, M.; ZHANG, Z.; SCARAMUZZA, D. Voxel map for visual SLAM. **CoRR**, abs/2003.02247, 2020. Available at: https://arxiv.org/abs/2003.02247.

NARDI, F.; CORTE, B. D.; GRISETTI, G. Unified representation and registration of heterogeneous sets of geometric primitives. **IEEE Robotics and Automation Letters**, v. 4, n. 2, p. 625–632, 2019.

NASCIMENTO JR., C. Estudo comparativo de métodos de combate à divergência de filtros de kalman. **BDITA**, p. 263, 1988.

NG, A. Y.; JORDAN, M. I.; WEISS, Y. On spectral clustering: Analysis and an algorithm. *In*: **Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic**. **Proceedings** [...]. Cambridge, MA, USA: MIT Press, 2001. (NIPS'01), p. 849–856.

OLIVEIRA, H.; SOUSA, A.; MOREIRA, A.; COSTA, P. Precise modeling of a four wheeled omni-directional robot. 04 2008.

OZAKI, R.; KURODA, Y. 6-dof ekf slam with global planar features in artificial environments. *In*: **2020 IEEE/SICE International Symposium on System Integration (SII)**. **Proceedings** [...]. [*S.l.: s.n.*], 2020. p. 531–535.

PEDDURI, S.; KOSURU, G.; KRISHNA, M.; PANDEY, A. K. On measurement models for line segments and point based slam. *In*: . **Proceedings** [...]. [*S.l.: s.n.*], 2009. p. 1 – 6.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine Learning in Python . **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

PHAM, T.; DO, T.-T.; SüNDERHAUF, N.; REID, I. **SceneCut: Joint Geometric and Object Segmentation for Indoor Scenes**. 2018.

PINTO, L. d. S.; FILHO, L. E. S. A.; MARIGA, L.; J?NIOR, C. L. N.; CUNHA, W. C. Ekf-slam with autonomous exploration using a low cost robot. *In*: **2021 IEEE International Systems Conference (SysCon)**. **Proceedings** [...]. [*S.l.: s.n.*], 2021. p. 1–7.

QUER, S.; MARCELLI, A.; SQUILLERO, G. The maximum common subgraph problem: A parallel and multi-engine approach. **Computation**, MDPI AG, v. 8, n. 2, p. 48, May 2020. ISSN 2079-3197. Available at: http://dx.doi.org/10.3390/computation8020048.

RAMBACH, J.; LESUR, P.; PAGANI, A.; STRICKER, D. Slamcraft: Dense planar rgb monocular slam. *In*: **2019 16th International Conference on Machine Vision Applications (MVA)**. **Proceedings** [...]. [*S.l.: s.n.*], 2019. p. 1–6.

RIISGAARD, S.; BLAS, M. R. **SLAM for Dummies: A Tutorial Approach to Simultaneous Localization and Mapping**. [*S.l.*], 2005. Available at: http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf.

ROSTEN, E.; DRUMMOND, T. Machine learning for high-speed corner detection. *In*: LEONARDIS, A.; BISCHOF, H.; PINZ, A. (Ed.). **Computer Vision – ECCV 2006**. **Proceedings** [...]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 430–443. ISBN 978-3-540-33833-8.

RUBLEE, E.; RABAUD, V.; KONOLIGE, K.; BRADSKI, G. Orb: An efficient alternative to sift or surf. *In*: **2011 International Conference on Computer Vision**. **Proceedings** [...]. [*S.l.: s.n.*], 2011. p. 2564–2571.

SCHNABEL, R.; WAHL, R.; KLEIN, R. Efficient ransac for point-cloud shape detection. **Computer Graphics Forum**, Blackwell Publishing, v. 26, n. 2, p. 214–226, jun. 2007.

SCHUBERT, E.; SANDER, J.; ESTER, M.; KRIEGEL, H. P.; XU, X. Dbscan revisited, revisited: Why and how you should (still) use dbscan. **ACM Trans. Database Syst.**, Association for Computing Machinery, New York, NY, USA, v. 42, n. 3, jul. 2017. ISSN 0362-5915. Available at: https://doi.org/10.1145/3068335.

SHEN, X.; FRAZZOLI, E.; RUS, D.; ANG, M. H. Fast joint compatibility branch and bound for feature cloud matching. *In*: **2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. **Proceedings** [...]. [*S.l.: s.n.*], 2016. p. 1757–1764.

SMITH, R.; SELF, M.; CHEESEMAN, P. A stochastic map for uncertain spatial relationships. p. 467–474, 01 1987.

