UNIVERSIDADE D
COIMBRA

Lucyanno Frota Fernandes

SMAP

A SEMANTIC MAPPING FRAMEWORK FOR MOBILE ROBOTS

Master's dissertation submitted in partial fulfilment of the Master Degree in Electrical and Computer Engineering specialization in Robotics Control and Artificial Intelligence, supervised by Professor Doctor Rui Paulo Pinto da Rocha, and presented to the Department of Electrical and Computer Engineering of Faculty of Sciences and Technology of University of Coimbra.

October, 2023

UNIVERSIDADE D
COIMBRA

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

SMAP: A Semantic Mapping Framework for Mobile Robots

Lucyanno Frota Fernandes

Coimbra, October 2023

SMAP: A Semantic Mapping Framework for Mobile Robots

Supervisor:

Prof. Doctor Rui Paulo Pinto da Rocha

Jury:

Prof. Doctor Jorge Manuel Moreira de Campos Pereira Batista

Prof. Doctor Lino José Forte Marques

Prof. Doctor Rui Paulo Pinto da Rocha

Dissertation submitted in partial fulfillment for the degree of Master of Science in Electrical and Computer Engineering specialization in Robotics Control and Artificial Intelligence.

Coimbra, October 2023

Acknowledgements

I want to express my sincere gratitude to all those who have supported and contributed to completing this dissertation. Without their assistance, guidance, and encouragement, this work would not have been possible.

I am profoundly thankful to my advisor, Prof. Doctor Rui Paulo Pinto da Rocha, for his unwavering support, insightful guidance, and mentorship throughout this research. His expertise and dedication were instrumental in shaping this project and my academic journey.

I am also grateful to the members of my dissertation committee, Prof. Doctor Jorge Manuel Moreira de Campos Pereira Batista and Prof. Doctor Lino José Forte Marques, for their valuable feedback and constructive criticism that significantly improved the quality of this document.

I extend my thanks to the faculty and staff at the University of Coimbra for providing an enriching academic environment and valuable resources for my research, special thanks to Fábio André dos Santos Faria for helping in the process of testing the Jetson board powered by an alternative power source.

My appreciation goes to my family and friends for their unwavering encouragement and moral support. Their patience and understanding during the demanding phases of this project were essential in keeping me motivated. I want to give special thanks to my friends, *Naiara Simões*, *New New New New New New New Nicole Petra Kraemer Medina* and *Victor Kawazoe Bem*, for helping me in the execution of the experiments that were performed in this work.

In closing, I am grateful to all those mentioned above and to anyone else who has played a part, however small, in the successful completion of this academic document.

Thank you all for your contributions and support.

Resumo

Esta dissertação de mestrado tem como objetivo geral criar um espaço comum entre diferentes abordagens de mapeamento semântico. Para o efeito, é proposta uma *framework* espaciotemporal probabilística modular e versátil. Esta abordagem tem as seguintes vantagens: torna possível a fácil integração de novos métodos de deteção e classificação semânticos; é expansível, tornando o sistema mais adaptável às características e necessidades específicas de cada aplicação; é robusta na forma como lida com mudanças no ambiente e com a incerteza; é computacionalmente eficiente e escalável, tornando praticável o crescimento do mapa e suas representações quando o sistema é executado num dispositivo de *computação de borda* instalado no robô.

O *pipeline semântico* que corporiza o *framework* proposto torna possível o uso de detetores estado da arte pré-treinados para estimar características 3D primitivas de objetos detetados e posicioná-los num *mapa topológico probabilístico*. A *framework* também pode lidar com a adição de novas classes ao sistema sem precisar recriar mapas antigos ou retreinar o sistema inteiro devido à *classificação expansível*.

O *mapa semântico topológico* utilizado é leve tanto em espaço de armazenamento quanto em requisitos de processamento. A representação pode armazenar informações mais ricas em menos espaço se comparado com mapas semânticos convencionais baseados em grelhas de ocupação.

O sistema foi implementado e testado num robô móvel equipado com um *System On Module (SOM)* Jetson (Nvidia), usando um ambiente de contentores *Docker* que possibilita a sua replicabilidade e boa escalabilidade.

Abstract

This dissertation focuses on creating a common ground between different semantic mapping approaches. With this aim, a probabilistic *spatio-temporal* framework is proposed, which has several advantages: it is modular, to make possible the easy integration of new semantic detection and classification methods; it is expandable, to introduce more adaptability to the system through time; it is robust in the way it handles environment changes and uncertainties; it is computational efficient enough to deal with the growth of the map and its representations while executing locally on the robot, in an onboard *edge computing* device.

The proposed *semantic pipeline* makes possible the use of pre-trained popular state-of-the-art detectors to estimate primitive 3D geometric characteristics of detected objects and place them into a probabilistic *Topological Semantic Map*. The framework can also handle the addition of new classes to the system without the need to recreate old maps or retrain the entire system due to the *expandable classification*.

The *Topological Semantic Map* utilized is lightweight in both storage space and computational requirements. It can store richer information in less space than conventional grid semantic maps.

The system was deployed and tested on a mobile platform equipped with a Nvidia Jetson System On Module (SOM) using a *Docker* containerized environment to have better applicability and scalability.

"We cannot solve problems with the kind of thinking we employed when we came up with them."

— Albert Einstein

Contents

List of Acronyms

AABB	Axis Aligned Bounding Box
AMCL	Adaptive Monte Carlo Localization
AP	Average Precision
ARM	Advanced RISC Machines
BB	Bounding Box
BDM	Binned Depth Map
CPU	Central Processing Unit
CSPDarknet53	Cross Stage Partial Darknet53
CUDA	Compute Unified Device Architecture
cuDNN	CUDA Deep Neural Network
DDS	Data Distribution Service
FN	False Negative
FoV	Field of View
FP	False Positive
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HOG	Histogram of Oriented Gradients
IoU	Intersection over Union
IPC	Inter Process Communication
ISR	Institute of Systems and Robotics
L4T	Linux For Tegra
LED	Light Emitting Diode

LIDAR	Light Detection and Ranging
ML	Machine Learning
MLP	Multi Layer Perceptron
MRL	Mobile Robotics Lab
NMS	Non-Maximum Suppression
OS	Operating System
OvA	One-vs-All
PANet	Path Aggregation Network
PCL	Point Cloud Library
PDF	Probability Distribution Function
PRE	Precision
QoS	Quality of Service
RMSE	Root Mean Square Error
RoA	Region of Acquisition
RoE	Region of Exclusion
RoI	Region of Interest
ROS	Robot Operating System
R-CNN	Regions with Convolutional Neural Network
SDK	Software Development Kit
SIFT	Scale-Invariant Feature Transform
SLAM	Simultaneous Localization and Mapping
SMAP	Semantic Mapping
SOM	System On Module
SOR	Statistical Outlier Removal
SSD	Single Shot Detector
SURF	Speeded Up Robust Features
SVM	Support Vector Machine
TCP	Transmission Control Protocol

TF	Transform
TN	True Negative
TOPS	Tera Operations Per Second
TP	True Positive
TRT	Tensor RT
YOLO	You Only Look Once

List of Figures

List of Tables

List of Pseudocodes

1 Introduction

The last three decades have been marked by technological advances that were key in solving current problems in healthcare, space exploration, industrial development, and many others. Most of these achievements can be attributed to computers, due to the high availability of cheap computing power, and to robots, for their ability to work with high loads and operate in places with adverse conditions.

1.1 Context and Motivation

Despite many improvements in processing power and robotics autonomy, the robots of today are limited mostly because often they only can perform tasks in constrained environments or can only understand a limited set of low-level commands that are almost unique to each robot. To follow the tendencies proposed by industry 4.0 and become more popular in domestic applications [1], those robots need to operate in complex dynamic, and uncertain environments while interacting with people in a natural way.

Semantic mapping provides a way to relate high-level labels (*e.g.* “chair”, “toy”, “computer”, ...) with low-level sensor data in map structures. Applications using those structures should easily give robots the capability to share richer map features with less bandwidth (*e.g.* instead of sending a raw point cloud that describes a chair, the robot can send just a sparse reconstruction and the label "chair") and be capable of understanding high-level commands, expressed in a human-like way (*e.g. find(“table”)* or *go_to(“office”)* instead of *go_to(x=47.8,y=5.30,z=-169.2)* [2]. Additionally, the understanding of more abstract concepts is a step towards richer human-robot interactions, such as object manipulations between humans and robots using natural language.

The attribution of semantic information about the environment perceived by the robot is key to robots becoming present in domestic environments and working cooperatively with humans in industries, offices, and hospitals.

1.2 Problem Statement

Often, robots are unable to interpret data as humans do. They may only collect sensorial data and interpret it as points in space, which is enough information to navigate a workspace when represented in a robotic navigation map. However, it is not enough if the robot has to engage in natural and human-like interactions with users because it lacks meaningful information for humans about the *semantics* of objects and places.

Two important concepts to consider are *anchoring* and *ontology*. The first one is a well-known concept in robotics and is defined in [3] as:

> *"We call anchoring the process of creating and maintaining the correspondence between symbols and sensor data that refer to the same physical objects."*

Ontology, the second one, is an area of study in philosophy that follows a more abstract concept. In philosophy, the term was defined in [4] as:

> *"[...] ontology, which can be defined as the science of what is: of the various types and categories of objects and relations in all realms of being."*

A less abstract definition of ontology applied to computer science is defined in [5] as:

> *"A specification of a representational vocabulary for a shared domain of discourse", "definitions associate the names of entities in the universe of discourse (e.g., classes, relations, functions, or other objects) with human-readable text describing what the names are meant to denote is called an ontology"*

To explain what *Semantic Mapping* is, one first has to understand what "semantics" is. *Semantics* is the study of the set of characteristics that can be attributed to a thing to describe it. *Semantic Mapping* can be defined as the ability of the robot to perform *anchoring* by clustering features in sensorial data which are then used to detect and classify things (*i.e.* objects, places) in the environment using human-like categories (*i.e.* symbols), to further establish a relationship between the detected things, the robot map, and a conceptual map (Figure 1.1).

An example can be the semantics associated with the object shown in Figure 1.2. By looking at it, it is possible to relate its shape to a thing that holds liquids; and, combined

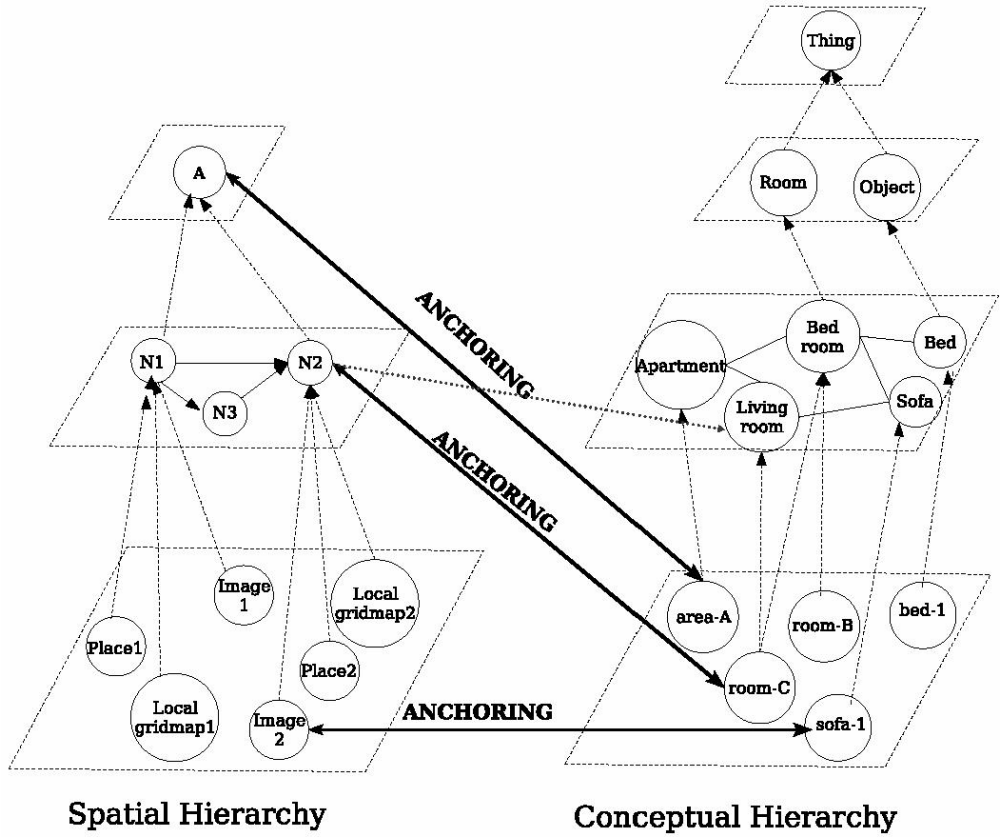

Figure 1.1: The spatial and semantic information hierarchies. On the left, spatial information is gathered by the robot sensors. On the right, is semantic information that models concepts in the domain and the relations between them. Image reproduced from [2].

with the label on the box, one gets enough information to determine that it is a milk box undoubtedly. Therefore, after determining its *semantics*, the object is no more for the robot than just a geometric volume located somewhere in the workspace, but also a human-like meaning is assigned to it, which embeds key information about possible interactions of the object with the human user and the environment. This arises when *e.g.* the user asks the robot "Please bring some milk to me"; or the robot reasons that if that object is found in some place, this is perhaps a kitchen or a dining room.

Figure 1.2: Milk box.

By using an *ontology* and building a *knowledge base* based on it, the semantic mapping concept can be expanded to incorporate even more information and improve the classification of objects. As an example, one can reutilize the milk box analogy; by knowing the shape of the object, it can be inferred that it must be a fluid container; consequently, we can increase the probability of the object being classified as a milk box or a juice box. One can also use ontology to reason about objects to decide where to search for them; for instance, a milk box requested by the user is more likely to be found in places like a kitchen or pantry; or even inside another object, like a fridge.

1.3 Objectives

The main goal of this dissertation is to develop a modular semantic mapping framework that can assign meaningful labels to the data collected by a robotic system and combine the results into a coherent map, to be later capable of performing tasks with richer *human-robot* interactions.

The specific objectives to be achieved by the framework are:

- Introducing a framework capable of detecting and extracting the geometry of objects in the surrounding environment, for later combining it to the robot map.

- To create a modular structure that can work with most of the popular *state-of-the-art* object detectors without the need to reshape the main code.
- To have expandable entity classification.
- Create a mechanism that can handle the uncertainty associated with temporal variations.
- To be lightweight enough to be deployable in an *edge computer*.
- To be implemented in a containerized environment and in a popular and recent middleware such as ROS[1] 2, in order to be available for further research and easily integrate into other systems.

In order to validate the developed framework, it should be integrated into a Pioneer P3-DX module robot driven by a SOM Jetson AGX Xavier[2]. The robotic platform is equipped with a ZED 2 Stereo Camera[3], and a Hokuyo URG-04LX Light Detection and Ranging (LIDAR).

1.4 Document Overview

This document is divided into 6 chapters each discussing one of the main aspects of the work. The first chapter is an introduction to the subject to be explored; it describes the motivation and the objectives of the work. The second chapter talks about the research on the topic and explains some key aspects of different *state-of-the-art* implementation. Chapter 3 discusses the functionality developed in this work; it proposes a semantic pipeline from *sampling* to the *semantic map* and explains the steps needed in the process while abstracting implementation details. Chapter 4 explains the details of the system implementation regarding sensors utilized, container architecture and relationship, ROS architecture, and the third-party tools utilized. Chapter 5 talks about the experiments conducted with the mobile platform P3-DX and its results. Finally, Chapter 6 highlights the main aspects addressed in the document and suggests some future improvements.

[1]ROS is an open source middleware developed by Open Robotics: https://www.openrobotics.org/.
[2]https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-series/
[3]https://www.stereolabs.com/zed-2/

2 Related Work and Fundamentals

The concept of semantic mapping has been explored a quite a long time. One of the first researchers was Benjamin Kuipers in 1978 with the paper "Modeling spatial knowledge" [6] where he tried to embed spatial knowledge in topological maps aiming for the use of concepts like "*you are here*", "*place*", "*path*", and "*go-to(place A, place B)*". In later contributions [7, 8], Kuipers determined a relationship between topological and metric data and related those with the control commands performed by the robot. This was the first effort in the area and it was mainly manual; it was a deterministic approach and did not introduce much information to the map.

In the early 2000s, the first confidence-based approach was proposed by Galindo [2] using topological maps to represent semantics, and metric maps to represent the environment. The relationship between the maps was achieved using *anchoring* (proposed by [3]) between links of two hierarchy graphs that relate spatial information (sensorial data) to conceptual knowledge. The ability to make decisions taking uncertainty into account became a baseline for most of the future works in the area.

Most of the future works used a confidence-based spatio-temporal accumulation method to ensure more stable results with less impact of false positives. This method utilizes a map represented by discrete cells arranged in a grid, in which each cell stores a probability of that cell to be occupied (*i.e.* 2D grid map). In this case, in addition to the occupancy value, each cell contains a vector of beliefs of each possible class [9, 10]. A similar approach was used in [11]. The authors used a vector like [9, 10], but with the addition of θ values to represent the complete pose in which the observation was acquired. Another slight variation was used in [12]. In this case, there is a grid map for each semantic class. A common downside of these approaches is the more extensive memory requirement to store the map, especially [12, 13], if compared with works based on topologic maps for semantic representation, like [7, 8, 11, 14]. Conceptually, these work in the same way as the ones based on a grid map, but have a more compact and efficient representation in terms of memory requirements.

The works [14, 15] follow the same idea as Galindo but with a more realistic representation, using a topologic map to represent the semantics and a probabilistic relational model that integrates commonsense knowledge about the environment (*i.e.* a *conceptual map*). This approach is capable of building new relations between concepts observed in the workspace and can improve the inference capability about the environment. For instance, if a computer is detected, the weights for classifying the place as a room or an office are increased. Despite the use of detected object information to better estimate the place, those approaches were scene-oriented and based on Support Vector Machine (SVM) classifiers.

The majority of articles focus on indoor environments, although ironically one of the first works in the semantic mapping research area was based on outdoor environments [7]. This aspect is fundamental to describe the main challenges related to the mapping task. In outdoor applications, the biggest challenges are to develop sensorial models robust enough to deal with adverse conditions (*i.e.* foliage, translucent surfaces), adapt the robotic tools (*e.g. mappings framework, object detection, etc.*) to work on a much larger scale, and develop highly efficient object detection/classification methods. In contrast, indoor applications are simpler in terms of the robustness of low-level system characteristics but tend to be more accurate. There are some examples of the combination of multiple sources of detection/classification into a more diverse or accurate result [11, 14, 12] (*i.e. Multi-Cue* detection/classification), and dynamic models that can learn conceptual relationships autonomously [14].

There are only a few examples of research in the robotics community that introduce the geometry of the objects in the map [1, 16, 13]. The use of object geometry is more common in the computer vision community, in works like [17, 18, 19, 20], mostly because of the high computational cost of the tasks which most mobile robots cannot afford (*e.g.* mesh segmentation, 3D object reconstruction).

There are a few open research problems in previous works about semantic mapping:

- There are no standards in the way labels are related to the point clouds or the map, *i.e.* there is a multitude of distinct map representations.
- As the implementations are mostly platform-specific and are not easily replicable in other circumstances, adding a new feature means doing almost all the work from the beginning.
- The lack of conventions makes difficult the adaptation to robotics of solutions from computer-vision applications.

Because of these problems, it is hard to benchmark different methods. Usually, the articles only present comparisons of the same method in different datasets or scenarios, but there is no indication of meaningful comparative benchmarks with similar methods.

	B. Kuipers [8]	C. Galindo [2]	R. Goeddel [9]	Y. Katsumata [10]	A. Pronobis [11]	N. Blodow [1]	N. Sünderhau [12]	A. Pronobis [14]	R. B. Rusu [13]	J. C. d. C. S. Fernandes [21]	N. Sünderhau [22]	D. Maturana [23]	This Work
Indoor Environment	X	X	X	X	X	X	X	X	X	X	X		X
Outdoor Environment	X											X	
Single-Cue Classification	X	X		X	X	X			X	X	X		X*
Multi-Cue Classification					X		X					X	X*
Object Detection				X		X	X	X	X	X	X	X	X*
Place Detection	X	X	X	X	X		X	X		X			X*
Expandable Classification					X		X						X
Semantic Grid Map			X	X	X	X	X	X	X	X	X	X	
Semantic Topological Map	X	X			X			X					X
Object Reconstruction						X			X		X	X	
Confidence-Based		X		X	X	X	X	X	X	X	X	X	X
Spatio-Temporal Coherence				X	X	X	X		X	X	X	X	X
Conceptual Map	X	X		X			X	X		X			X
Automatic Concept Building								X					

Table 2.1: Semantic mapping related works comparison. (*) Given the modularity of the system, these features are dependent on the detectors utilized in the application.

A comparison of the methods surveyed in this section is presented in Table (2.1). The table shows that features such as *Confidence-Based* and *Spatio-Temporal Coherence* are mandatory in *semantic mapping* models. Additionally, most of the works use *Single-Cue Classification* and use *Semantic Grid Maps*. Rare characteristics are *Expandable Classification* and *Automatic Concept Building*. This dissertation proposes a versatile modular

framework that is independent of a specific object detector and gives the user the ability to tune the computational cost and model performance by selecting different detectors.

2.1 Object Recognition

Object recognition is a field of computer vision that aims to identify objects in images. It is primarily composed of two main tasks: object detection and object classification. These two tasks are closely related but are essentially different, as described by Zheng Song in [24]:

> *"The object classification task aims to predict the existence of objects within images, whereas the object detection targets localizing the objects."*

In the early 2000s, the task of detecting objects in images where performed by region descriptors like Histogram of Oriented Gradients (HOG) and Scale-Invariant Feature Transform (SIFT). The first one, HOG [25], has fixed-size oriented square boxes as descriptors, and the orientation of the boxes is determined by the gradients around the key points. The second one, SIFT [26, 25], is more robust and is like an extension of HOG, it adds a final step that resamples local pixels into planes with multiple scales and orientations to better estimate the correct one.

Despite the robustness of SIFT in dealing with affine transformations in its patches and its faster version Speeded Up Robust Features (SURF) [26, 25], those methods were outperformed by non-blockwise region based techniques like Regions with Convolutional Neural Network (R-CNN) [27], Fast R-CNN [28] and Faster R-CNN [29].

The current state-of-the-art methods are Single Shot Detector (SSD), You Only Look Once (YOLO), and Retina Net. All three were inspired by the region proposal concept introduced by Girshick in [27], but each of them tries to improve the accuracy or latency in its manner. Being the SSD the best in terms of latency, RetinaNet the best in terms of accuracy, and YOLO as the best overall performance between lower latency and accuracy.

2.1.1 YOLO

YOLO is a state-of-the-art real-time object detector and classifier. It was initially proposed by Joseph Redmon in 2015 [30] and became one of the main methods used in many applications related to object detection. Over the years YOLOv2 and YOLOv3 [31] were published officially by the original creator, and versions v4 up to v8 were unofficial releases

published by the community. These subsequent versions of the deep neural network-based image detector proposed over the years improved their performance with small changes in its architecture and better integration with modern ML deployment tools.