STACHNISS, C. Robot mapping - ekf slam. *In*: . **Proceedings** [...]. [*S.l.: s.n.*], 2011.

The Construct. **Robot Modeling - using gazebo plugins to simulate control mecanum wheels robot - EP.3**. Sep 2019. `https://www.theconstructsim.com/use-gazebo-plugins-simulate-control-robot`.

THRUN, S.; BURGARD, W.; FOX, D. **Probabilistic robotics**. Cambridge, Mass.: MIT Press, 2005. ISBN 0262201623 9780262201629. Available at: http://www.amazon.de/gp/product/0262201623.

TOONY, Z.; LAURENDEAU, D.; GAGNé, C. Pgp2x: Principal geometric primitives parameters extraction. *In*: . **Proceedings** [...]. [*S.l.: s.n.*], 2015.

ULAS, C.; TEMELTAS, H. Plane-feature based 3d outdoor slam with gaussian filters. *In*: **2012 IEEE International Conference on Vehicular Electronics and Safety (ICVES 2012)**. **Proceedings** [...]. [*S.l.: s.n.*], 2012. p. 13–18.

URBANčIč, T.; FRAS, M.; STOPAR, B.; BOžO, K. The influence of the input parameters selection on the ransac results. **International Journal of Simulation Modelling**, v. 13, p. 159–170, 06 2014.

WARD, J. H. Hierarchical grouping to optimize an objective function. **Journal of the American Statistical Association**, v. 58, p. 236–244, 1963.

WEINGARTEN, J. Feature-based 3d slam. EPFL, Lausanne, p. 137, 2006. Available at: http://infoscience.epfl.ch/record/86078.

WEN-PENG, GUANG-YUN, MING-LEI, LI, SHUAI-XIN, survey of laser scan matching methods. **Chinese Optics**, v. 11, p. 914–930, 12 2018.

XU, G.; PANG, Y.; BAI, Z.; WANG, Y.; LU, Z. A fast point clouds registration algorithm for laser scanners. **Applied Sciences**, v. 11, n. 8, 2021. ISSN 2076-3417. Available at: https://www.mdpi.com/2076-3417/11/8/3426.

ZHOU, Q.-Y.; PARK, J.; KOLTUN, V. Fast global registration. *In*: LEIBE, B.; MATAS, J.; SEBE, N.; WELLING, M. (Ed.). **Computer Vision – ECCV 2016**. **Proceedings** [...]. Cham: Springer International Publishing, 2016. p. 766–782. ISBN 978-3-319-46475-6.

ZHOU, Q.-Y.; PARK, J.; KOLTUN, V. **Open3D: A Modern Library for 3D Data Processing**. 2018. Available at: http://arxiv.org/abs/1801.09847.

Appendix A - Rough memory models

A.1 Rough memory model for parameterized primitive features

Element	Propriety group	Propriety	Type	Memory usage C++ (bytes)	Memory usage C++ (bytes)
Plane	Infinite plane representation	n_x	float	4	12
		n_y	float	4	
		n_z	float	4	
	2D center	c_x	float	4	8
		c_y	float	4	
	2D rotation	psi_p	float	4	4
	2D size	width	float	4	8
		height	float	4	
	Color	Cor_x	unsigned char	1	3
		Cor_y	unsigned char	1	
		Cor_z	unsigned char	1	
Cylinder	Position	Pos_x	float	4	12
		Pos_y	float	4	
		Pos_z	float	4	
	Size	height	float	4	8
		radius	float	4	
	Color	Cor_x	unsigned char	1	3
		Cor_y	unsigned char	1	
		Cor_z	unsigned char	1	
Cuboid	Position	Pos_x	float	4	12
		Pos_y	float	4	
		Pos_z	float	4	
	Size	height	float	4	8
		width	float	4	
		depth	float	4	
	2D rotation	psi_p	float	4	4
	Color	Cor_x	unsigned char	1	3
		Cor_y	unsigned char	1	
		Cor_z	unsigned char	1	
Sphere	Position	Pos_x	float	4	12
		Pos_y	float	4	
		Pos_z	float	4	
	Size	radius	float	4	4
	Color	Cor_x	unsigned char	1	3
		Cor_y	unsigned char	1	
		Cor_z	unsigned char	1	

A.2 Rough memory model for point cloud

Point cloud					
Element	Propriety group	Propriety	Type	Memory usage C++ (bytes)	Memory usage C++ (bytes)
Point	Position	Pos_x	float	4	12
		Pos_y	float	4	
		Pos_z	float	4	
	Color	Cor_x	unsigned char	1	3
		Cor_y	unsigned char	1	
		Cor_z	unsigned char	1	