YOLO became popular because of its fast inference time while maintaining reasonably good accuracy. It was achieved by combining the two steps of detection into a single network. The YOLOv3 object detection works by first creating a $S \times S$ grid with cells with equal sizes within the image and then creating 5 anchors for each cell, the center of the cell is the reference point of the anchors. Each anchor is the patch of pixels that are used for classification. Each anchor only can classify one object, *e.g.* for a grid with 5×10 cells, it can have at maximum $5 \times 10 \times 5 = 250$ objects detected. The parameters of the anchors have prior fixed values (p_w, p_h) and are defined in the training by the inverse of the cost function of each anchor parameter. The parameters are combined to form the final cost functions [31]:

$$\sigma(t_o) = Pr(object) * IOU(b, object) \quad (2.1) \qquad cost(x) = b_x + b_y + b_w + b_h + \sigma(t_o) \quad (2.2)$$

where b_x, b_y, b_w, b_h are the cost functions of the anchor parameters and σ is the sigmoid function. $\sigma(t_o)$ is the confidence score, and it is defined by the product of the object prediction confidence by the Intersection over Union (IoU). The parameters referred to above are illustrated in figure 2.1.

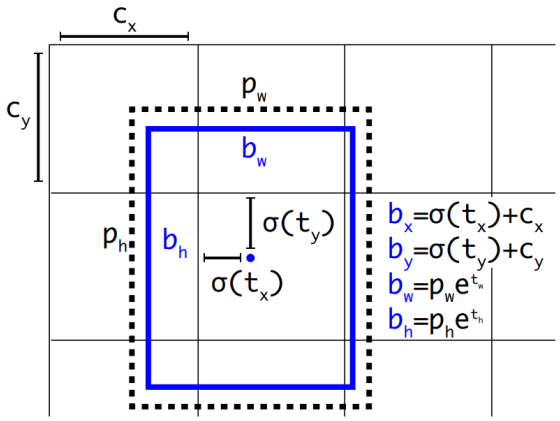

Figure 2.1: Bounding boxes with dimension priors and location prediction. Image reproduced from [32].

The YOLOv5, one of the most popular detectors of the YOLO family, introduces some changes to the architecture that make it more performant in both, accuracy and latency by using a Cross Stage Partial Darknet53 (CSPDarknet53) backbone [33]. It also improves

the localization of objects on different scales with the use of a Path Aggregation Network (PANet) (a network similar to RetinaNet) as a Neck. The YOLOv5 model was utilized in this dissertation because, in addition to its good architecture improvements, it also has out-of-the-box support to run in popular ML inference engines such as TensorRT and ONNX.

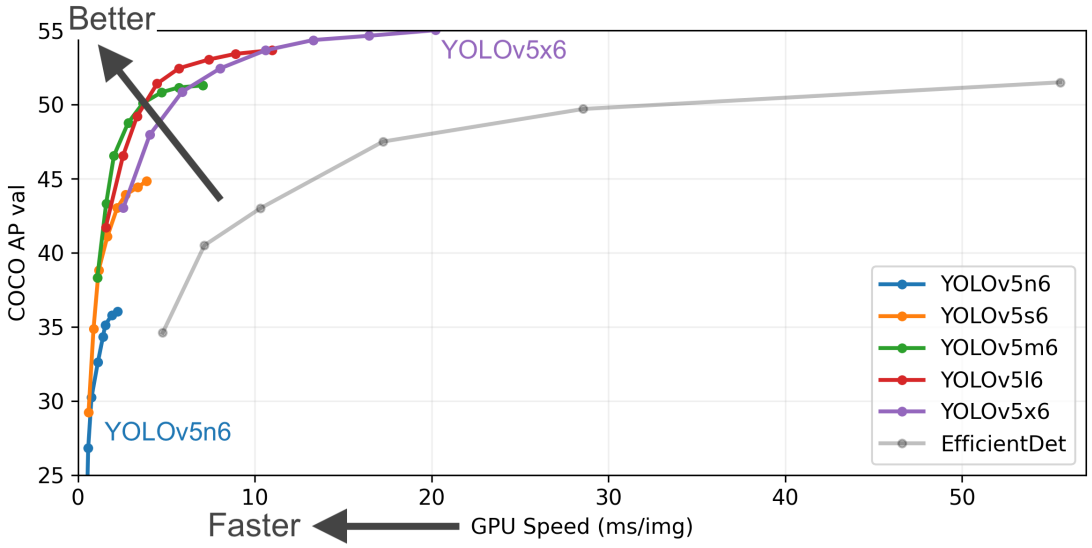

Figure 2.2: YOLOv5 models performance comparison in MS COCO dataset. Average Precision (AP). Image reproduced from [1].

2.2 Stacking Ensembles

Stacking [34] is a Machine Learning (ML) technique of the family of ensembles. An ensemble is the combination of the output of a group of independent classifiers into a new model that is trained with the results of the base classifiers. They are a powerful tool to achieve better classification performance and can be applied to almost any type of classification problem, with the downside of increased computational cost. Because of that, they are usually based on simple independent classifier models having low computational cost (*i.e.* weak learners).

Stacking is different from bagging [35], which is an ensemble technique used to better handle noisy datasets using multiple copies of the same classifier. And it is also different from boosting, which is an ensemble technique used to improve the performance of classifiers with copies of itself trained in different datasets [34]. Stacking uses heterogeneous classifiers and, by using a meta classifier, it can combine non-deterministic results (see Figure 2.3). These

[1] https://github.com/ultralytics/yolov5

two characteristics are very important for the modularity purpose of this work, because of the ability to combine different classification algorithms and consequently be more diverse if used with models that were trained with different datasets. The meta classifier can be any operation or algorithm capable of combining the inputs into a coherent output vector. It can be a simple normalized sum of the input vectors, or even another classification layer implemented using algorithms like SVM [36] and Multi Layer Perceptron (MLP) [37].

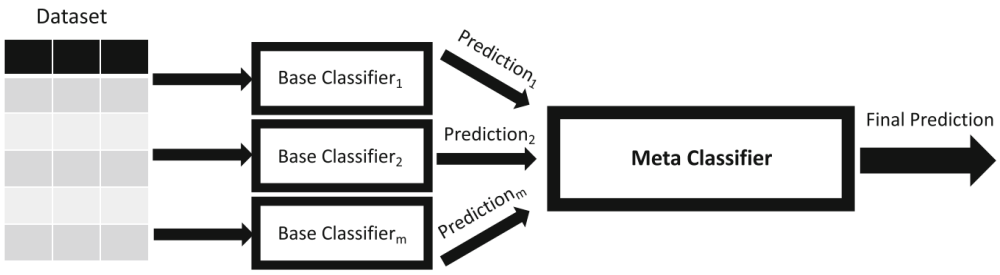

Figure 2.3: Stacking ensemble diagram. Image reproduced from [34].

3 Proposed Pipeline

This chapter proposes a scalable semantic mapping pipeline with parallel processing capabilities. In a general way, the process can be described in five major steps: *Sampling* (Section 3.1), where all the relevant data is acquired and packed; *Object Detection* (Section 3.2), where images are processed by one or many object detectors to extract labeled 2D BB; *Geometric Segmentation* (Section 3.3) in which a correlation between image points and cloud points is established and a 3D Axis Aligned Bounding Box (AABB) is extracted for each object detection; *Point Cloud Binning* (Section 3.4, page 23) which pre-processes the occlusion detection by computing a Binned Depth Map (BDM); and the *Data Association* (Section 3.5, page 25) component that assimilates the data processed by previous steps into a topological map. Figure 3.1 depicts an outline of the pipeline proposed in this thesis; the complete pipeline is depicted with more details in Appendix A (page 64).

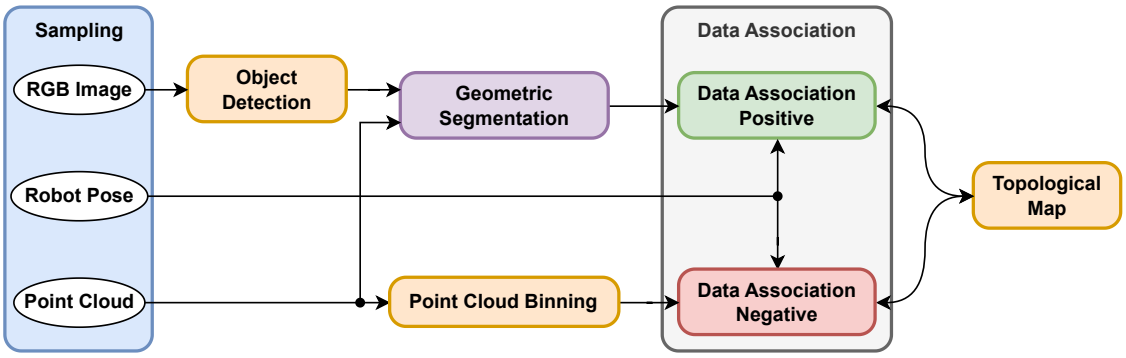

Figure 3.1: Semantic pipeline diagram.

The processing of the *Data Association* steps (Positive and Negative associations) occurs in parallel since they are independent of each other, and both Object Detection and Geometric Segmentation can have multiple instances working in parallel. Each object detector outputs a list of object propositions, and each object of this list can be processed in parallel in the Geometric Segmentation. Additionally, multiple object detectors with different sets of classes can be utilized. A diagram of the pipeline with just one *object detector* is presented in Figure 3.1.

3.1 Sampling

The sampling component is a data collection step utilized to group and synchronize the necessary data for the pipeline to work as intended. It expects as input the *Robot Pose* $\mathbf{p}$, a *RGB Image* $\mathcal{I}$, and a *RGB Point Cloud*. It outputs a structure with a combination of these 3 data types.

The *Robot Pose* is defined by the vector $\mathbf{p} = \begin{bmatrix} x & y & z & \psi & \phi & \theta \end{bmatrix}^T$ in relation to the world frame, it can be estimated by any *state-of-the-art* localization or Simultaneous Localization and Mapping (SLAM) algorithm using *range data* or *visual information* (*e.g.* SLAM Toolbox[1] or ORB-SLAM3 [40]).

The *RGB Point Cloud* is a set of $\begin{bmatrix} x & y & z & r & g & b \end{bmatrix}$ points in space relative to the sensor frame, it needs to be a "*projectable*" and "*organized*" point cloud. As defined in the documentation of the Point Cloud Library (PCL) [2]:

> "*A projectable point cloud dataset is the name given to point clouds that have a correlation according to a pinhole camera model between the (u,v) index of a point in the organized point cloud and the actual 3D values. This correlation can be expressed in its easiest form as* $u = f\ x/z$ *and* $v = f\ y/z$."

> "*An organized point cloud dataset is the name given to point clouds that resemble an organized image (or matrix)like structure, where the data is split into rows and columns.*"

3.2 Object Detection

In the object detection step, a set of 2D AABBs is generated to indicate the limits of the detected object (Figure 3.2). A normalized vector of the probability of the object belonging to each class is generated for each bounding box.

Any object detector capable of generating a 2D AABB and a probability vector, *e.g.* HOG-based detector, R-CNN, SSD, YOLO, *etc.*, can be used due to the framework flexibility. Most off-the-shelf options are designed to output the AABB alongside the label of the most probable class and its confidence. Although this imposes constraints, in most cases

[1] SLAM Toolbox is a graph-based SLAM based on *Open Karto* [38]. It is also the official SLAM package of ROS 2 [39]

[2] PCL is a reputable general purpose library to process point cloud data efficiently [41, 42]. It was first developed in ROS and later became so popular that it became an independent library to cover a much bigger range of applications.

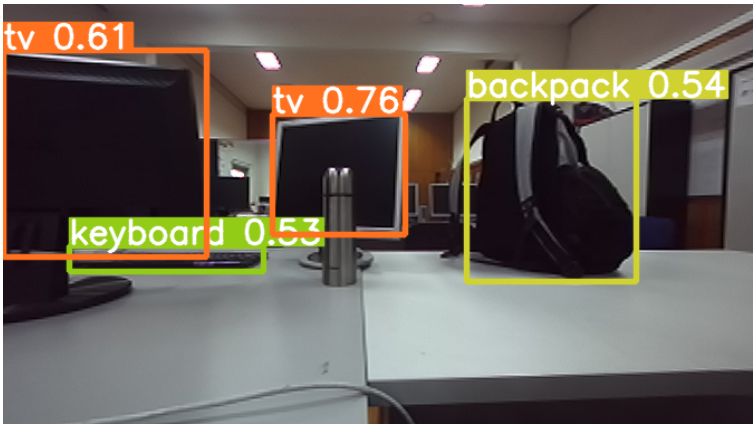

Figure 3.2: YOLOv5 detections with annotated labels.

a modification in the pipeline can adapt the network to work in the system. In cases in which the detector outputs only the most probable label, the last layer of classification can be removed and the values normalized to achieve the desired output.

Denoting as D the "*detections matrix*", where d is the number of detected objects (each line represents a single detection), $2D\ AABB_{[1\times4]}$ is a vector containing the minimums and maximums $[x, y]$ coordinates that define a 2D bounding box (top left corner and bottom right corner); and $\mathbf{x}_i$ as the "*vector of class labels*" of the classifier index i, in which the output vector has a length equal to the number of known classes N and follows properties of a Probability Distribution Function (PDF):

$$\mathbf{x}_i = \begin{bmatrix} x_1 & x_2 & \cdots & x_N \end{bmatrix}, \tag{3.1}$$

$$x_c \geq 0,\ \forall c \in \{1, N\}, \tag{3.2}$$

$$\sum_{c=1}^{N} x_c = 1, \tag{3.3}$$

$$D_{[d\times(N+4)]} = \begin{bmatrix} \left[\mathbf{x}_{[1\times N]}, 2D\ AABB_{[1\times4]}\right]_1 \\ \vdots \\ \left[\mathbf{x}_{[1\times N]}, 2D\ AABB_{[1\times4]}\right]_d \end{bmatrix}. \tag{3.4}$$

3.3 Geometric Segmentation

The *Geometric Segmentation* component expects as input an organized and projectable point cloud, as mentioned in Section 3.1, and a 2D AABB indicating the limits of a detected object in a 2D image. This component is composed of 7 steps of point cloud processing to get to the desired output which is the 3D AABB with its associated confidence. The 3D AABB includes the subset of points from the original point cloud that correspond to the detected

object in the scene. An illustration of this specific pipeline for geometric segmentation can be seen in Figure 3.3. The different steps of the pipeline are presented in the following subsections.

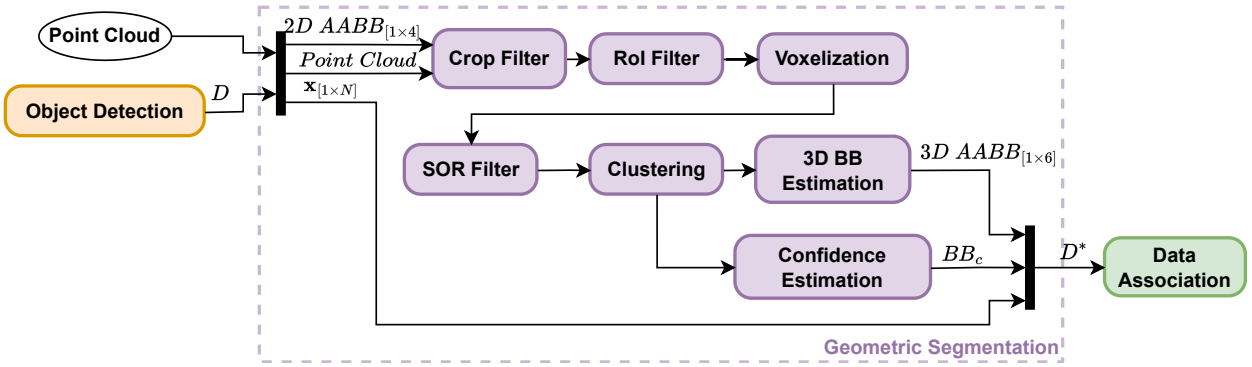

Figure 3.3: Geometric segmentation pipeline. $\mathbf{D}$ (Eq. 3.4) is the result of the classifier which is composed of 2D AABBs, and the classification vector $\mathbf{x}$. $\mathbf{D}^*$ (Eq. 3.5) is the composition of 3D AABBs, $\mathbf{x}$, and BB_c (Bounding Box (BB) confidence).

$$D^*_{[d\times(N+7)]} = \begin{bmatrix} \left[\mathbf{x}_{[1\times N]}, 3D\ AABB_{[1\times 6]}, BB_c\right]_1 \\ \vdots \\ \left[\mathbf{x}_{[1\times N]}, 3D\ AABB_{[1\times 6]}, BB_c\right]_d \end{bmatrix} \tag{3.5}$$

Crop Filter

The crop filter consists of segmenting the point cloud corresponding to the points of the detected object. Since the point cloud is organized, the processing is equivalent to cropping the relative 2D AABB of an image. The 2D AABB can be defined by two $[u, v]$ points, the points with minimum and maximum coordinates inside the box. Figure 3.4 illustrates the influence of this step.

Considering $p_1 = [u_1, v_1]$, $p_2 = [u_2, v_2]$ as the points with minimum and maximum coordinates respectively, and $\mathcal{I}$ as the image matrix, the matrix $\mathcal{I}_c$ is the image crop matrix:

$$\mathcal{I}_c \subseteq \mathcal{I}, \tag{3.6}$$

$$\mathcal{I}_c[u - u_1, v - v_1] = \mathcal{I}[u, v],\ \forall (u, v) \in [u_1..u_2], [v_1..v_2]. \tag{3.7}$$

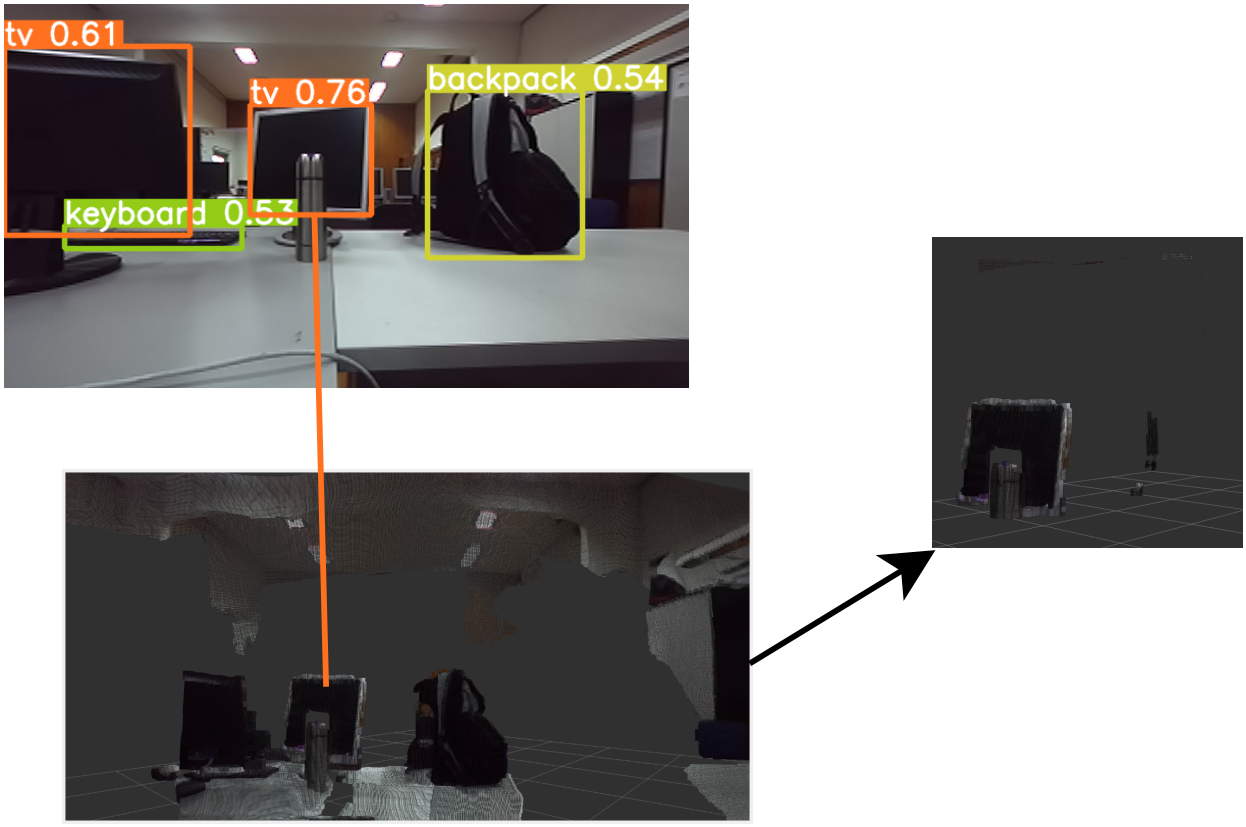

Figure 3.4: Crop filter comparison: image annotated with detections of an image detector (top), the equivalent point cloud (bottom), and the cropped point cloud corresponding to the bounding box of one of the detected objects (right). Figure 3.6 illustrates the next step of the pipeline.

Region of Interest (RoI) Filter

The RoI filter is responsible for filtering the region in which the precision of the acquired point cloud is acceptable (see Figure 3.5). This region is defined by 3 parameters, R_{min}, R_{max}, and the Field of View (FoV) of the sensor. Points outside this region are not completely invalid, but errors associated with their position are greatly increased and can degrade further steps of the geometric segmentation process. Figure 3.6 illustrates the influence of this step.

Voxelization

Voxelization is a downsampling process utilized to convert continuous 3D structures into discrete 3D grids. Continuous points are transformed into voxels through equation (3.8), the equivalent of points in a 3D grid, based on a parameter that dictates the size of the voxels S_{leaf} leaf size (see Figure 3.7).

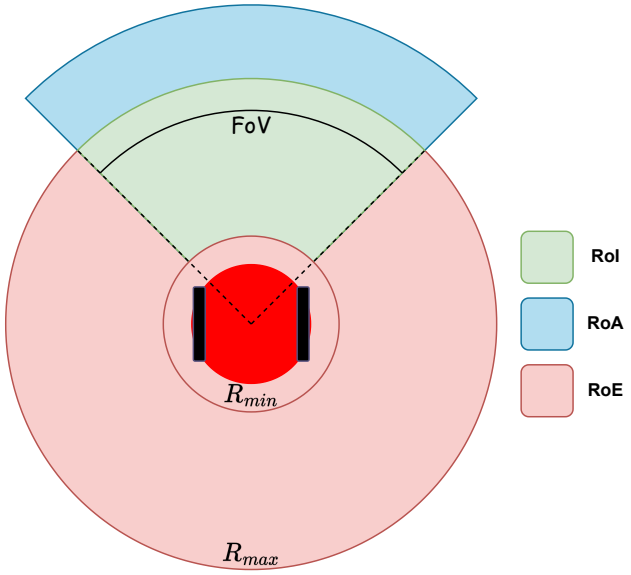

Figure 3.5: Illustration of the Region of Interest (RoI) and its relation with the Region of Acquisition (RoA) and the Region of Exclusion (RoE).

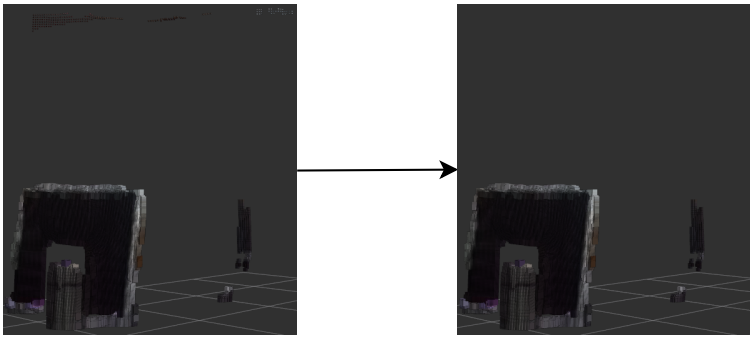

Figure 3.6: RoI filter comparison: left image shows the segmented point cloud from 3.4, right image shows the filtered point cloud. Figure 3.8 illustrates the next step of the pipeline.

$$point = (x, y, z) \in \mathbb{R}^3 \longrightarrow \mathbf{v} = [i, j, k] \in \mathbb{Z}^3 \tag{3.8}$$

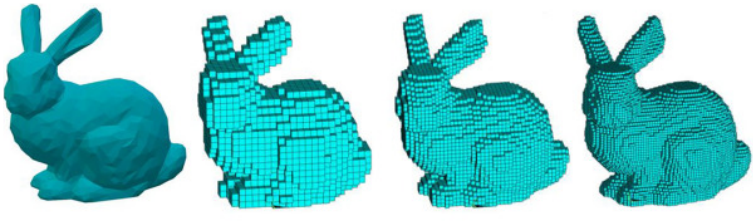

Figure 3.7: Illustration of the "*Stanford bunny*" voxelized using different values of S_{leaf}. Image reproduced from [43].

The voxelization of a point cloud equalizes the density of points through the space and removes excess data points in regions with higher densities. In sensors such as stereo cameras

and 3D LIDARs, objects closer to the sensor usually appear in the field of view with a higher number of points and a reduced number in marginal areas; this is a consequence of the perspective projection utilized in the sensor model. Figures 3.7 and 3.8 illustrate the influence of this step.

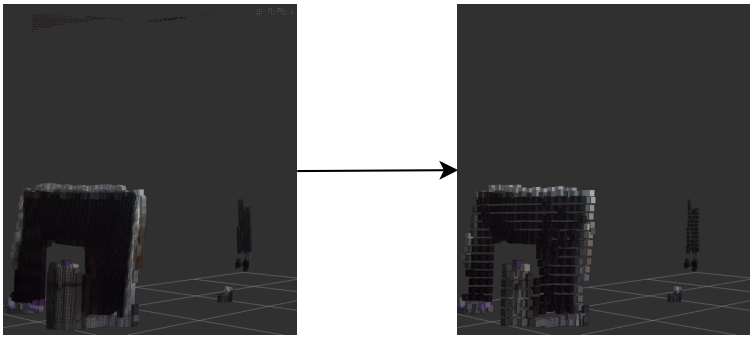

Figure 3.8: Voxelization comparison: left image shows the RoI filtered cloud from 3.6, right image shows the voxelized point cloud. Figure 3.10 illustrates the next step of the pipeline.

Statistical Outlier Removal (SOR) Filter

The SOR filter addresses the visual artifacts generated in projective point clouds. Those artifacts tend to occur in regions of transition between surfaces yielding abrupt depth variations. They look like a connection between the surfaces but they are actually outliers that can degrade the model (see Figure 3.9).

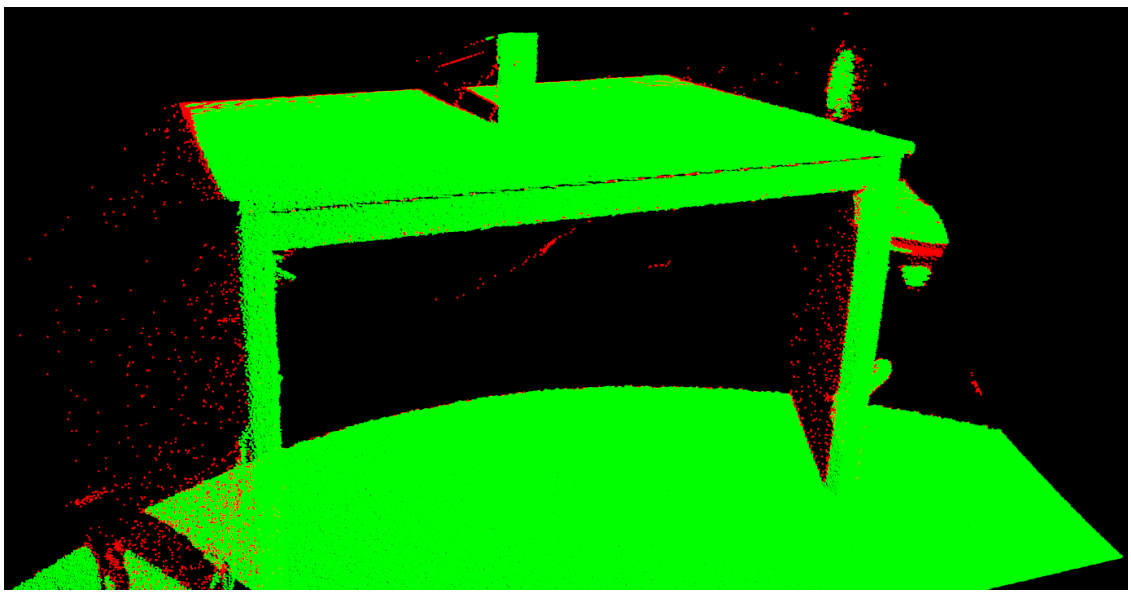

Figure 3.9: Illustration of the Statistical Outlier Removal (SOR): inliers are depicted in green and outliers are depicted in red. The image shows an office chair alongside a table with a small box over it. Illustration based on the table_scene_lms400 dataset[3].

This filter aims to remove the image artifacts by estimating a Gaussian distribution for the neighborhood of each point of the cloud and removing values with a large standard deviation multiplier. Figure 3.10 illustrates the influence of this step. The Pseudocode 3.1 describes the process.

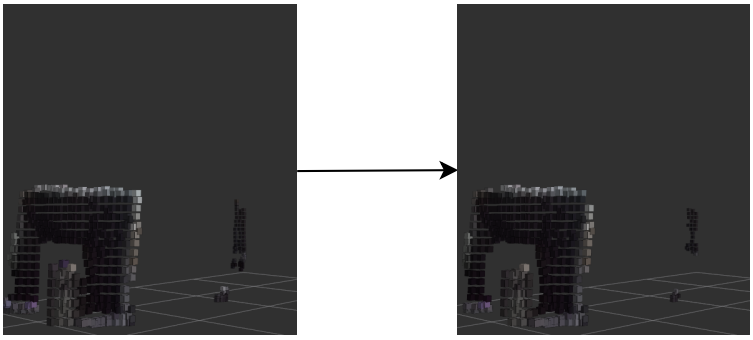

Figure 3.10: Statistical Outlier Removal (SOR) filter comparison: left image shows the voxelized point cloud from 3.8, right image shows the filtered point cloud. Figure 3.12 illustrates the next step of the pipeline.

Pseudocode 3.1: Statistical outlier removal filter

```
// mean_k -> Number of neighbor points to consider
// std_multiplier -> Max standard deviation value for a point to be considered inlier
for each point in point_cloud:
    Get mean_k neighbor points;
    Estimate a Gaussian distribution of the neighbors;
    for each neighbor in neighbors:
        Compute distance from neighbor to point;
        if distance greather than (std_multiplier·σ):
            Remove neighbor from point_cloud; // Outlier
```

Clustering Filter

Clustering is the process of arranging multiple sampled points into groups based on given a metric. If the Euclidean distance is chosen as a metric ($E_d = \sqrt{x^2 + y^2 + z^2}$), it can be utilized to split the point cloud into multiple smaller ones, in which the biggest cloud generated represents the most relevant one [44]. An example of clustering is depicted in Figure 3.11. The clusters are generated based on two parameters: the minimum amount of points to be considered a cluster ($cluster_{min}$), and the distance tolerance between points to be considered in the same cluster ($cluster_{tol}$).

Because of the previous steps of filtering, Box Filter (Subsection Crop Filter) and RoI Filter (Subection Region of Interest (RoI) Filter), the point cloud at this stage of the pipeline

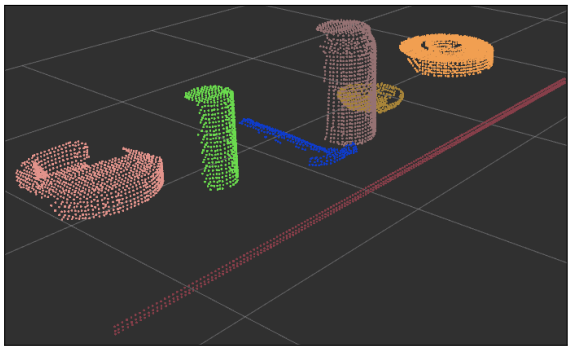

Figure 3.11: Point cloud composed of points of multiple objects sectioned into clusters represented by different colors. Image reproduced from [45].

represents a cloud composed of small patches of points in which the majority of the points come from the same desired object. Selecting the biggest cluster is equivalent to selecting the points most likely to belong to the detected object. Figure 3.12 illustrates the influence of this step.

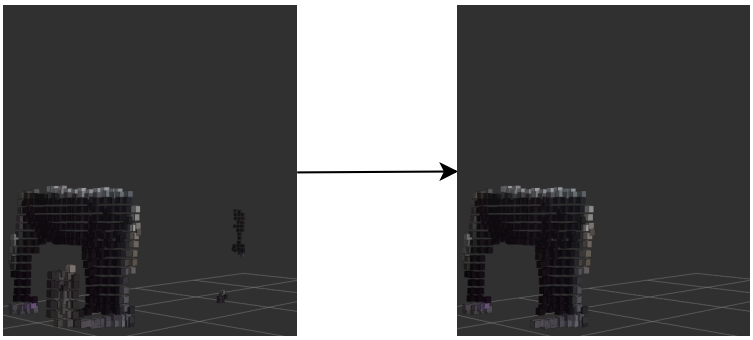

Figure 3.12: Clustering comparison: left image shows the SOR filtered point cloud from 3.10, right image shows the selected cluster. Figure 3.13 illustrates the next step of the pipeline.

3D Axis Aligned Bounding Box (AABB) Estimation

The 3D AABB represents the smallest cuboid, aligned with the world frame axes, that contains a group of points. It can be defined by only two 3D points, the minimum, and maximum $[x, y, z]$ of the sampled points. This process is described in detail by the Pseudocode 3.2. Figure 3.13 illustrates the influence of this step.

Pseudocode 3.2: AABB estimation

```
min = random point of point_cloud;
max = random point of point_cloud;
transformed_point_cloud = transform(point_cloud);
for each point in transformed_point_cloud:
    if min.x greater than point.x:
        min.x = point.x;
    if point.x greater than max.x:
        max.x = point.x;
    // Repeat for y and z
centroid = (min + max) / 2;
```

The transformation referred to in the Pseudocode 3.2 is the coordinate transformation from the sensor frame to the world frame. It can be described through equation (3.9):

$$\boldsymbol{P}^{world} = {}^{world}\boldsymbol{T}_{robot} \times {}^{robot}\boldsymbol{T}_{sensor} \times \boldsymbol{P}^{sensor}. \tag{3.9}$$

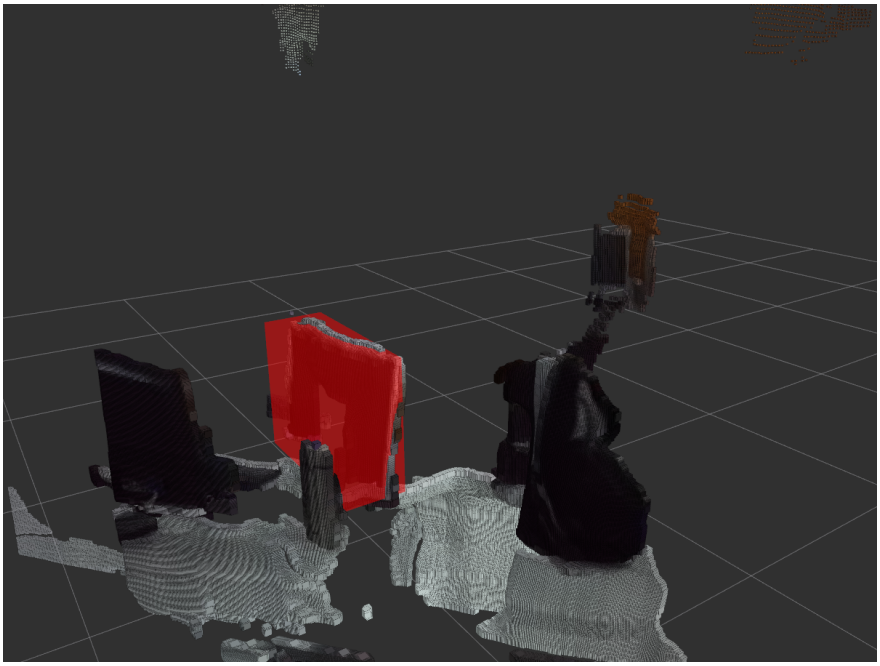

Figure 3.13: AABB illustration: The red box shows the estimation of the object volume in the initial point cloud. Figure 3.12 illustrates the previous step of the pipeline.

Confidence Estimation

A confidence value is generated based on the distributions of the points of the point cloud. Due to the nature of point clouds generated by sensors, such as stereo cameras and 3D LIDARs, the farther the points are from the sensor less accurate they are. The confidence response is defined by 3 parameters, the minimum and maximum possible distances R_{min},

R_{max} (see Subsection Region of Interest (RoI) Filter), and a threshold in which the confidence suffers no penalty (obj_ths). Pseudocode 3.3 illustrates the process:

Pseudocode 3.3: Confidence estimation

```
// R_min , R_max - minimum and maximum distances of points in point cloud
closest = random point of point_cloud;
farthest = random point of point_cloud;
for each point in point_cloud:
    closest_distance = compute_distance(closest);
    farthest_distance = compute_distance(farthest);
    point_distance = compute_distance(point);
    if point_distance greather than farthest_distance:
        farthest = point;
    if point_distance less than closest_distance:
        closest = point;
num = farthest - closest;
den = R_max - R_min;
if num less than or equal obj_ths:
    confidence = 1;
else:
    confidence = (den-(num-obj_ths))/den;
```

The parameters R_{min} and R_{max} are defined in Subsection Region of Interest (RoI) Filter. Figure 3.14 shows the response curve of this function

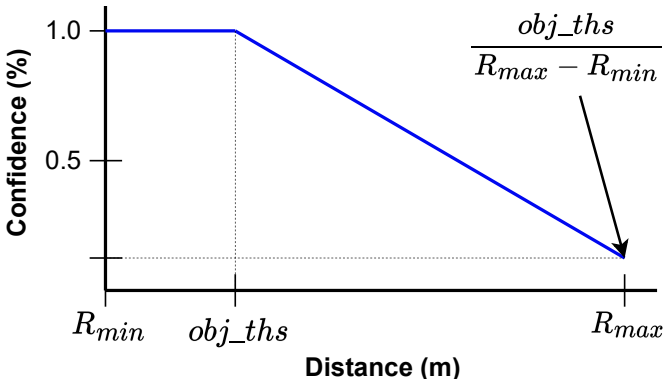

Figure 3.14: Confidence values as a function of the distance between the closest and farthest point of the cloud.

3.4 Binned Depth Map

The depth map is a representation of the distances from the camera frame to the solids in the scene. It is usually represented as a matrix of distances with each cell representing the distance related to the pixel with the same $[u, v]$ indexes in the image. In the binned

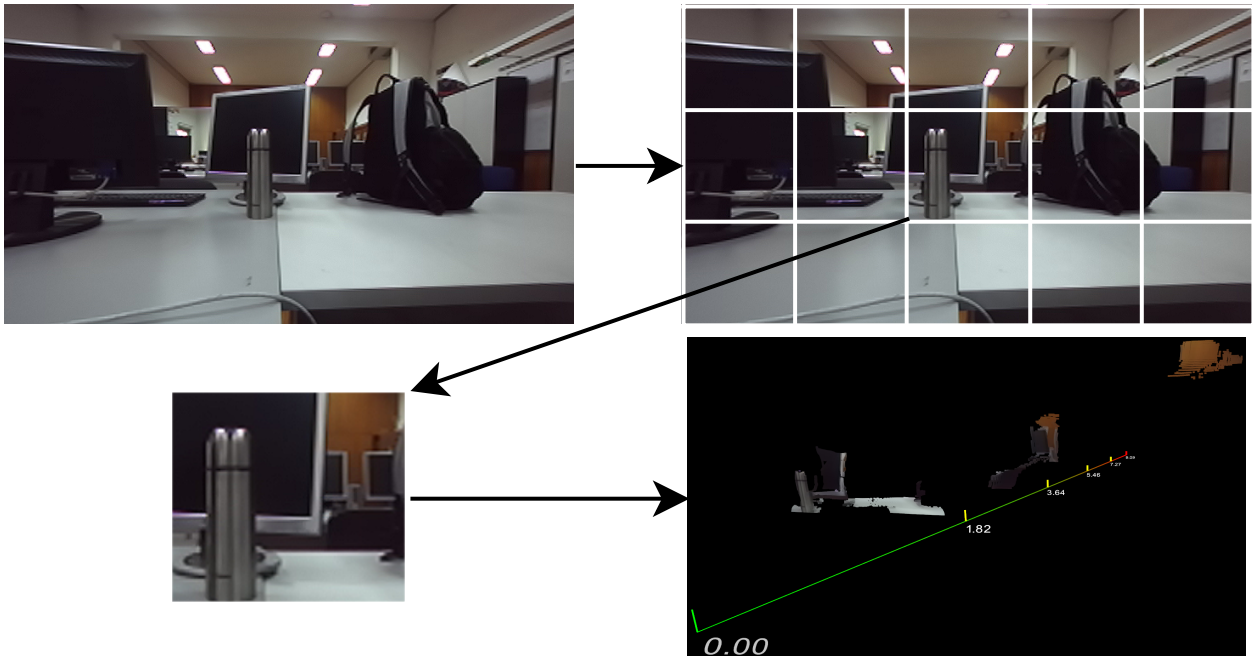

Figure 3.15: Illustration of the depth map binning process. The last image shows the cloud points associated with the segmented region of the *Point Cloud*.

depth map, each cell represents a panel with a group of distances related to multiple pixels in an image.

A binned depth map is used to lower the number of computations necessary to process the occlusions in the scene. The use of the binned depth map makes possible a more efficient occlusion processing by pre-computing the most intensive operations of the occlusion step (Section 3.5) in parallel with the *Geometric Segmentation* (Section 3.3). The parallelism is possible due to the independence of other steps of the pipeline, as it only requires an ordered *Point Cloud*.

The binned depth map is a simpler representation of the *Point Cloud* that enables less computationally expensive occlusion detection. It is generated based on ordered *Point Cloud* points, in which the size of the bins, defined by 2 xyz points, is determined by smaller square segments of the cloud. As shown in Figure 3.15, the image is split into smaller regions using a grid with cells of equal $[s_u, s_v]$ sizes, and further, those regions are related to the point of the cloud and a AABB of each region is extracted. Figure 3.16 shows a visual representation of the BDM.

It is important to note that due to the projective nature of the data, this process can produce boxes that have overlapping regions with each other. An illustration of this phenomenon is presented in Figure 3.17.

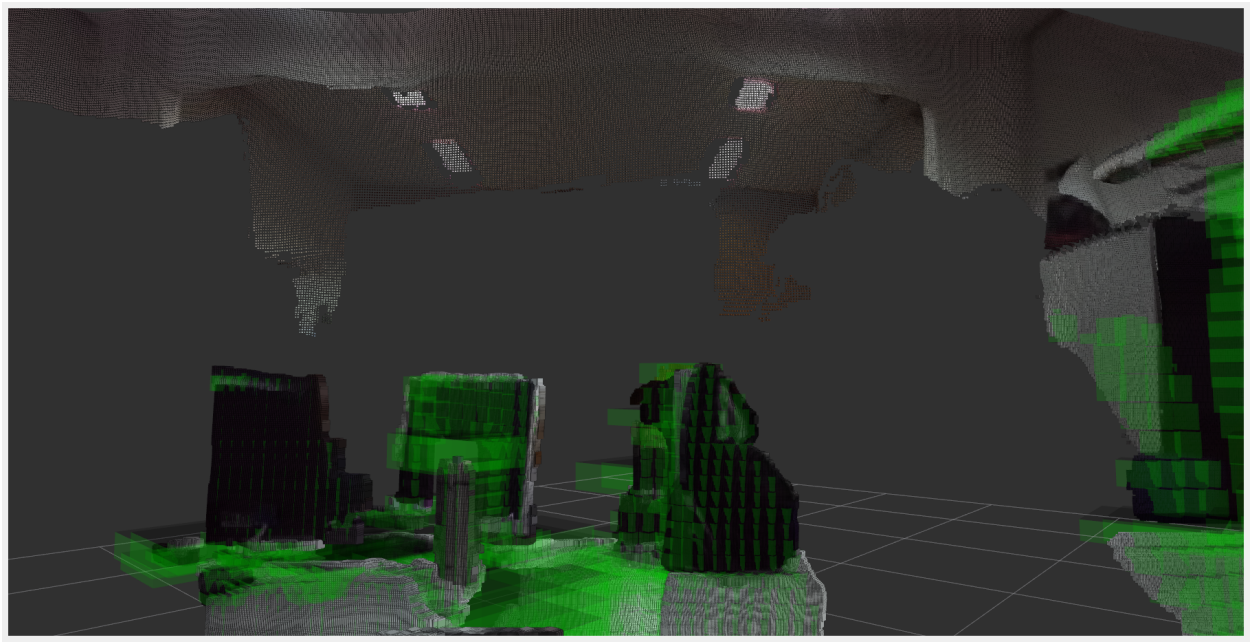

Figure 3.16: Binned depth map visual representation.

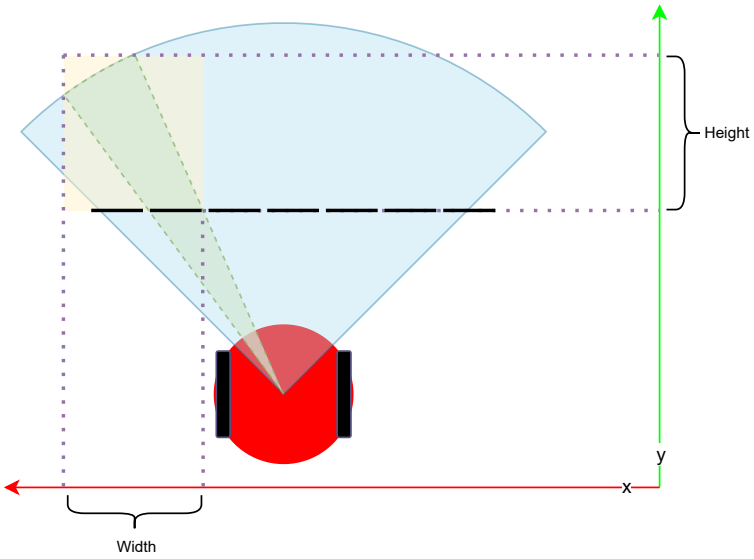

Figure 3.17: Projection of a region related to a bin. The black stripes represent the bin regions from a top-down view, the blue cone is the FoV, the green cone is the projection related to one bin, and the yellow square is the AABB of an arbitrary bin. The yellow square has a width that is bigger than the width of the bins because of the perspective projection.

3.5 Data Association

This Section will approach how the observations are processed, selected, and combined to incorporate spatio-temporal semantic information into a coherent topological map. It starts by introducing the topologic map and its data structures in Subsection 3.5.1. Subsection 3.5.2 presents the visibility histogram, which was utilized to mitigate object vanishing due to

occlusion. The combination of multiple detections and the update of the object's semantic values is discussed in Subsection 3.5.3. Finally, Subsections 3.5.4 and 3.5.5 explain how the mechanisms presented in this Section are logically combined to create the map using positive and negative observations.

3.5.1 Topological Map and Vertex Structure

The topological map was chosen to represent the system's semantics because of its coarser granularity, if compared to a typical grid map, which leads to less computational cost. Despite using a topological map, the system still needs the grid map to retain metric information. The vertices of the topological map store the semantic payload alongside the relationship with the spatial location of the objects in the grid map (Pseudocode 4).