A.3 Rough memory model for voxel grid

Voxel grid					
Element	Propriety group	Propriety	Type	Memory usage C++ (bytes)	Memory usage C++ (bytes)
Grid	Origin	Pos_x	float	4	12
		Pos_y	float	4	
		Pos_z	float	4	
	Size	Size	float	4	4
	unordered_map structure			128	128
Bucket	head pointer per bucket			16	16
Non-empty Voxel	Next pointer	Pointer	ptr*	8	8
	grid_index	i_x	int	4	12
		i_y	int	4	
		i_z	int	4	
	Color	Cor_x	unsigned char	1	3
		Cor_y	unsigned char	1	
		Cor_z	unsigned char	1	

A.4 Rough memory model for Octree

Octree					
Element	Propriety group	Propriety	Type	Memory usage C++ (bytes)	Memory usage C++ (bytes)
Root node	Origin	Pos_x	float	4	12
		Pos_y	float	4	
		Pos_z	float	4	
	Size	Size	float	4	4
	Division	max_division	int	4	4
	Children pointer	c_ptr [0]	pt*	8	64
		c_ptr [1]	pt*	8	
		c_ptr [2]	pt*	8	
		c_ptr [3]	pt*	8	
		c_ptr [4]	pt*	8	
		c_ptr [5]	pt*	8	
		c_ptr [6]	pt*	8	
		c_ptr [7]	pt*	8	
Internal node	Children pointer	c_ptr [0]	pt*	8	64
		c_ptr [1]	pt*	8	
		c_ptr [2]	pt*	8	
		c_ptr [3]	pt*	8	
		c_ptr [4]	pt*	8	
		c_ptr [5]	pt*	8	
		c_ptr [6]	pt*	8	
		c_ptr [7]	pt*	8	
Leaf node	Color	Cor_x	unsigned char	1	3
		Cor_y	unsigned char	1	
		Cor_z	unsigned char	1	

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO	2. DATA	3. REGISTRO N°	4. N° DE PÁGINAS
DM	21 de janeiro de 2022	DCTA/ITA/DM-117/2021	123

5. TÍTULO E SUBTÍTULO:

Solving the Simultaneous Localization and 3D Mapping Problem in Mobile Robotics Using Multi-Level Parameterized Representations

6. AUTOR(ES):

Leonardo Mariga

7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES):

Instituto Tecnológico de Aeronáutica - ITA

8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR:

EKF, 3D mapping, SLAM

9.PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

Dinâmica de robos; Exploração; Navegação autônoma; Robótica; Controle

10. APRESENTAÇÃO: **(X) Nacional ( ) Internacional**

ITA, São José dos Campos. Curso de Mestrado. Programa de Pós-Graduação em Engenharia Eletrônica e Computação. Área de Sistemas e Controle. Orientador: Prof. Dr. Cairo Lúcio Nascimento Júnior. Defesa em 15/12/2021. Publicada em 2021.

11. RESUMO:

The use of mobile robots for mapping and navigating unknown environments has recently gained importance in military applications, search and rescue missions and domestic activities. This work proposes a solution for the Simultaneous Location and Mapping (SLAM) problem using an expansion for the 3D environment of the landmark-based Extended Kalman Filter SLAM (EKF-SLAM). We propose an algorithm that uses a point-and-plane representation to map indoor environments using a 3-DoF model of a mobile robot with an RGBD camera. Our solution reconstructs a point cloud map by initially simplifying the scene into parameterized features: cylinders and planar segments. Then, we segment and associate the point cloud to its specific EKF feature. The unified use of both parameterized features and point cloud enables the creation of maps using different representations and levels of abstraction, which is helpful when working in applications with limited memory requirements. This work starts by reviewing the methods available in the literature. Then, we specify the EKF-SLAM framework used for this work. In the feature extraction method, we use RANSAC for planar segmentation. The non-planar points are clustered using DB-SCAN and parameterized as cylinders after passing through validation gates. On each step, the EKF framework propagates and updates the features' states. This also triggers the point-cloud growth and increments the point cloud of the observed feature. Additionally, we make a comparative study of five different 3D mapping representations (point cloud, primitive geometries, and voxels) in terms of memory usage. The simulation results show that this method successfully reconstructs maps, reducing the IAE criteria to 15% of trajectory error compared to the odometry. Finally, we validate the algorithm in a real environment, using an Intel Realsense camera. Despite suffering from limitations in the camera and in the feature extraction process, the work proved to be a strong method for scene understanding along with the basic SLAM capabilities.

12. GRAU DE SIGILO:

(X) OSTENSIVO **( ) RESERVADO** **( ) SECRETO**