As stated in "*Topological Mapping and Navigation in Real-World Environments*" [46]:

> "*A constructed topological map is a graph-like abstraction of large-scale space that represents the world as two types of areas: places, where qualitatively distinct decisions are presented to the robot, and paths, which are connections between places.*"

In terms of the graph structure. The vertices represent the landmarks and the connections represent the path that links the landmarks. Each vertex is composed of metric information which relates it to the grid map. Additionally, each vertex has a semantic payload that stores the semantics associated with this node. The structure of the topological map objects is described in Pseudocode 4.

Pseudocode 3.4: Topological map structures

```
vertex = {
    index: // unique integer
    position: // 3d_point float[3](x, y, z)
    this_thing: // thing object
    related_things: // list of thing objects
}
```

```
thing = {
    object_id: // unique integer
    type: // enum (OBJECT - 0, LOCATION - 1)
    histogram: // histogram object
    position: // 3d_point float[3](x, y, z)
    position_confidence: // float
    AABB: // tuple (3d_point min, 3d_point max)
    class_probabilities: // Probability hash map
}
```

Vertices are composed of 4 attributes, *index* (identifier number), *position* (3D point in space), *this_thing* (thing object), and *related_things* (list of thing objects). The first two are the minimum necessary to create a graph and store the metric information of the vertice; the last two are based on the *thing* structure and they represent the semantic payload.

The *thing* structure is a generic data structure utilized to describe semantic entities. It can be of 2 types (*OBJECT* - 0, *LOCATION* - 1). Vertices are always of the type *LOCATION* because they already describe the metric aspects of a place. *Things* of type *OBJECT* are stored in the *related_things* list and are always related to a unique vertex within a fixed vertex distance (V_d).

Things stores the position of the center point of the entity, its position confidence, the AABB, a visibility histogram (Subsection 3.5.2), and a hash map of all classes probabilities. A hash map is utilized, instead of a simple vector, to simplify the process of adding new classes to the data structure without the need to keep track of order.

Topologic Map Construction

The topologic map construction is based on the motion of the robot and the position of detected objects. New vertices and edges are added based on the distance from the robot to the nearest map vertex. If an object is detected at a position in space with a distance from a vertex greater than $2 \times V_d$ a support vertex will be created.

To ensure map consistency, the following conditions have to be satisfied:

1. Distance between any two vertices should be less than $2 \times V_d$.
2. Distance from an object to a vertex should be less than V_d.
3. Edges should have a max length of $V_d \times E_f$.

V_d and E_f denotes *Vertex Distance* and *Edge Factor*. Although both are tunable parameters, the E_f should be contained in the interval $]0, 1[$. $V_D \times E_f$ defines the minimum value acceptable for a new edge to be created.

3.5.2 Visibility Histogram

The visibility histogram is a polar histogram structure that maps the acquisition angle of the observations collected through time for each object. It is used to try to mitigate the vanishing of objects from the map due to occlusion. Each bin stores the log-odds of an observation occurring from that direction. Additive observations increase the value l, and subtractive observations decrease it l. The log odds function and its inverse are expressed through equations (3.10) and (3.11), where $p(x)$ is the probability of the object being visible.

$$\text{Log-Odds}: \; l(x) = ln\left(\frac{p(x)}{1-p(x)}\right) \quad (3.10) \qquad \text{Log-Odds}^{-1}: \; p(x) = 1 - \frac{1}{1+e^{l(x)}} \quad (3.11)$$

The bin values can be interpreted as follows:

- $(p(x) > 0.5)$ or $(l(x) > 0) \rightarrow$ the object should be visible.
- $(p(x) = 0.5)$ or $(l(x) = 0) \rightarrow$ undefined.
- $(p(x) < 0.5)$ or $(l(x) < 0) \rightarrow$ the object should not be visible.

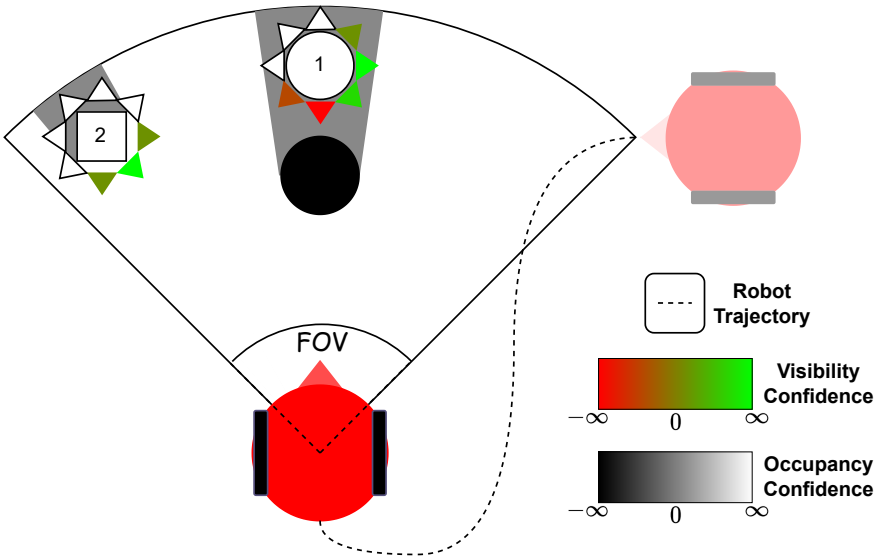

Figure 3.18: Representation of the visibility histogram of two objects. The robot has two objects inside his FoV but only one is visible at the current instant (object nr. 2), the other one is occluded by a column (object nr. 1). The triangles around the objects represent the bins of the histogram with colors ranging from red to green, with the white ones being undefined bins $(p(x) = 0.5)$ or $(l(x) = 0)$. The values shown in the figure are log-odds.

Figure 3.18 illustrates a scene where the robot collected observations while navigating through the presented trajectory. Based only on the current viewpoint of the robot, object nr. 1 should be deleted after a few seconds because it cannot be detected by the robot. Alternatively, by using the visibility histogram generated through the trajectory, the robot should be capable of interpreting this case as an occluded viewpoint and attenuate the probability decay to avoid removing the object from the map because it knows that the object was detected previously from a different viewpoint. Even if an object is considered occluded, a small decay factor is applied to account for temporal uncertainties and remove some false positives.

Histogram Update

The values of the histogram are updated based on the angle from where the object was observed ϕ_o and its distance d (see Figure 3.19). The value to be added to the bin as δ_b is defined in equation (3.12), where δ_{base} is the maximum desired update value.

$$\delta_b(d) = \frac{\delta_{base}}{\frac{d}{2} + 1} \tag{3.12}$$

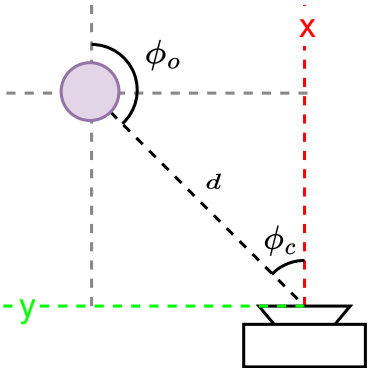

Figure 3.19: Representation of the angle of observation from the camera's perspective ϕ_c, and from the object's perspective ϕ_o.

Collected observations can be jittery due to factors such as platform movement, light conditions, and detector jitter. To minimize this issue, an iterative smoothening filter distributes newly acquired observation confidence through multiple bins instead of just the closest one. Its behavior is described below.

Consider an observation that should be assigned to an arbitrary bin h at a sample k as $\mathbf{b}_{h,k}$, a filter length as l_f, and a weight vector $\mathbf{w}_{1\times l_f}$:

$$\mathbf{b}_{h+j,1:k} = \mathbf{b}_{h+j,1:k-1} + \frac{1}{\sum \mathbf{w}_{1\times l_f}} \cdot \mathbf{w}\left[1 + \frac{l_f}{2} + j\right] \cdot \delta_b(d) \qquad \forall j \in \left[-\frac{l_f}{2} \,..\, \frac{l_f}{2}\right]. \tag{3.13}$$

To ensure a controlled convergence time and avoid numerical problems, a clamping step must be used to contain the log-odds value between known limits l_{max} and l_{min}. Consequently $l_{min} \leq \mathbf{b}_{h+j,1:k} \leq l_{max}$.

3.5.3 Semantic Update

This Subsection discusses how to perform an expandable classification, which aggregates the classification of different detectors, using a stacking ensemble configuration, and how

it is accumulated over time with a Bayesian filter [47] to incorporate prior and temporal knowledge.

This Subsection is mostly inspired by the *Expandable Classification* done by N. Sünderhau in [12] (Equations 3.14, 3.15, 3.16, 3.17, 3.18, 3.19). In his work, only one classifier was used, an AlexNet [48] network trained on Places205 dataset [49]. Conversely, this work aims to use multiple generic classifiers and differs in the prediction step of the Bayesian filter. Each of those classifiers was trained in a particular dataset and has its own set of known classes, consequently, they are only capable of evaluating a pre-defined set of classes, also known as closed-set classifiers.

Expandable Classification

The classification problem is commonly implemented using a closed-set assumption. They are trained in a fixed number of classes and are never presented with unknown classes at training time. This limited set has a significant impact on the ability of the system to adapt to new environments and is more susceptible to misclassification of unknown object classes.

In contrast, the open-set classifiers can reject unknown classes instead of misclassifying them. Closed-set classifiers can be extended to become open-set ones [50]. This can be accomplished by using a stacking ensemble (see Section 2.2, page 11) to create a meta classifier capable of combining the results of multiple classifiers that were trained on different class subjects. This meta-classifier can be as simple as the normalized sum of the input classification vectors into an expanded vector, or it can be more complex and use a ML algorithm to better fit and tune the expected results.

If using a ML algorithm, the meta-classifier learns how to combine the results of the classifiers into a bigger set of classes and improve the accuracy of repeated ones. The meta classifier is trained in an One-vs-All (OvA) arrangement using just a small training set. For instance, classifiers 1 and 2 are trained to classify objects with classes [A, B, C] and [A, B, D, E] respectively. The meta classifier should be able to identify classes [A, B, C, D, E] and possibly improve the precision of classes A and B.

The combined set $\hat{\mathbf{x}}$ is the combination of the classes of the M classifiers and it needs to abide by the following conditions:

- It must have independent classes.
- It cannot have repeated classes. Repeated classes should be combined into a single cell of the combined set $\hat{\mathbf{x}}$.

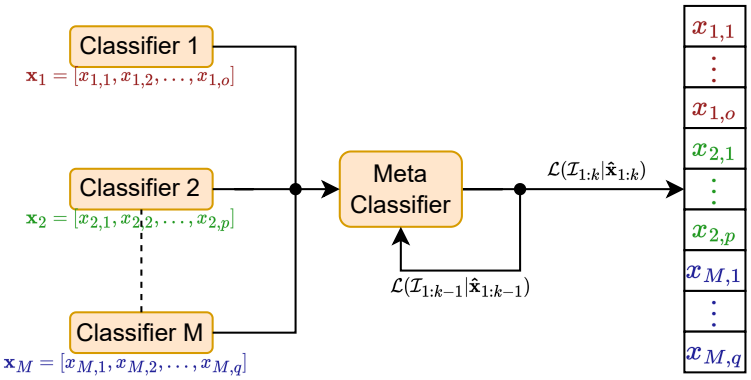

Figure 3.20: Stacking classification. $\hat{\mathbf{x}}_1$, $\hat{\mathbf{x}}_2$, $\cdots$, $\hat{\mathbf{x}}_M$ are the sets of classes of each classifier. Each classifier has its own set of classes it has been trained on. $\hat{\mathbf{x}}$ is the combined set and $\mathcal{L}(\mathcal{I}_\mathbf{k}|\hat{\mathbf{x}}_\mathbf{k})$ is the combined likelihood.

- The combined likelihood must be normalized to distribute the probabilities over the newly added classes and preserve the properties imposed by equations (3.2) and (3.3).

The combined set is defined in equation (3.14); An illustrative example is shown in Figure 3.21. This figure illustrates a case in which 3 classifiers have a common class, and that class is assigned to cell 3 of $\hat{\mathbf{x}}$. The value of cell 3 will be obtained by any operation that combines the 3 values (i.e. sum, product, weighted sum).

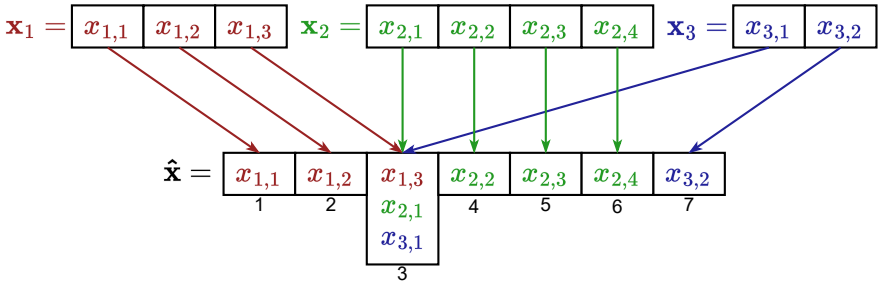

Figure 3.21: Combined set $\hat{\mathbf{x}}$ example. Cell 3 of the $\hat{\mathbf{x}}$ is the combination of the values $x_{1,3}$, $x_{2,1}$, $x_{3,1}$, that represents the common class known by the 3 classifiers.

$$\hat{\mathbf{x}} = \big(\underbrace{[x_{1,1}, \ldots, x_{1,o}]}_{\mathbf{x}_1}, \underbrace{[x_{2,1}, \ldots, x_{2,p}]}_{\mathbf{x}_2}, \underbrace{[x_{M,1}, \ldots, x_{M,q}]}_{\mathbf{x}_M}\big) \tag{3.14}$$

Taking Figure 3.20 as an illustration reference. Let $p(\mathbf{x}_i|\mathcal{I}_k)$ be the probability distribution, in which $\mathbf{x}_i$ represents the set of known classes of a classifier i, and $\mathcal{I}_k$ is the presented image at sample k. The likelihood of a classifier will be:

$$\mathcal{L}(\mathcal{I}_k|\hat{\mathbf{x}}) = \big[p(\mathbf{x}_1|\mathcal{I}_k), \ldots, p(\mathbf{x}_M|\mathcal{I}_k)\big] \tag{3.15}$$

Then, the combined likelihood can be obtained using equations (3.14) and (3.15):

$$\mathcal{L}(\mathcal{I}_k|\hat{\mathbf{x}}_k) = \big(p(x_{1,1}|\mathcal{I}_k), \ldots, p(x_{1,o}|\mathcal{I}_k), p(x_{2,1}|\mathcal{I}_k), \ldots, p(x_{2,p}|\mathcal{I}_k), \ldots, p(x_{M,1}|\mathcal{I}_k), \ldots, p(x_{M,q}|\mathcal{I}_k)\big) \tag{3.16}$$

Bayesian Filter

The output beliefs of classifiers can vary over time, even in an almost static environment, due to model imperfections. Stable outputs with low false positive rates are key to having a consistent representation of the environment. Because of that, a Bayesian filter is utilized to incorporate prior and temporal knowledge into the update.

As described in [51], a Bayesian smoother is:

"[...] a class of methods that can be used for estimating the state of a time-varying system which is indirectly observed through noisy measurements."

This filter is implemented in two steps, *prediction* and *update*, and its use can smooth the beliefs of classifiers over time by combining previous results with the current one into a more coherent, stable output with temporal knowledge. It is important to note that this method assumes that the next state is dependent only upon the current state (*i.e.* first-order Markov assumption).

The combination of the *prediction* and *update* steps can estimate the probability distribution $p(\hat{\mathbf{x}}_k|\mathcal{I}_{1:k})$ over all possible labels $\hat{\mathbf{x}}_k$, given the obeserved images $\mathcal{I}_{1:k}$.

The *prediction* step of the filter is the process presented in Expandable Classification in which its output is $\mathcal{L}(\mathcal{I}_k|\hat{x}_k)$ (Eq. 3.16). The *update* is the combination of the prior knowledge $p(\hat{\mathbf{x}})$, the current predicted value $\mathcal{L}(\mathcal{I}_k|\hat{\mathbf{x}}_k)$ (Eq. 3.16), and the result of a previous iteration $p(\hat{\mathbf{x}}_{k-1}|\mathcal{I}_{k-1})$, which results in equation (3.17).

$$p(\hat{\mathbf{x}}_k|\mathcal{I}_k) = \underbrace{p(\hat{\mathbf{x}})}_{prior\ \ knowledge} \cdot \overbrace{\mathcal{L}(\mathcal{I}_k|\hat{\mathbf{x}}_k)}^{prediction} \cdot \underbrace{p(\hat{\mathbf{x}}_{k-1}|\mathcal{I}_{k-1})}_{previous\ \ result} \tag{3.17}$$

The prior knowledge can be introduced by an external factor that influences the prediction (*e.g.* the place classification can be used to change the likelihood of different classes), and the temporal knowledge is represented by fusion of previous classification with new ones (Eq. 3.17).

Entity Update

The classifications made must be stored and updated coherently in a map-like structure. Most of the works in literature use grid maps to store and represent the semantics of the environment, like [9, 10, 12]. Although this work aims to use a topologic map, the process of updating the cells/vertices is still the same. The only difference is the way cells are selected to be updated. The logic for this will be discussed in Subsections 3.5.4 and 3.5.5. The beliefs of each cell can be updated using equation (3.18).

$$p_k(\hat{\mathbf{x}}_i|\mathcal{I}_{1:k}) = \left[1 + \underbrace{\frac{1-p_k(\hat{\mathbf{x}}_i|\mathcal{I}_k)}{p_k(\hat{\mathbf{x}}_i|\mathcal{I}_k)}}_{current\ \ classification} \cdot \overbrace{\frac{1-p_k(\hat{\mathbf{x}}_i|\mathcal{I}_{1:k-1})}{p_k(\hat{\mathbf{x}}_i|\mathcal{I}_{1:k-1})}}^{accumulated\ \ classification} \cdot \underbrace{\frac{p(\hat{\mathbf{x}}_i)}{1-p(\hat{\mathbf{x}}_i)}}_{prior\ \ knowledge}\right]^{-1} \tag{3.18}$$

Converting equation (3.18) to a log-odds representation and assuming a prior knowledge $p(\hat{x}_i) = 0.5$ (mentioned in page 5 of [52]), we have (Eq. 3.19):

$$\mathcal{L}_k(\hat{\mathbf{x}}_i|\mathcal{I}_{1:k}) = \mathcal{L}(\hat{\mathbf{x}}_i|\mathcal{I}_{1:k-1}) + \mathcal{L}(\hat{\mathbf{x}}_i|\mathcal{I}_k). \tag{3.19}$$

This representation needs a clamping step to limit the values frontiers, the same as the one presented in Subsection 3.5.2 (page 29). It can be described by $l_{min} \leq \mathcal{L}_k(\hat{\mathbf{x}}_i|\mathcal{I}_{1:k}) \leq l_{max}$.

Position Update

The position of objects can vary over time due to measurement errors or real object motion. In consequence, the position of an object has to be updated to ensure map coherence. The position of an object and its 3D AABB are updated using the low-pass filter expressed through equation (3.20).

$$\mathbf{o}(k) = \mathbf{o}(k-1) \cdot \left(1 - \frac{1}{1+e^{l_c(k)}}\right) + \overline{\mathbf{o}}(k) \cdot \left(\frac{1}{1+e^{l_c(k)}}\right) \tag{3.20}$$

Let define respectively $l_c(k)$ and $\bar{l}_c(k)$ as the clamped accumulated log-odds value of the bounding box confidence BB_c (Eq. 3.21) at an instant k, and the log-odds value of the estimated bounding box confidence BB_c at an instant k. $\mathbf{o}$ is the vector of positional object parameters,
$\mathbf{o} = \begin{bmatrix} x_{min} & y_{min} & z_{min} & x_{max} & y_{max} & z_{max} & x_{centroid} & y_{centroid} & z_{centroid} \end{bmatrix}$, wherein the minimum and maximum points represent the 3D AABB, and the centroid represents the assumed position of the object.

$$l_c(k) = l_c(k-1) + \bar{l}_c(k) \tag{3.21}$$

3.5.4 Positive Observations

A positive observation $\mathbf{D}^*$ is the group of data that is used by the system to make positive associations about the entities of the map. $\mathbf{D}^*$ represents a detected object, it can be used to either create a new object instance in the map or update an existing one. The diagram presented in Figure 3.22 illustrates the process.

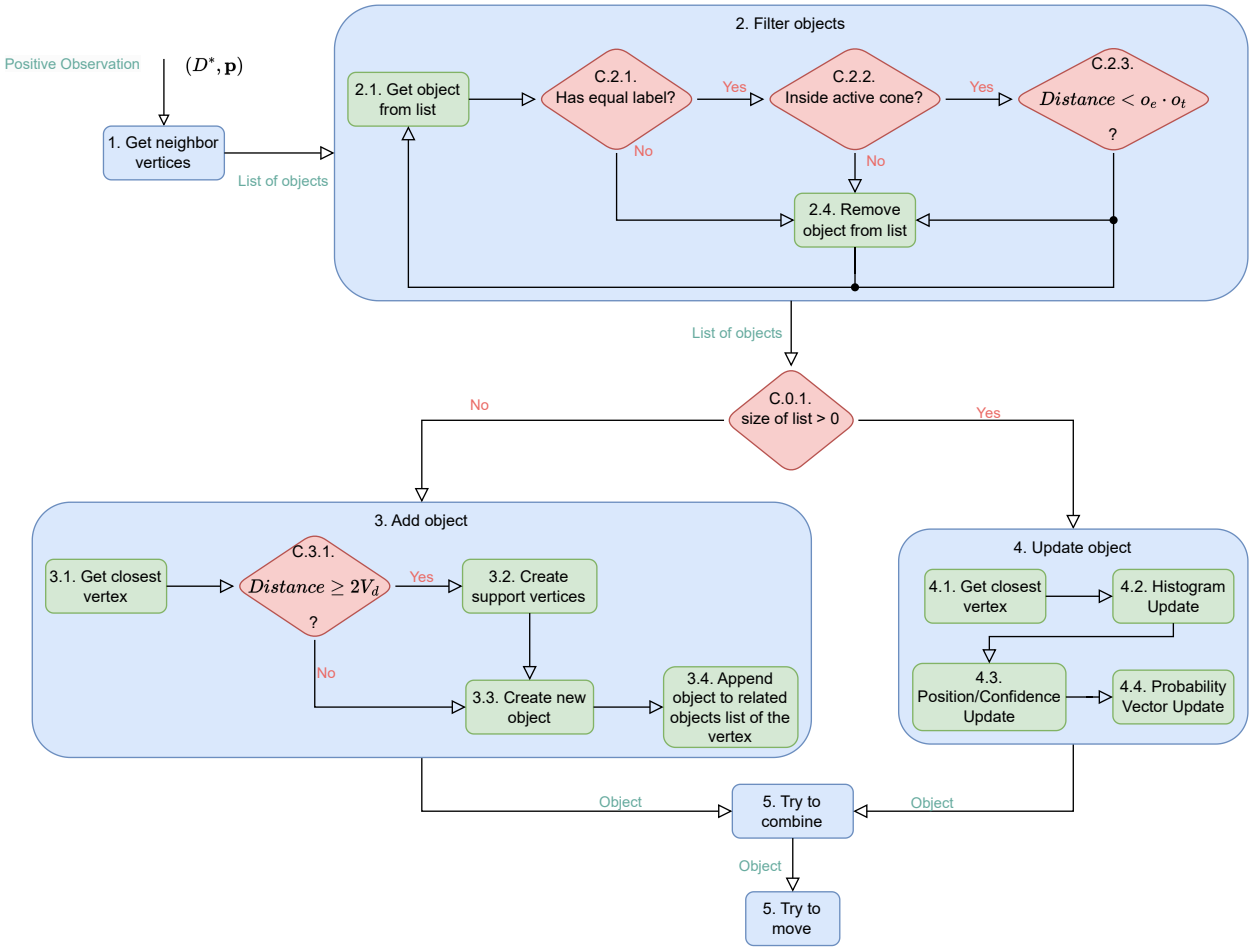

Figure 3.22: Positive observation flowchart. $(\mathbf{D}^*)$ represents a detected object, defined in equation (3.5, page 16). $(\mathbf{p})$ represents the robot pose.

As for the steps indicated in Figure 3.22, it is important to note:

- (1.) Gather vertices in a 3-layer deep neighborhood in relation to the robot position.
- (2.) Select only the candidates that can be matched with $(\mathbf{D}^*)$ (Eq. 3.5, page 16):
  - (C.2.1.) Check if both instances have the same classification label.

  - (C.2.2.) Check if the object is inside the active cone (similar to Figure 3.5, page 18).
  - (C.2.3.) Check if the distance between both instances is less than the product of *object position error* (o_e) and *object tracking factor* (o_t).

- (C.0.1.) Check if the list has any candidates left.
- (3.) Add a new object:
  - (3.1.) Get the closest vertex in relation to the object position.
  - (C.3.1.) Condition to enforce rule 1 proposed in Subsection 4 (page 27).
  - (3.3.) Create a new object instance with the data defined ($\mathbf{D}^*$) (Eq. 3.5, page 16).
- (5.) Try to combine two object instances if they are close enough.
- (6.) Try to move an object to another vertex to enforce condition 2 defined in Subsection 4 (page 27).

3.5.5 Negative Observations

The negative observations are responsible for removing invalid objects from the map. It reduces the beliefs of objects in a specific area based on the expected object position and distance from the camera. This is done by validating the registered object geometry with newly acquired cloud points that are expected to exist in the neighborhood of the AABB of the object. This process detects object absence and occlusions, as well as partial occlusions using the BDM referred to in Section 3.4. Figure 3.23 illustrates this process.

As for the steps indicated in Figure 3.23, it is important to note:

- (1.) Gather vertices in a 5-layer deep neighborhood in relation to the robot position.
- (C.1.) Check if the object is inside the active cone (similar to Figure 3.5, page 18).
- (3.) Compute the 8 points that represent the corners of the object AABB.
- (5.) Compute the corner matches. Corner matches are the BDM panels that have the closest angular position as the corner points. The angular difference is determined through equation (3.23).

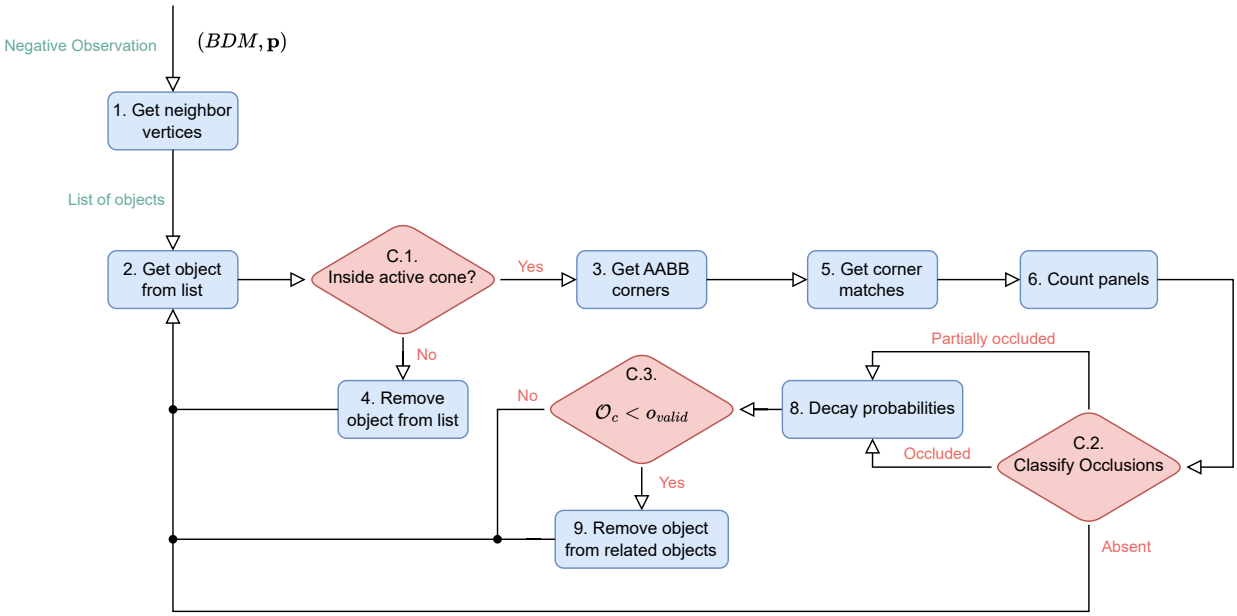

Figure 3.23: Negative observation flowchart. BDM is the binned depth map introduced in Section 3.4. $\mathbf{p}$ represents the robot pose.

Equation (3.22) shows a function that enforces the continuity of an angle difference between $]-\frac{\pi}{2}, \frac{\pi}{2}]$. Equations (3.24) and (3.25) express respectively, in relation to the camera, the angles to the panel and corner; where ϕ_{cc} is the angle from the camera to the corner, and ϕ_{cp} is the angle from the camera to the panel.

$$g(\alpha - \beta) = atan2(sin(\alpha - \beta), cos(\alpha - \beta)) \tag{3.22}$$

$$\phi_{diff} = g(\phi_{cc} - \phi_{cp}) \tag{3.23}$$

$$\phi_{cp} = g(atan2(y_{camera} - y_{panel}, x_{camera} - x_{panel}) - \theta_{camera}) \tag{3.24}$$

$$\phi_{cc} = g(atan2(y_{camera} - y_{corner}, x_{camera} - x_{corner}) - \theta_{camera}) \tag{3.25}$$

- (6.) Make the count of panels. Panels before the object (p_{be}), inside the object (p_{in}), and after the object (p_{af}). The panels are classified depending on the distance from the camera. Figure 3.24 illustrates the types of panels.

- (C.2.) Classify object as occluded, partially occluded, or absent based on p_{be}, p_{in}, p_{af}.

- (8.) Reduce the beliefs of the visibility histogram and likelihood vector $\mathcal{L}_k(\hat{\mathbf{x}}_i|\mathcal{I}_{1:k})$ (Subsections 3.5.2, and 3.5.3). The decay factor is attenuated when partially occluded.

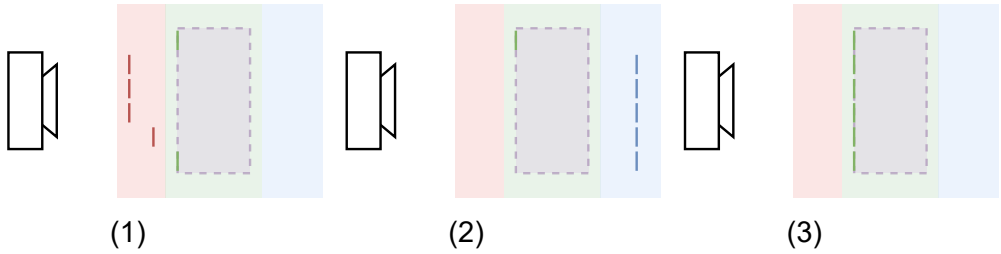

Figure 3.24: This image shows a top-view representation of a case of object occlusion (1), object absence (2), and a valid object (3). The purple box is the supposed object BB. The red area delimits panels before the object, the green area represents panels inside the object BB with tolerance, and the blue area represents panels after the object.

- (C.3.) Check if an object is still considered valid based on the condition *Object Confidence* ($\mathcal{O}_c$) $<$ *valid object* threshold (o_{valid}) (Eq. 3.27 and Eq. 3.10, page 28).

$$H_c = f_{ohc}(\underbrace{\mathbf{b}_{1:k}}_{\text{histogram bins vector}}) \tag{3.26}$$

$$\mathcal{O}_c = f_{oc}\left(\underbrace{p(argmax(\mathcal{L}_k(\hat{\mathbf{x}}_i|\mathcal{I}_{1:k})))}_{tp_1\text{: most probable class confidence}}, \overbrace{p(l_c(k))}^{tp_2\text{: bounding box confidence}}, \underbrace{H_c}_{tp_3\text{: observation histogram confidence}}\right) \tag{3.27}$$

4 System Implementation

The system was implemented in a Pioneer P3-DX platform having a Jetson AGX Xavier on board as its main computer (Figure F.2.2, page 112). The Jetson was equipped with an Intel 8265 M.2 Wireless (Figure C.6.36, page 100) Network to provide a Wi-Fi connection. In relation to the sensors, the system was equipped with a ZED 2 stereo camera (Figure C.6.34, page 99), and a Hokuyo URG-04LX laser range finder (Figure C.6.35, page 99). As for the OS, the system has the NVIDIA L4T 35.3.1 with the Jetpack SDK 5.1.1. Figure C.6.32 in Appendix C.6 (page 98) shows the robot and its sensors.

As mentioned before, one of the objectives of this dissertation is to develop a modular semantic mapping framework; a *microservice architecture* [53] was chosen to address the matter because of its flexibility, modularity, and isolation paradigms. Although ROS itself constitutes a *microservice* framework with each node being a service responsible for a simple task, the composition of nodes to perform a more complex but well-defined task is still considered as a microservice. In this work, the system was divided into self-contained components responsible for each task; it was structured to work inside Docker containers to isolate different types of dependencies and avoid possible conflicts. Both ROS Noetic and ROS Foxy were utilized as middlewares to support the development of the system. ROS Noetic was utilized because the P3-DX robotic platform only has support in ROS (up to the ROS Noetic distribution); the package *ros1_bridge* was utilized to make a connection between the two middlewares.

Despite the simultaneous use in the system of instances of ROS Noetic and ROS Foxy, the development was focused on ROS Foxy, thus on ROS 2. Tools like *RVIZ2*[1] and *Gazebo*[2] were utilized alongside standard ROS Foxy packages for common robotic tasks, such as *SLAM Toolbox* [54], to perform efficient SLAM, and *Nav2* [55, 56, 57, 58] for localization, path planning, motion control, and navigation behaviors.

This chapter describes the implementation details of the framework proposed in Chapter 3

[1]RVIZ2: https://github.com/ros2/rviz

[2]Gazebo Simulator: https://gazebosim.org/home

(page 13). Section 4.1 will present the containerized structure of packages utilized in the implementation and essential details about its components. Section 4.2 discusses the node architecture and its relationships, including communication details.

4.1 Container Architecture and Communication

The container structure of the system was developed with modularity and scalability as the main objectives. All containers are based on the official NVIDIA L4T 35.2.1 docker image and have isolated dependencies depending on each application case. The only case of cross-container dependencies is with respect to inter-container communication; containers need to have the same message types to correctly communicate. The interfaces utilized are a combination of standard ROS 2 messages, ZED interfaces[3], and Semantic Mapping (SMAP) interfaces (see repository H.1.2, page 124), which is a custom interface developed specifically for the needs of this project.

The communication between containers is achieved using a combination of a docker host network driver and the ROS middleware interface. The ROS middleware interface was completely remodeled in the transition from ROS (Transmission Control Protocol (TCP) based) to ROS 2 (Data Distribution Service (DDS) based). Since this project is oriented towards the ROS Foxy distribution, all the inter-container communication is handled by the DDS communication, with the TCP communication being used only inside the P3-DX container for intra-container communication (see Figure 4.1). Different from the TCP ROS implementation, the ROS 2 DDS is not a ROS 2 implementation but a wrapper. The developer has the option to choose between a few commercial-grade implementations[4] being the Cyclone DDS[5] the only one capable of handling a stable and reliable communication using a Jetson host and a remote desktop client.

P3-DX Robot Interface Container

The *P3-DX container* handles the interactions with the physical robot. It has the *ROS Aria Node* (robot drivers), the Robot Description Node, which was ported from ROS Noetic (see repository H.1, page 124, in the appendices), the *Emergency Stop Node*, which was adapted to work with a Jetson host, and the *ROS Bridge Node*[8]. The *ROS Aria Node* is

[3]ZED interfaces: github.com/stereolabs/zed-ros2-interfaces
[4]ROS 2 RMW implementations: tinyurl.com/2p98y8f9
[5]Cyclone DDS: github.com/eclipse-cyclonedds/cyclonedds
[8]ROS Bridge Package: github.com/ros2/ros1_bridge

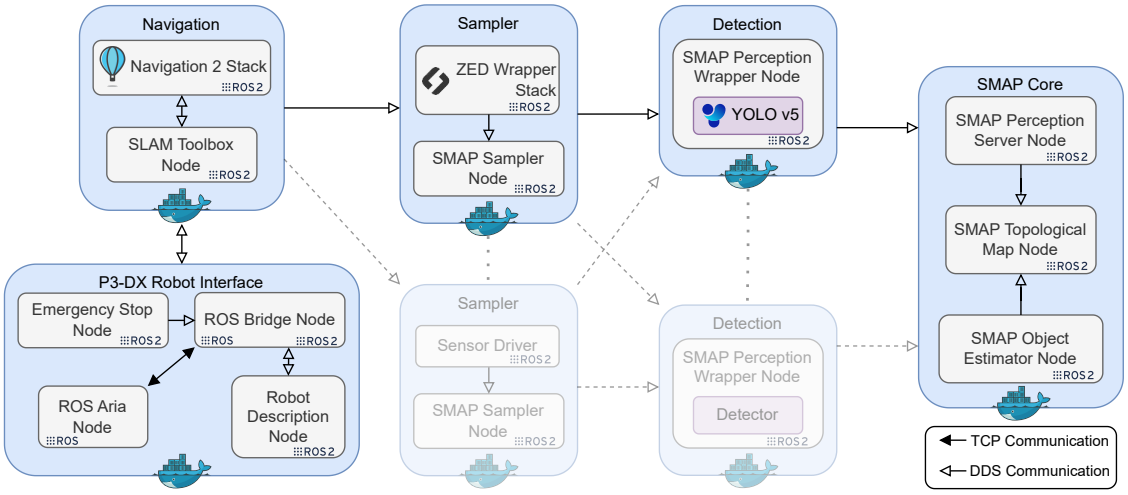

Figure 4.1: Container architecture diagram. The diagram shows the docker containers of the system and its main applications; the continuous lines represent connections of the containers, and the dashed lines represent possible extra connections. Navigation 2 Stack [55]. ZED Wrapper Stack[6]. Ultralytics YOLOv5[7].

responsible for the wheels' motor actuation and the odometry of the platform. *Robot Description Node* defines and updates the static and dynamic mathematical model of the robot, thus defining the transformations between robot frames and updating the joint values over time. The *Emergency Stop Node* implements a physical switch to disable the robot motors in case of an emergency (see Appendix G, page 122). Lastly, the *ROS Bridge Node* deals with the communication between ROS Noetic and ROS Foxy environments. A parametric bridge was utilized to avoid bridge overloading. This means that only explicitly defined messages and services will be forwarded to the other environment[9].

Navigation Container

The *navigation container* is responsible for all the actions related to the navigation of the platform, ranging from SLAM to failure recovery behaviors. The SLAM task, usually performed in ROS 1 by third-party packages like *Gmapping*[10], *Cartographer*[11] and RTab[12], were substituted in ROS 2 by the officially supported *SLAM Toolbox* package [56]. This package is focused in creating lifelong maps, that can keep improving the map even after system reinitalization, by maintaining pose graphs between sessions. The package is based on the well popular *open_karto* SLAM library [59] that implements an efficient pose graph

[9] *Parameter bridge* parameter definition: tinyurl.com/76ych53v
[10] Gmapping: openslam-org.github.io/gmapping.html
[11] Cartographer: google-cartographer.readthedocs.io/en/latest/
[12] RTab-Map: github.com/introlab/rtabmap

SLAM algorithm. The *Navigation 2 Stack* [56] is a ROS 2 package developed to easily incorporate production-ready navigation capabilities into robots. It provides functionalities and tasks like *Adaptive Monte Carlo Localization (AMCL)*, *speed controller*, *path planning*, *path planning smoothening*, *collision monitoring*, *waypoint following*, *recovery behaviors* and many others. Both packages *Navigation 2* and *SLAM Toolbox* are part of the same ecosystem[13] and are directly integrated, providing seamless interoperability between them.

Sampler Container

The *sampler container* is responsible for the sampling of all the data necessary to perform semantic mapping. It contains the SDK and packages necessary to collect and process the raw sensorial data. This container is composed of two components: *ZED wrapper Stack* and *SMAP Sampler Node*. The former is a dependency of the ZED 2 stereo camera; it performs the tasks necessary to collect and process the raw sensor data, as well as establishing the transformations between the multiple camera sensors and the robot frame. The latter is responsible for aggregating all the data necessary to perform semantic mapping into a single message packet (see repository H.1.2, page 124, in appendices).

Detection Container

This container is used to enclose a ROS 2 wrapped object detector and all of its dependencies. In this case, a YOLOv5[14] object detector was wrapped using the developed *SMAP Perception Wrapper Node* (see repository H.1.2, page 124). This node is a generic ROS 2 node packed with utility tools such as image pre-processing and post-processing, as well as predefined services to register the detector in the system and publish the detections in the correct format. The detector registration process will be explained in Section 4.2.

Although the *YOLOv5* implementation is *Python*-based using *pyTorch*, the model itself is serialized and deployed in the *C++ NVIDIA deployment interface TRT*, which offers a performance boost due to high levels of optimization for production environments. The model utilized was a pre-trained "*YOLOv5s*" model[15]; it was trained for 300 epochs with images of the COCO Dataset[16].

[13]Open Navigation LLC: opennav.org/

[14]YOLOv5: https://github.com/ultralytics/yolov5

[15]YOLO pre-trained models: github.com/ultralytics/yolov5#pretrained-checkpoints

[16]COCO Dataset: cocodataset.org/#home

SMAP Core Container

The *SMAP Core Container* has the necessary library (PCL, and *Boost Library*[17]) dependencies necessary to run the SMAP Nodes to perform the semantic mapping task. It is composed of the *SMAP Perception Server Node*, which is responsible for the detector registration; the *SMAP Object Estimator Node* which handles the *Geometric Segmentation* and *BDM* construction tasks; and the *SMAP Topological Map Node* which implements the mechanisms necessary to maintain the map.

The *SMAP Perception Server Node* is a ROS server that handles the registration of new detectors. It manages the structure of the "*classes_probabilities*" hash map mentioned in Subsection 3.5.1 (page 26), and only the values of it can be updated by the *SMAP Topological Node*. This node is essential for the system to know what classes are known by each detector and ensures that the values are assigned to the correct classes.

The *SMAP Object Estimator Node* performs the *Geometric Segmentation* and BDM tasks. For the first one, the node receives as input a group of detections $\mathbf{D}$ (Eq. 3.4, page 15), which has a d number of detected objects. It launches a new thread, with a limited lifetime, for each object in order to reduce the overall latency of processing. The maximum number of threads allowed is 8. The lifetime of threads is explained in Appendix F.4, page 114. The second task is executed in an independent thread and expects as input an *ordered point cloud*. The cloud is processed to extract multiple AABBs related to cloud subregions (see Section 3.4, page 23 that are stored in a 2D matrix-like structure that is propagated to the *negative association* step of the *SMAP Topological Map Node*.

The *SMAP Topological Map Node* creates and maintains a graph that represents a *Topological Map* (see Subsection 3.5.1, page 26). The structure of the graph is implemented using an *Adjacency List* [60], in which the *vertices* represent the landmarks that store the semantic payload relative to that physical location, as well as the objects related to that place. The *edges* of the graph represents traversability between two vertices. In addition to graph management, this node has callbacks to handle incoming object features ($\mathbf{D}^*$, BDM) (see Figure A.1, page 65); to perform the *Data Association* task (see Subsections 3.5.4, and 3.5.5); and a callback to expand the graph based on the robot movement (see Subsection 4, page 27).

[17]Boost Libraries: boost.org/

4.2 ROS Architecture

This section discusses the ROS architectural details of the implementation as well as essential details about each SMAP node. Figure 4.2 shows the node graph related just to the *SMAP nodes*. The complete node graph of the system and the *Transform (TF) tree diagram* are presented in Appendix C.5.

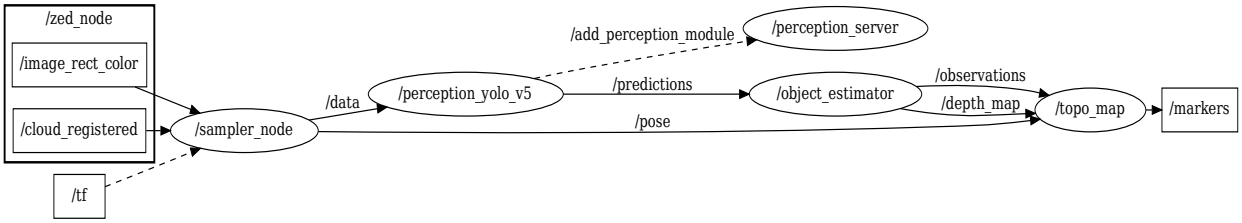

Figure 4.2: *SMAP ROS nodes* graph. Solid lines represent topics and dashed lines represent services. Topic prefixes were omitted for simplicity.

sampler_node

As mentioned before, the *sampler node* (see Appendix H.1.2) is an aggregator node whose main purpose is to gather all the data necessary to accomplish the *semantic mapping* task. It gathers 3 types of information: *RGB* image ("*/image_rect_color*")[18], *colored point cloud* ("*/cloud_registered*")[19], and *robot pose* ("*/tf*"). The node has 2 timers: the first one triggers a callback that publishes the *robot pose* to "*/pose*"[20] at a 20 Hz rate; the second one triggers a callback that publishes the gathered data to the topic "*/data*"[21], limited by the sensor publish rate that is configured to 10 Hz. Additionally, the *sampler node* makes memory level verifications in its messages before publishing, to ensure that only valid data is propagated.

perception_yolo_v5

This node is an instance of the *smap_perception_wrapper* (see Appendix H.1.2, page 124) that encapsulates the YOLOv5 detector[22]. When started, the node makes a call to the service "*/add_perception_module*" to register this instance and its known classes. If successful, the node starts to make predictions. Multiple objects can be detected in an image received in

[18] sensor_msgs/Image: tinyurl.com/54s3b4nh
[19] sensor_msgs/PointCloud2: tinyurl.com/3rwcrznc
[20] geometry_msgs/PoseStamped: tinyurl.com/586cezn8
[21] *SmapData*: tinyurl.com/46f6r8fe
[22] YOLOv5 list of classes: https://github.com/ultralytics/yolov5/blob/master/data/coco.yaml

the topic "*/data*". Each detected object is represented by a *SmapObject*[23], and the group of objects detected in an image composes a *SmapDetections*[24] (Eq. 3.4, page 15) message that is published to the topic "*/predictions*".

perception_server

The *perception server* is responsible for the management of the *probability hash map* (Pseudocode 4, page 26). It preserves the coherence of the *hash map* while adding new detectors. New detectors are added through requests made in the service "*/add_perception_module*". Each detector receives a unique identifier, that is used to distinguish the origin of different detections that were classified with the same class. The *perception server* also maintains metadata related to added detectors to be later able to re-establish a connection with a previously registered detector.

object_estimator

This node performs 2 important tasks: *geometric segmentation* (Section 3.3, page 15) and BDM creation. Both of them are handled by the "*/predictions*" callback. This function splits the data into multiple threads to balance the computational load (see Appendix F.1, page 114).

topo_map

The *topo_map node* is responsible for making the *data association* (see Subsections 3.5.4, page 34 and 3.5.5, page 35) and publishing object markers to *RVIZ2*. The positive observations are handled by the "*/observations*" callback (see Subsection 3.5.4, page 34), and negative observations are handled by the "*/depth_map*" callback (see Subsection 3.5.5, page 35).

In addition to the callbacks mentioned before, this node has two timers that trigger callbacks for publishing and updating the map visualization. Both callbacks are processed in independent threads and a mirrored representation of the map, stored in ROS markers array message format, is utilized to reduce access times to the shared *topological map* structure. The publishing timer has a frequency of 4 Hz and only publishes the existing message. The update timer has a frequency of 1 Hz; it accesses the proper *topological map* and updates the marker array. All the accesses to the topological map and the marker array are synchronized using mutexes provided by the Linux operating system.

[23] *SmapObject*: tinyurl.com/mr3x7hd7

[24] *SmapDetections*: tinyurl.com/ytddt2ny

5 Experiments and Evaluation

The system described in Chapter 4 was deployed and tested inside the Mobile Robotics Lab (MRL) of the Institute of Systems and Robotics (ISR) in a non-controlled environment. Despite the ability of the robot to navigate and map the environment autonomously, the robot was moved manually around the lab because we did not manage to acquire in time the electronic parts necessary to fulfill the power requirements (20VDC:4.5A∼9VDC:10A) of the Jetson AGX Xavier[1] to be powered by the robot's batteries. Figure 5.1 shows a sample of the semantic map of the MRL.

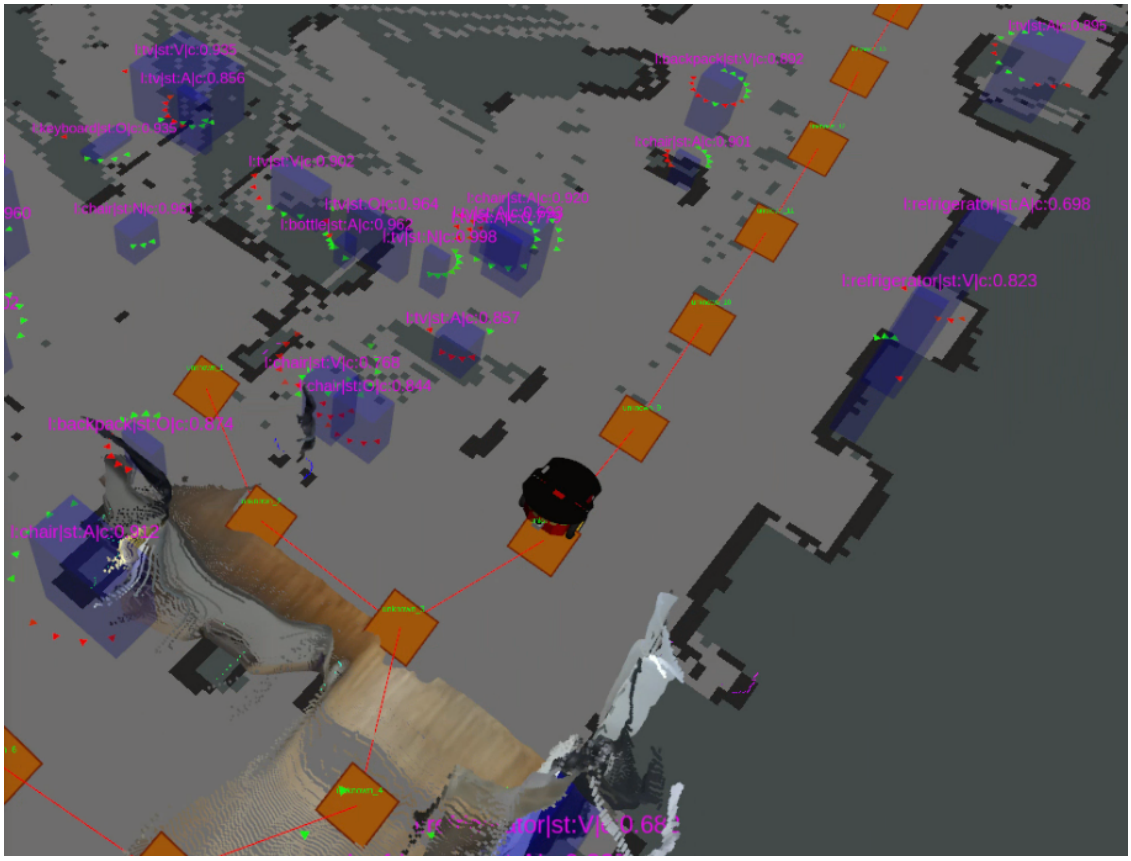

Figure 5.1: SMAP map in RVIZ. Figures C.4.26, C.4.27, C.4.28, and C.4.29 (page 93) illustrate the correspondent locations in MRL. Figures 5.2 and 5.3 explain the SMAP indicators.

[1]https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-series/

Two experiments were performed. In the first one, the platform traveled the path illustrated in Figure C.4.23 and essential data, such as execution times (Section 5.1), point cloud decay (Section 5.2), object classification precision (Section 5.3), and map size (Section 5.4), were recorded. The second experiment aimed to evaluate the object positioning accuracy (Section 5.3); the robot was conducted through a smaller path (see Figure C.4.25, page 91) in which 15 scene objects had their centroid coordinates manually measured (see Figure C.4.24, page 90) to obtain an object *ground truth*.

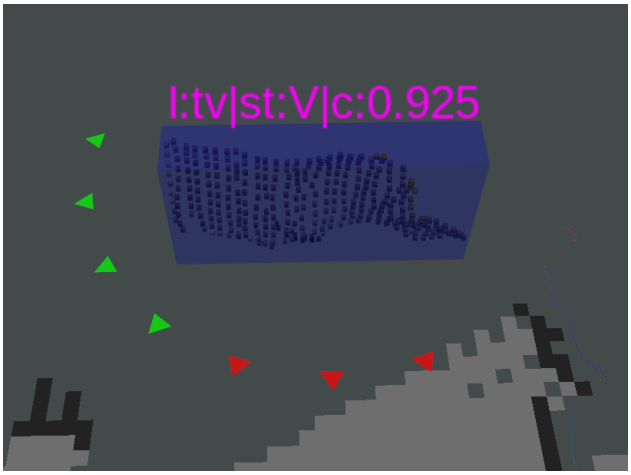

Figure 5.2: SMAP object indicator. The purple text shows the label ("*l:TV*"), object state ("*st:V*") (*V - Visible, O - Occluded, A - Absent*), and object confidence ("*c:0.925*"). The blue parallelepiped is proportional to the object's volume. Triangles around the object represent bins of the *Visibility Histogram* (see Subsection 3.5.2, page 27); the color ranges from red ($l_{odds} < 0$) to green ($l_{odds} > 0$), and values close to $l_{odds} = 0$ (No observations) have the triangles ommited.

The parameters utilized in the experiments are presented in Appendix D (page 102); Figure 5.1 shows an example of the *Semantic Map* of the MRL, and a video of the system in action is available in Appendix I (page 126). More examples of the SMAP map are available in Section 5.3 and Appendices C.2 and C.3. Extra results are available in Appendix E.

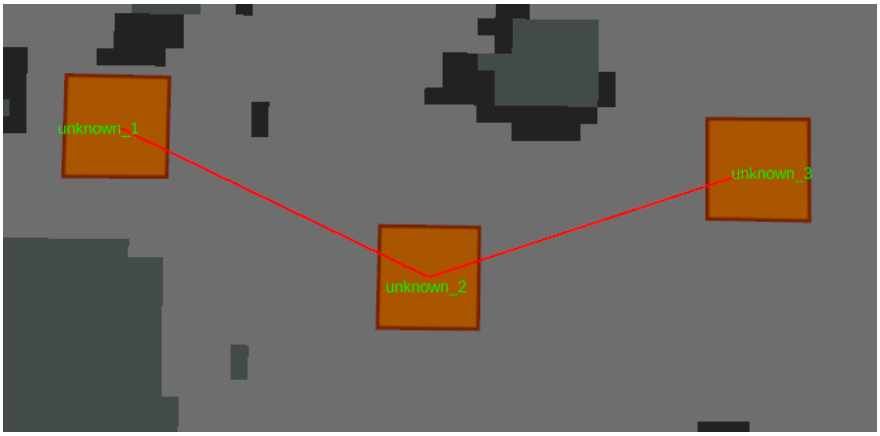

Figure 5.3: SMAP place indicator. The orange squares represent the position of the main vertices of the topological map, the green text is the place classification of this vertice, and the red line shows the connections between vertices (the *edges*).

5.1 Execution Times

This section presents the execution times of the main steps of the system presented in Chapters 3 and 4. The execution times were recorded using only one thread of the geometric segmentation process to simplify the execution time estimation.

The plot presented in Figure 5.4 shows the execution times of the YOLOv5 object detector. It shows that the *Pre Processing* (letterbox transformation[2] + tensor initialization), *Inference* (object predictions) and *Non-Maximum Suppression (NMS)*[3] steps can be performed fast. Contrarily, the *Post Processing* step which overlays BBs in an image (see Figure 3.2, page 15) tends to bottleneck the object detection. The median and the maximum execution times are $91ms$ $(10Hz)$ and $213ms$ $(4Hz)$ respectively.

Figure 5.5 shows execution times of a single thread of the *Geometric Segmentation* pipeline. The plot contains a lot of outliers due to the high variation in the point cloud number of points and the patterns in it. *Voxelization*, *Statistical Outlier Removal (SOR)* and *Clustering* are the steps most affected. Despite the big-time variations, the whole segmentation pipeline can be executed in $82ms$ $(12Hz)$, in the worst-case scenario, and in most of the time in just $0.36ms$ $(2.7KHz)$.

The execution time of the system's callbacks is shown in Figure 5.6. Each callback is responsible for performing one of the main steps of the semantic pipeline (see Figure A.1,

[2]Letterbox transformation is a process that resizes the image to a square format. 640×640 in YOLOv5.

[3]Non-Maximum Suppression (NMS) is a filter step that removes lower score overlapping Bounding Box (BB).

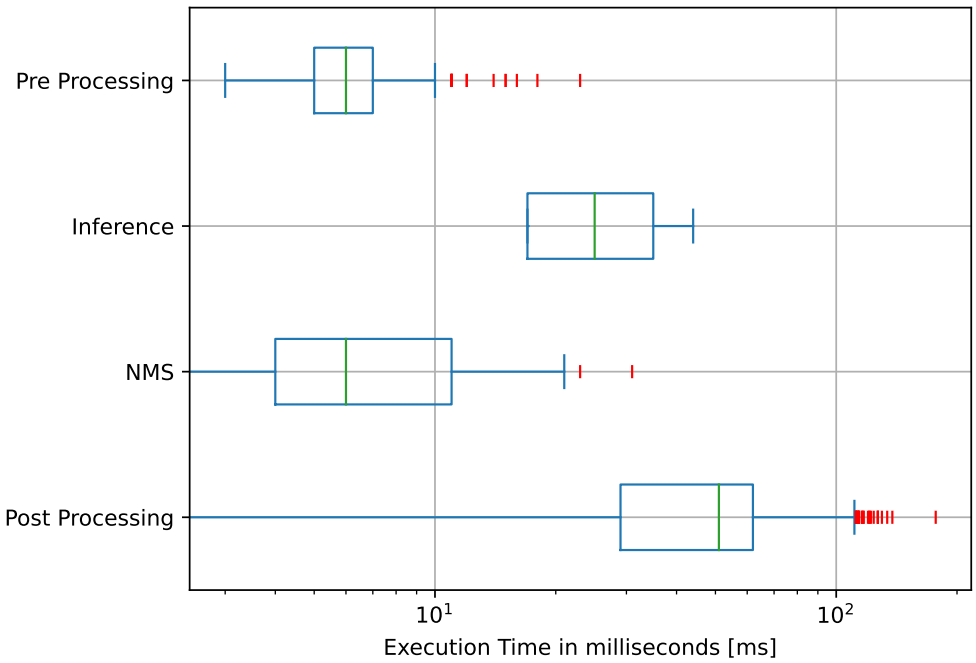

Figure 5.4: Object detector execution times (Box Plot details in Appendix C.7, page 101).

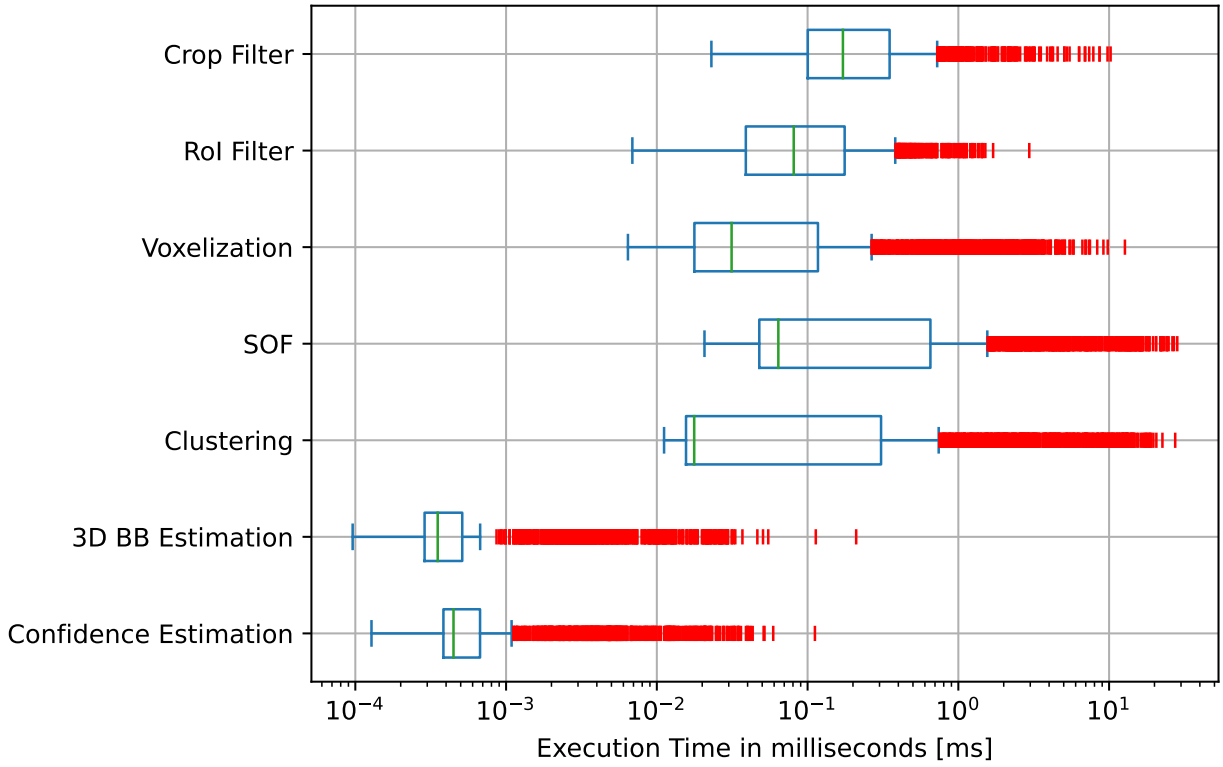

Figure 5.5: Geometric segmentation times (Box Plot details in Appendix C.7, page 101).

page 65). The plot shows the system's main implementation weakness: both the *Geometric Segmentation* and *Point Cloud Binning* are processed in independent threads that are susceptible to system preemption. This aspect is visible at the execution times of the *Geometric Segmentation* callback in Figure 5.6 and is propagated to its steps (Figure 5.5). The *Point*

Cloud Binning is less impacted due to fewer function calls. Disabling system preemption in the Operating System (OS) and setting reasonable task priorities can lead to performances close to the median times. The sum of the maximums and the medians are $3155s$ ($316mHz$) and $167ms$ ($6Hz$) respectively.

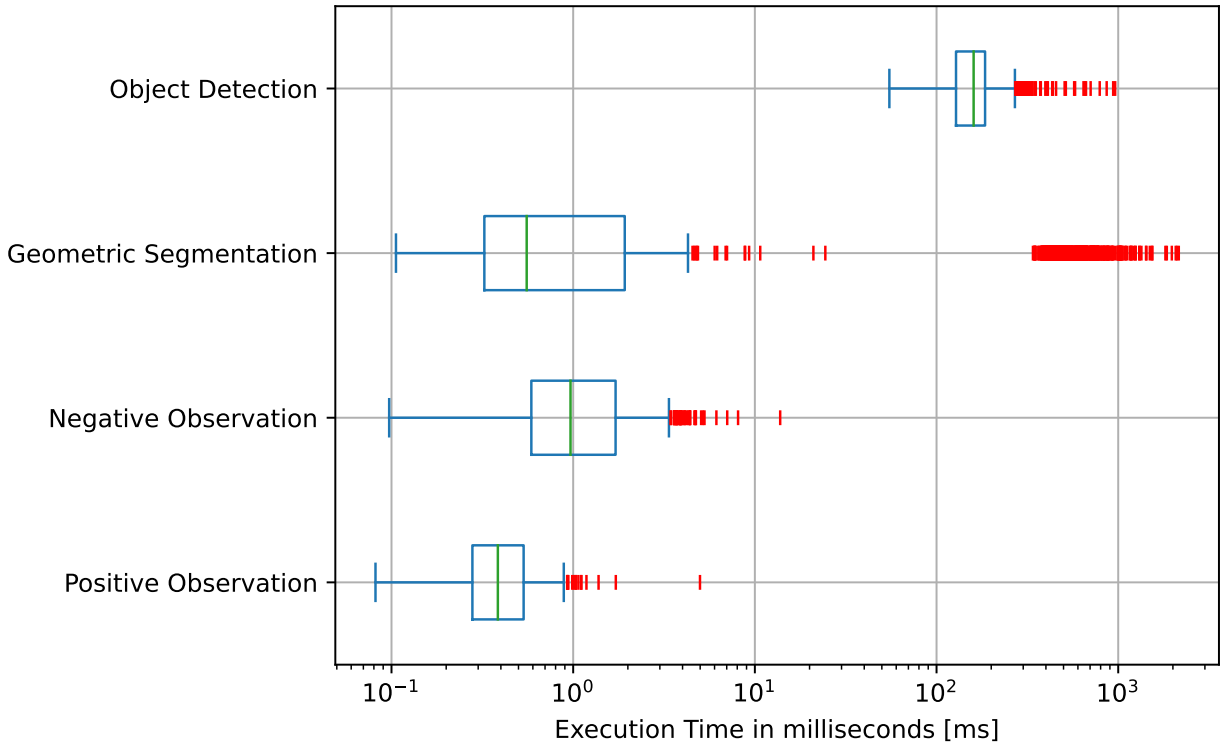

Figure 5.6: Callback times (Box plot details in Appendix C.7, page 101).

5.2 Point Decay

This section presents the reduction of cloud points along steps of the *Neck* of the *semantic pipeline* (see Figure A.1, page 65). Figure 5.7 shows the filtering influence of *Geometric Segmentation* steps, although *Region of Interest (RoI) Filter* and *Statistical Outlier Removal (SOR)* seem to have almost no influence, they are essential to remove outliers and prevent bad estimations.

Different from the *Geometric Segmentation*, *Point Cloud Binning* has a fixed output size that depends only on the system parameters (see Appendix D, page 102). In this case, the output Binned Depth Map (BDM) has size $24 \times 16 \times 3 = 1152$ points.

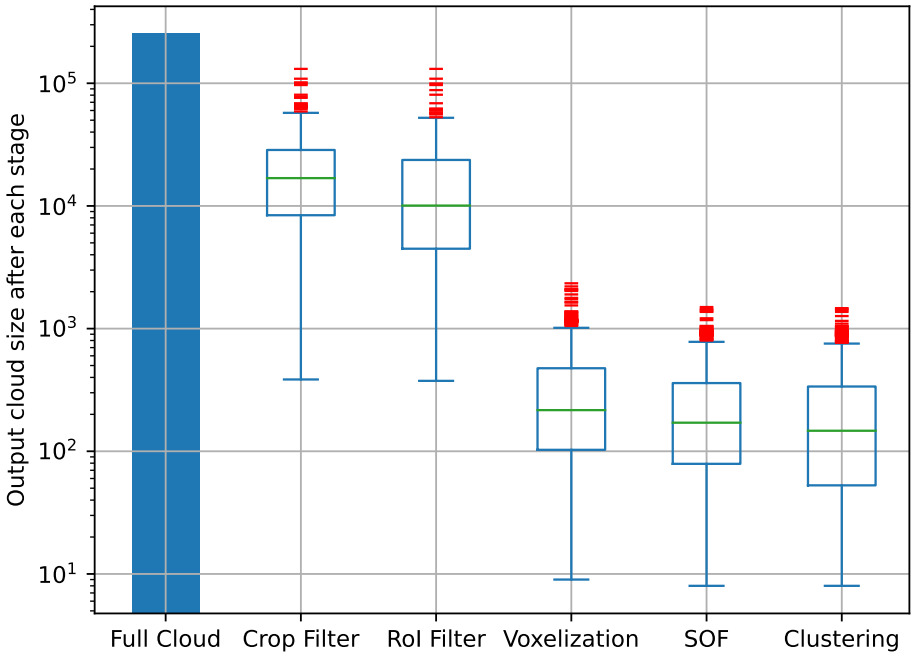

Figure 5.7: Geometric segmentation points. Plot details in Appendix C.7, page 101.

5.3 Object Registration

The *Object Registration* was evaluated in two experiments: Figures 5.1 and 5.8 illustrate the maps collected while experimenting. The first experiment tried to evaluate metrics like True Positive (TP), False Positive (FP), and Precision (PRE) of the system. The acquisition of the negative metrics True Negative (TN) and False Negative (FN) was not possible due to the non-controlled environment utilized for the tests, making it impracticable to determine if an object should have been registered or not. Table 5.1 shows the results obtained. The metrics were computed based on the exported equivalent 2D maps generated by the *Map Exporter* (see Appendix F.6, page 116). The map of each class was manually compared and the TP and FP metrics were evaluated.

	Backpack	Bottle	Chair	Couch	Keyboard	Suitcase	Table	TV	Umbrella
TP	1	1	15	1	2	1	3	10	1
FP	0	0	11	1	1	1	1	8	1
PRE	100%	100%	57.7%	50%	66.7%	50%	75%	55.6%	50%

Table 5.1: Object registration metrics. The grid maps utilized to create the table are presented in Appendix C.3 (page 78). Precision (PRE).

It is important to note that Table 5.1 was generated using an optimistic system configuration ($o_{valid} = 0.5$) to generate a large number of entities in the map. As a consequence, in regions where the robot makes fast turns the systems tend to create multiple instances of the same entity instead of updating the existing one, therefore reducing the Precision (PRE). This is visible in the demonstration video in Appendix I (page 126) as well as in Figures C.3.21 and C.3.16, Appendix C.3.

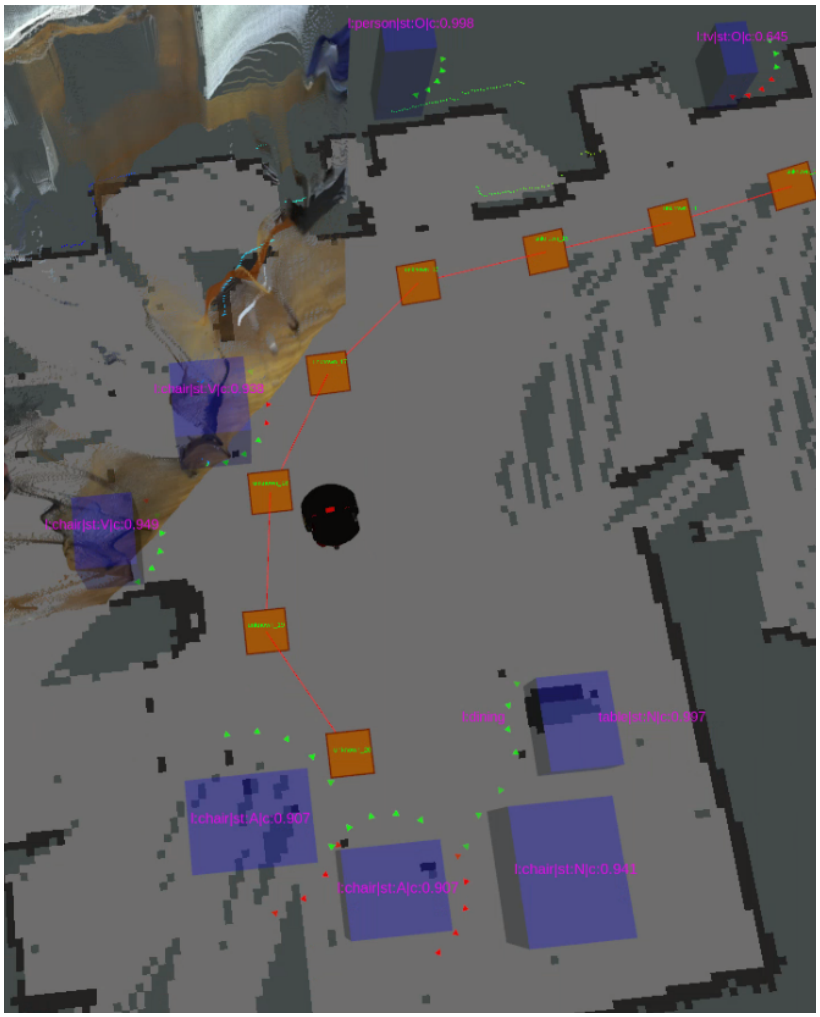

Figure 5.8: SMAP visualized in RVIZ2. The image shows good estimations of both the position and volume of objects in the scene. Figure C.4.28 (page 93) shows the correspondent location in the MRL. Figures 5.2 and 5.3 explain the indicators present in the figure.

The second experiment tried to estimate the positional accuracy of the map. A *ground truth* plant of the MRL (Figure C.4.24, page 90) was manually measured and compared with the positions acquired by the system while mapping the area. The results of the experiment (Table 5.2) show a noticeable contribution of the zz-axis error to the total error (if object 1 is considered an outlier). This can be explained by the low number of observations of a certain object. Figure 5.9 shows an example of two objects viewed from only one static

perspective. Therein, the AABB of the chair is floating (*i.e.* it is above the floor) because the system did not have the opportunity to observe the chair's legs. Figure 5.9 also displays a problem related to the system's preference for the biggest cluster available while acquiring the object's geometry. This greedy preference has led the system to wrongly position the AABB of the supposed "*refrigerator*" on top of the bottle, that it did not recognize. Those two problems are mostly attenuated when the robot has the opportunity to make multiple observations from different perspectives.

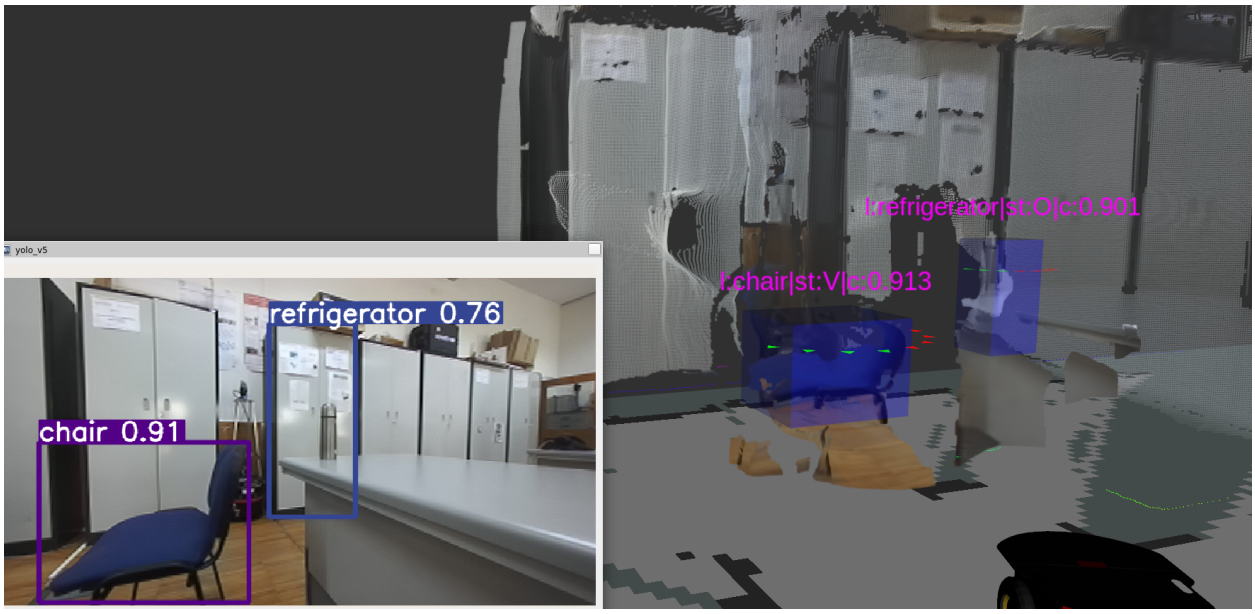

Figure 5.9: SMAP estimation problems. The image displays the map's problems when observing objects from only one static perspective. The chair is floating due to limited Point Cloud information and the "*refrigerator*" is placed in the wrong position in space (see Section 5.3, page 50). Figures 5.2 and 5.3 explain the indicators present in the figure.

It is important to note that objects 1 and 2 (see Table 5.2) of the second experiment had bad results due to a small object dimension combined with a low number of observations because of the trajectory traversed by the platform and their positions (Figure C.4.24, page 90).

Reference	Class	Error [m]			
		x	y	z	Distance
1	Bottle	1.49	-0.14	0.13	1.50
2	Bottle	-	-	-	-
3	TV	0.02	-0.01	0.02	0.03
4	TV	-0.02	-0.02	-0.01	0.03
5	TV	-0.15	0.21	0.05	0.26
6	TV	0.04	-0.03	0.03	0.06
7	TV	0.01	0.03	0.03	0.05
8	TV	0.33	-0.31	-0.12	0.47
9	TV	0.02	-0.04	0.03	0.05
10	TV	0.03	-0.02	0.02	0.04
11	Chair	-0.00	0.02	0.30	0.30
12	Chair	-0.02	0.03	0.26	0.27
13	Chair	-0.02	-0.04	-0.02	0.05
14	Chair	0.01	0.01	0.23	0.23
15	Chair	0.01	-0.01	0.23	0.23

RMSE	x	2.36
	y	0.17
	z	0.30
	Total	0.43

Table 5.2: Root Mean Square Error (RMSE) table generated from the mean object error of tables E.1.1, E.1.2 and E.1.3 (see Appendix E.1). Object 2 was not detected by any of the tests and object 1 has a much bigger error than the other objects. RMSE without objects 1 and 2: $RMSE_x = 0.14$, $RMSE_y = 0.15$, $RMSE_z = 0.28$, $RMSE_{total} = 0.20$.

5.4 Map Size

The occupancy grid map representation is widespread in various robotics applications including semantic mapping (see Table 2.1, page 8). While the grid map is an intuitive spatial representation for human visualization, it takes more space while storing less information; each cell only has the resolution of 3 values (Free, Unknown, Occupied), and an independent grid map for each known class is needed (see Table 2.1, page 8). Contrarily, the "*.smap*" file format serializes all the information necessary to describe the SMAP map, such as *Visibility Histogram* (see Subsection 3.5.2, page 27), *Entity 3D Axis Aligned Bounding Box (AABB)*, and the full likelihood vector $\mathcal{L}_k(\hat{\mathbf{x}}_i|\mathcal{I}_{1:k})$ (see Subsection 3.5.3, page 29) with floating point precision. Figure 5.10 compares the size of the same map in different formats. The 2D grid map was obtained using the *Map Exporter* (see Appendix F.6). The 3D grid map is an extrapolation of the 2D grid map using the same resolution and assuming the same height of the Mobile Robotics Lab (MRL) ($\frac{3.75m}{0.05m/cell} = 75\,cells$).

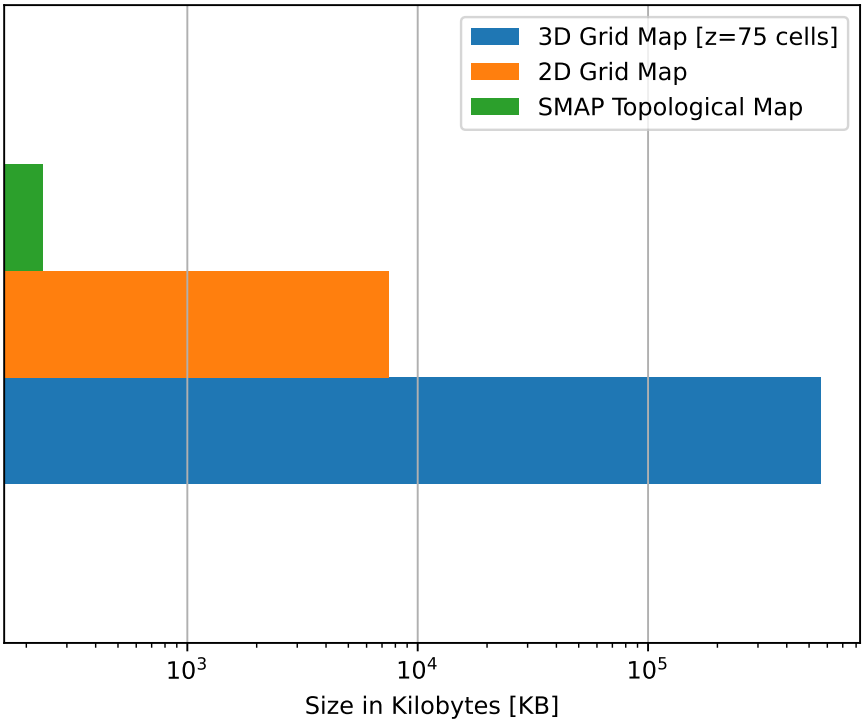

Figure 5.10: Map size comparison. The SMAP topological map uses roughly $250KB$ while the 2D and 3D grid maps can take up to $7.5MB$ and $550MB$ respectively.

6 Conclusion

In this dissertation, a semantic mapping framework (SMAP) was proposed, implemented, deployed, and tested in a real mobile robotic platform using *edge computing* provided by the Jetson AGX Xavier.

The framework was structured in a microservice architecture [53] with modularity and scalability in mind. Containers with simple and well-defined objectives were combined to accomplish a complex task; the containers also isolated groups of dependencies to ensure easy system deployment and reproducibility.

A comprehensive *Semantic Pipeline* was developed with a built-in *Geometric Segmentation* (Section 3.3, page 15) capable of transforming generic 2D image detectors (such as YOLO) into 3D object estimators. The pipeline also takes advantage of an *Visibility Histogram* (Subsection 3.5.2, page 27) to better understand the environment around it and avoid wrong object removal. Additionally, the pipeline enables expandable classification (see Subsection 3.5.3) that makes possible the use of old maps while adding new classifiers to the system.

Finally, the topological map utilized to store the environment semantics is lightweight and capable of fast entity access. The serialized "*.smap*" format is richer than the common grid map and the overall system is performant enough to be deployed in an edge platform such as the Jetson embedded computer or similar.

Future Work

Despite many features and advantages, the system developed is far from perfect and has a long list of improvements that were not possible due to the limited time window of this dissertation project. The following list summarize some improvements that would be worth to exploit in future work:

- *Better multi-threading implementation.* The multi-threading solution utilized works as

intended in terms of parallelism, but is not correctly configured to obtain the expected performance gains, leaving behind a lot of performance potential.

- *Place classification.* The framework was built with all the structures and features necessary to incorporate place semantics[1][2] with even features like custom, per class, traversability cost and place prior knowledge influence in object classification. Despite all the support structures, a proper place classifier was not integrated.

- *Better parameter tuning.* Due to a lack of time, a lot of parameter values that were utilized are far from ideal, especially the functions (f_{ohc}, f_{oc}) (see Table D.2, page 104) that were simplified to make the deployment of a first iteration of the framework doable in time. For instance, the tp_3 component (Eq. 3.27, page 37) of f_{oc} should have been a multiplication factor in the whole function to impose a greater influence of the *Visibility Histogram* (see Section 3.5.2, page 27) instead of just a weighted sum of components tp_1, tp_2 and tp_3.

- *Load balancing.* Thanks to the *microservice architecture*, the computing load of different processing steps of the SMAP framework can be balanced using a *container orchestration*[3] system (such as *Docker Swarm* and Kubernetes) to deploy multiple containers performing the same task. Additionally, taking advantage of the Data Distribution Service (DDS) adopted in ROS 2, the containers can be deployed in different devices within the same network to offload the Jetson.

[1] Vertex structure: https://tinyurl.com/2duwc42f

[2] Edge structure: https://tinyurl.com/knvtt4ds

[3] Container orchestration automatically provisions, deploys, scales, and manages containerized applications without worrying about the underlying infrastructure. Developers can implement container orchestration anywhere containers are, allowing them to automate the life cycle management of containers. [https://cloud.google.com/discover/what-is-container-orchestration].

7 Bibliography

[1] N. Blodow, L. C. Goron, Z.-C. Marton, D. Pangercic, T. Rühr, M. Tenorth, and M. Beetz, "Autonomous semantic mapping for robots performing everyday manipulation tasks in kitchen environments," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4263–4270, IEEE, 2011.

[2] C. Galindo, A. Saffiotti, S. Coradeschi, P. Buschka, J.-A. Fernandez-Madrigal, and J. González, "Multi-hierarchical semantic maps for mobile robotics," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2278–2283, IEEE, 2005.

[3] S. Coradeschi and A. Saffiotti, "An introduction to the anchoring problem," *Robotics and Autonomous Systems*, vol. 43, no. 2, pp. 85–96, 2003. Perceptual Anchoring: Anchoring Symbols to Sensor Data in Single and Multiple Robot Systems.

[4] B. Smith, "Objects and their environments: from aristotle to ecological ontology," in *Life and motion of socio-economic units*, pp. 84–102, CRC Press, 2000.

[5] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, 1993.

[6] B. Kuipers, "Modeling spatial knowledge," *Cognitive Science*, vol. 2, no. 2, pp. 129–153, 1978.

[7] B. Kuipers and Y.-T. Byun, "A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations," *Robotics and Autonomous Systems*, vol. 8, no. 1-2, pp. 47–63, 1991.

[8] B. Kuipers, "The spatial semantic hierarchy," *Artificial Intelligence*, vol. 119, no. 1-2, pp. 191–233, 2000.

[9] R. Goeddel and E. Olson, "Learning semantic place labels from occupancy grids using cnns," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3999–4004, IEEE, 2016.

[10] Y. Katsumata, A. Taniguchi, Y. Hagiwara, and T. Taniguchi, "Semantic mapping based on spatial concepts for grounding words related to places in daily environments," *Frontiers in Robotics and AI*, vol. 6, p. 31, 2019.

[11] A. Pronobis, O. M. Mozos, B. Caputo, and P. Jensfelt, "Multi-modal semantic place classification," *International Journal of Robotics Research (IJRR), Special Issue on Robotic Vision*, vol. 29, pp. 298–320, Feb. 2010.

[12] N. Sünderhauf, F. Dayoub, S. McMahon, B. Talbot, R. Schulz, P. Corke, G. Wyeth, B. Upcroft, and M. Milford, "Place categorization and semantic mapping on a mobile robot," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5729–5736, IEEE, 2016.

[13] R. B. Rusu, Z. C. Marton, N. Blodow, A. Holzbach, and M. Beetz, "Model-based and learned semantic object labeling in 3d point cloud maps of kitchen environments," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3601–3608, 2009.

[14] A. Pronobis and P. Jensfelt, "Large-scale semantic mapping and reasoning with heterogeneous modalities," in *Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA)*, (Saint Paul, MN, USA), May 2012.

[15] A. Pronobis, *Semantic Mapping with Mobile Robots*. PhD thesis, KTH Royal Institute of Technology, Stockholm, Sweden, Dec. 2011.

[16] R. Dube, D. Dugas, E. Stumm, J. Nieto, R. Siegwart, and C. Cadena, "SegMatch: Segment based place recognition in 3d point clouds," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[17] J. McCormac, A. Handa, A. Davison, and S. Leutenegger, "Semanticfusion: Dense 3d semantic mapping with convolutional neural networks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4628–4635, IEEE, 2017.

[18] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: an open-source library for real-time metric-semantic localization and mapping," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1689–1696, IEEE, 2020.

[19] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. Kelly, and A. J. Davison, "Slam++: Simultaneous localisation and mapping at the level of objects," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1352–1359, 2013.

[20] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik, "Learning rich features from rgb-d images for object detection and segmentation," in *European Conference on Computer Vision*, pp. 345–360, Springer, 2014.

[21] J. C. d. C. S. Fernandes, "Semantic mapping with a mobile robot using a rgb-d camera," Master's thesis, Universidade de Coimbra, 2019.

[22] N. Sünderhauf, T. Pham, Y. Latif, M. Milford, and I. D. Reid, "Meaningful maps - object-oriented semantic mapping," *CoRR*, vol. abs/1609.07849, 2016.

[23] D. Maturana, *Semantic Mapping for Autonomous Navigation and Exploration*. PhD thesis, Carmegie Mellon], Pittsburgh, PA, August 2021.

[24] Z. Song, Q. Chen, Z. Huang, Y. Hua, and S. Yan, "Contextualizing object detection and classification," in *CVPR 2011*, pp. 1585–1592, 2011.

[25] S. Routray, A. K. Ray, and C. Mishra, "Analysis of various image feature extraction methods against noisy image: Sift, surf and hog," in *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pp. 1–5, 2017.

[26] S. Srivastava, "SIFT vs SURF: quantifying the variation in transformations," *CoRR*, vol. abs/1504.06740, 2015.

[27] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 580–587, 2014.

[28] R. Girshick, "Fast r-cnn," *CoRR*, vol. abs/1504.08083, 2015.

[29] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2016.

[30] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[31] J. Redmon, "Yolo v3: Real-time object detection." `https://pjreddie.com/darknet/yolo/`, 2022. Last accessed November 2022.

[32] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," 2016.

[33] A. Khalfaoui, A. Badri, and I. E. Mourabit, "Comparative study of yolov3 and yolov5's performances for real-time person detection," in *2022 2nd International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET)*, pp. 1–5, 2022.

[34] M. Z. Saeed Shafieian, "Multi-layer stacking ensemble learners for low footprint network intrusion detection," *Complex & Intelligent Systems*, vol. 1007, pp. 2198–6053, 2022.

[35] L. Breiman, "Bagging Predictors," *Machine Learning*, vol. 24, pp. 123–140, 1996.

[36] J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua, and A. Lopez, "A comprehensive survey on support vector machine classification: Applications, challenges and trends," *Neurocomputing*, vol. 408, pp. 189–215, 2020.

[37] T. F. Rathbun, S. K. Rogers, M. P. DeSimio, and M. E. Oxley, "Mlp iterative construction algorithm," *Neurocomputing*, vol. 17, no. 3, pp. 195–216, 1997.

[38] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent, "Efficient sparse pose adjustment for 2d mapping," *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 22–29, 2010.

[39] S. Macenski and I. Jambrecic, "Slam toolbox: Slam for the dynamic world," *Journal of Open Source Software*, vol. 6, no. 61, p. 2783, 2021.

[40] C. Campos, R. Elvira, J. J. Gomez, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, 2021.

[41] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, (Shanghai, China), IEEE, May 9-13 2011.

[42] R. B. Rusu and S. Cousins, "Point cloud library basic structures." `https://pointclouds.org/documentation/tutorials/basic_structures.html`, 2023. Last accessed July 2023.

[43] Y. Zhou, H. Lu, G. Wang, J. Wang, and W. Li, "Voxelization modelling based finite element simulation and process parameter optimization for fused filament fabrication," *Materials & Design*, vol. 187, p. 108409, 2020.

[44] R. Rusu, "Semantic 3d object maps for everyday manipulation in human living environments," 2009.

[45] "Euclidean cluster extraction." `https://adioshun.gitbooks.io/pcl/content/Tutorial/Segmentation/euclidean-cluster-extraction-pcl-python.html`, 2023. Last accessed July 2023.

[46] C. E. Johnson and B. Kuipers, *Topological Mapping and Navigation in Real-World Environments*. Ph.D. dissertation, Univ. Michigan, 2018.

[47] M. Opiela and F. Galčík, "Grid-based bayesian filtering methods for pedestrian dead reckoning indoor positioning using smartphones," *Sensors*, vol. 20, no. 18, 2020.

[48] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, (Red Hook, NY, USA), p. 1097–1105, Curran Associates Inc., 2012.

[49] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba, "Places: A 10 million image database for scene recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 6, pp. 1452–1464, 2018.

[50] R. Yoshihashi, W. Shao, R. Kawakami, S. You, M. Iida, and T. Naemura, "Classification-reconstruction learning for open-set recognition," *CoRR*, vol. abs/1812.04246, 2018.

[51] S. Särkkä, *Bayesian Filtering and Smoothing*. Bayesian Filtering and Smoothing, Cambridge University Press, 2013.

[52] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: an efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, pp. 189 – 206, 2013.

[53] F. Auer, V. Lenarduzzi, M. Felderer, and D. Taibi, "From monolithic systems to microservices: An assessment framework," *Information and Software Technology*, vol. 137, p. 106600, 2021.

[54] S. Macenski, "Slam toolbox." `https://github.com/SteveMacenski/slam_toolbox`, 2023. Last accessed July 2023.

[55] S. Macenski, "Navigation 2." `https://github.com/ros-planning/navigation2`, 2023. Last accessed July 2023.

[56] S. Macenski, F. Martín, R. White, and J. Ginés Clavero, "The marathon 2: A navigation system," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[57] A. Merzlyakov and S. Macenski, "A comparison of modern general-purpose visual slam approaches," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[58] S. Macenski, S. Singh, F. Martin, and J. Gines, "Regulated pure pursuit for robot path tracking," *Autonomous Robots*, 2023.

[59] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent, "Efficient sparse pose adjustment for 2d mapping," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 22–29, 2010.

[60] G. Valiente, "Adjacency maps and efficient graph algorithms," *Algorithms*, vol. 15, no. 2, 2022.

[61] P. Silva and R. Rocha, "Low-power footprint inference with a deep neural network offloaded to a service robot through edge computing," in *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, SAC '23, (New York, NY, USA), p. 800–807, Association for Computing Machinery, 2023.

[62] P. Silva and R. Rocha, *Edge Computing to Enable Onboard Computation of Deep Learning Algorithms in Service Robots*. M.Sc. Thesis, University of Coimbra, 2022.

[63] J. Pérez, J. Díaz, J. Berrocal, R. López-Viana, and A. González-Prieto, "Edge computing: A grounded theory study," *Computing*, vol. 104, p. 2711–2747, dec 2022.

Appendix A

Complete system pipeline

The diagram shown in Figure A.1 presents the complete system pipeline. It shows all the components necessary to perform semantic mapping, from *Data Collection*, to *Semantic Pipeline*, and finally the *Map*. An overall description of the Semantic Pipeline will be presented in the topics below, and the in-depth details are discussed in Chapter 3 (page 13).

Backbone

The *backbone* is where the features are extracted from the data (see Section 3.2, page 14); in this case, the input data is an *RGB Image* that is processed by an image detector. The image detector extracts multiple objects from the image and outputs the matrix ($\mathbf{D}$); in which each line represents a detected object. Objects are represented by a combination of the *vector of class labels* ($\mathbf{x}$), and a 2D AABB; ($\mathbf{x}$) has length (N) equal to the number of classes know by the detector. This stage can be performed by 1 or more detectors with a different set of classes working in parallel as mentioned in (see Section 4.1, page 39).

Neck

The *neck* is where features are transformed and refined; in this case there are 2 parallel processes, *Geometric Segmentation* (Section 3.3, page 15) and *Point Cloud Binning* (see Section 3.4, page 23). The first one, inputs the matrix ($\mathbf{D}$) and estimates a pair of 3D AABB and *Bounding Box Confidence* (BB_c) for each object; the output is the matrix ($\mathbf{D}^*$) which is the composition of ($\mathbf{x}$), 3D AABB and (BB_c). The second one, *Point Cloud Binning*, downsamples the relevant aspects of the *Point Cloud* into a *Binned Depth Map* (BDM).

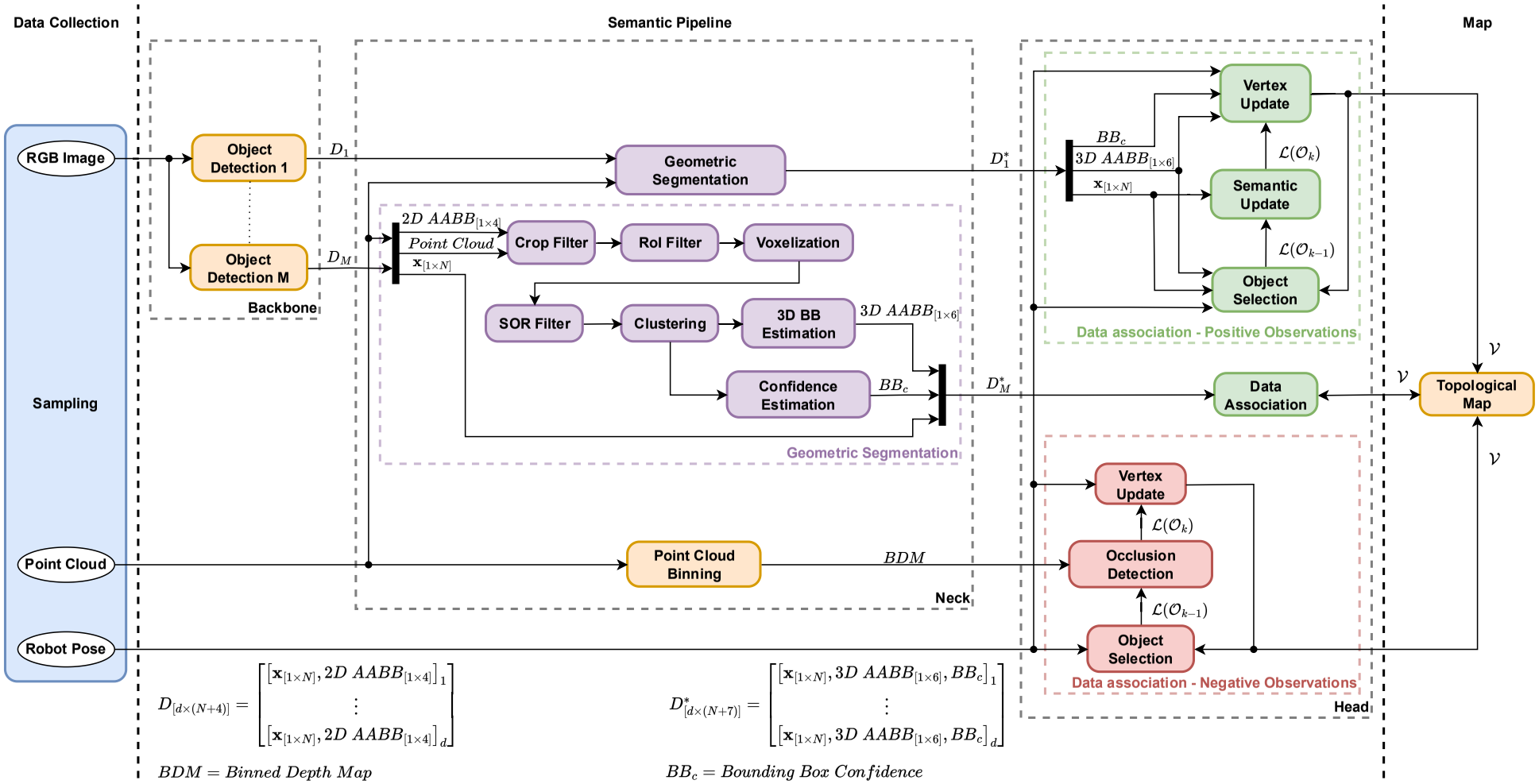

Figure A.1: Detailed system diagram.

Head

The *head* is where the main task of the model is performed; in this case, it is the *semantic mapping*. The *semantic mapping* is built using associations of acquired features with their relative positions in a metric map; the associations are divided into 2 groups that are processed in parallel; association of *positive observations* (Subsection 3.5.4, page 34) and associations of *negative observations* (Subsection 3.5.5, page 35). The positive ones use the matrix ($\mathbf{D}^*$) to create and update instances of objects in the map; the second one utilizes the BDM to validate the expected positions of mapped objects and detect cases of object occlusion and object absence.

Appendix B

Metrics

The majority of the formulae that compose this appendix were extracted from Scikit Learn[1] web page.

Notation

- TP - True Positive
- TN - True Negative
- FP - False Positive
- FN - False Negative
- REC - Recall = TPr
- TPr - True Positive Rate
- TNr - True Negative Rate
- FPr - False Positive Rate
- FNr - False Negative Rate
- SPE - Specificity = TNr

B.1 TPr, TNr, FPr, FNr

$$TPr = \frac{TP}{Positives} = \frac{TP}{TP + FN} \quad \text{(B.1)} \qquad FPr = \frac{FP}{Negatives} = \frac{FP}{FP + TN} \quad \text{(B.3)}$$

$$TNr = \frac{TN}{Negatives} = \frac{TN}{TN + FP} \quad \text{(B.2)} \qquad FNr = \frac{FN}{Positives} = \frac{FN}{FN + TP} \quad \text{(B.4)}$$

[1]Scikit learn: https://scikit-learn.org/stable/modules/model_evaluation.html

B.2 Accuracy (ACC)

Accuracy is how close the measurement is to its true value.

$$Accuracy = \underbrace{\frac{Correct\ Classifications}{All\ Classifications}}_{Multi\ Class\ Classification} = \underbrace{\frac{TP+TN}{TP+TN+FP+FN}}_{Binary\ Classification} \tag{B.5}$$

B.3 Precision (PRE)

Precision is how close the measurements are to each other.

$$Precision = \frac{TP}{TP+FP} \tag{B.6}$$

B.4 Average Precision (AP)

Average precision is the area under the precision-recall curve. ($p - precision$, $r - recall$)

$$AP = \int_{r=0}^{1} p(r)dr \tag{B.7}$$

That can be approximated using:

$$AP = \sum_{k=1}^{N} (REC(k) - REC(k-1)) \cdot PRE(k) \tag{B.8}$$

B.5 Mean Average Precision (MAP)

Is the mean of all classes AP.

$$AP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{B.9}$$

B.6 Intersection over Union (IoU)

IoU is a relationship between the ground truth, and prediction bounding boxes; it is defined by the division of the area of intersection by the area of union. Figure B.6.1 illustrates this metric.

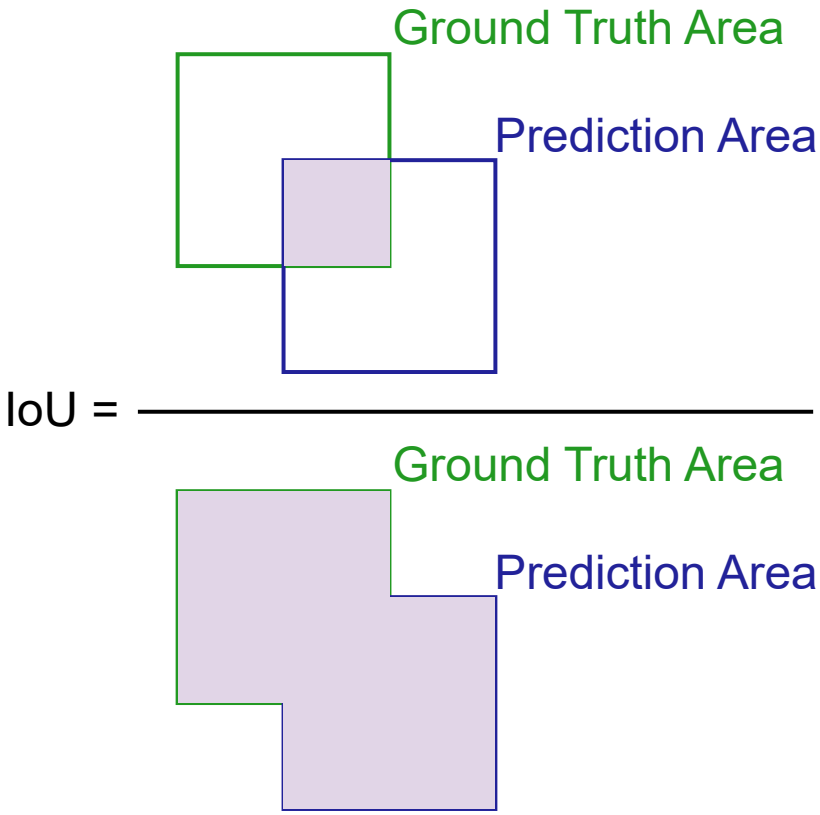

Figure B.6.1: Intersection over union.

Appendix C

Additional Images

C.1 Geometric Segmentation Pipeline

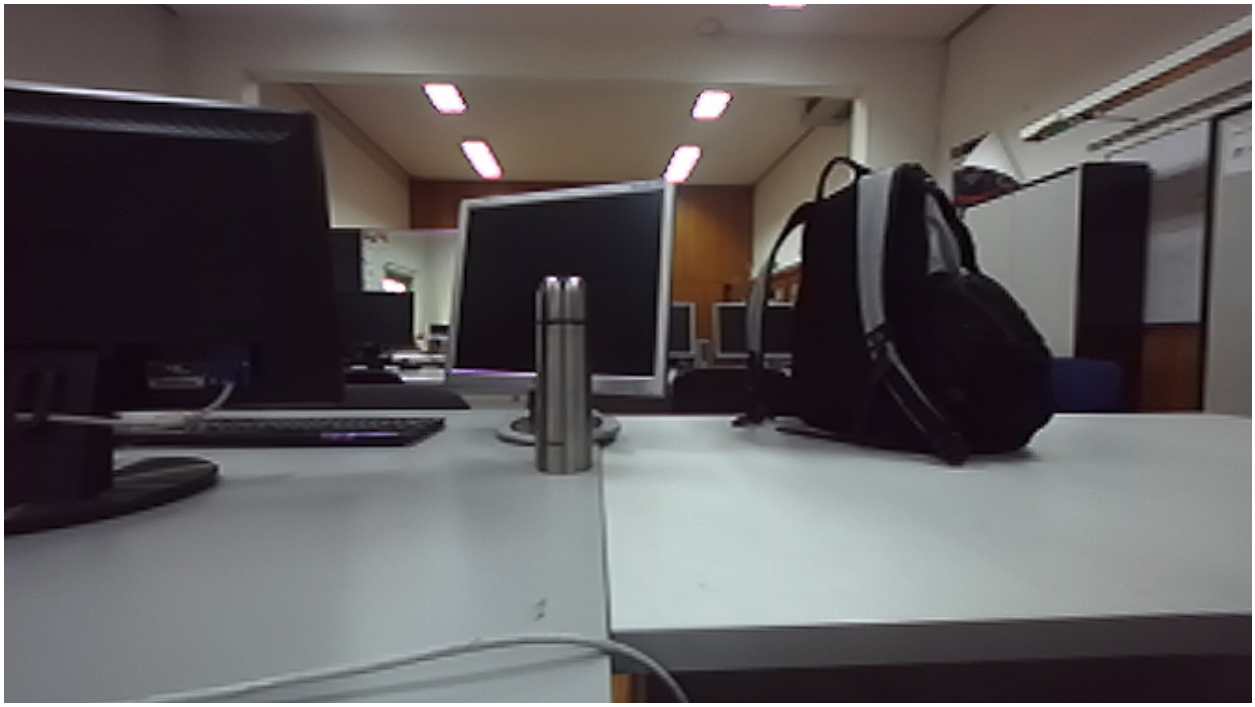

Figure C.1.1: Reference image.

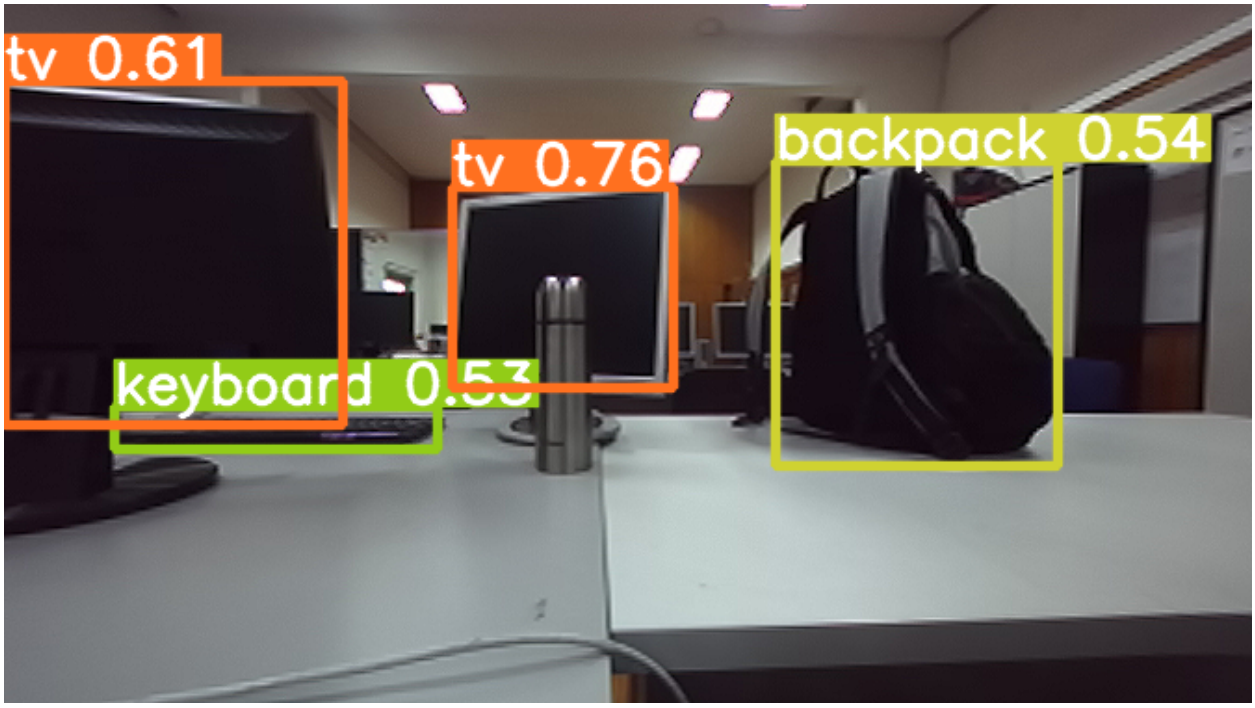

Figure C.1.2: YOLOv5 detections.

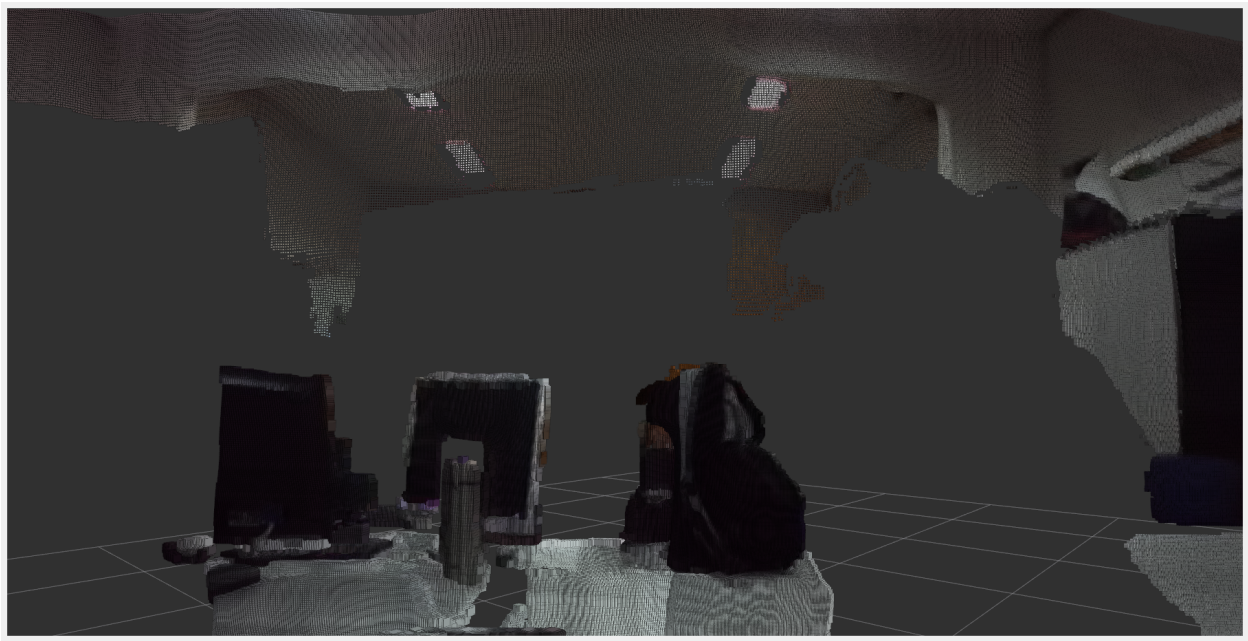

Figure C.1.3: Reference point cloud.

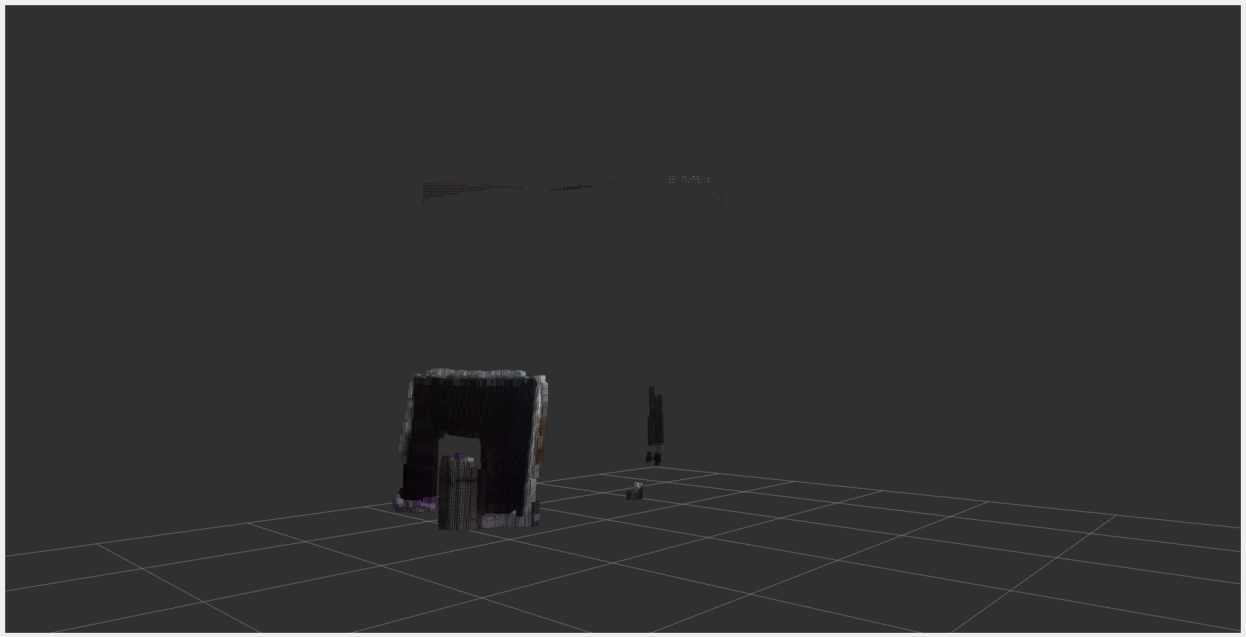

Figure C.1.4: Object segmentation steps: Crop Filter.

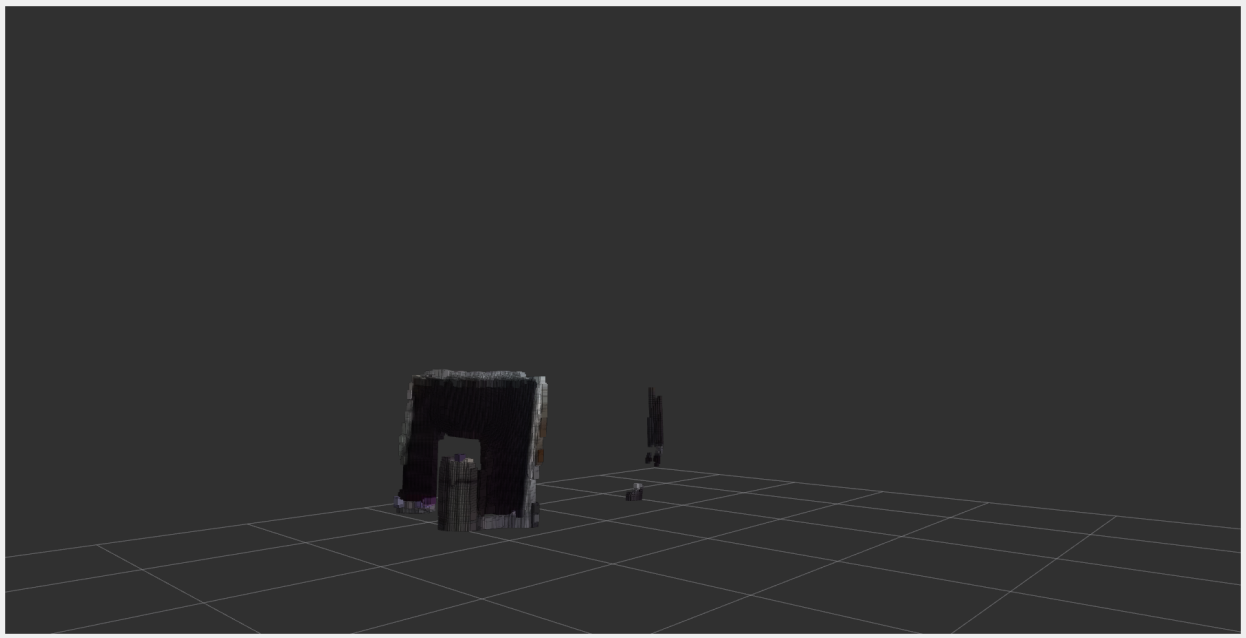

Figure C.1.5: Object segmentation steps: Crop Filter, RoI filter.

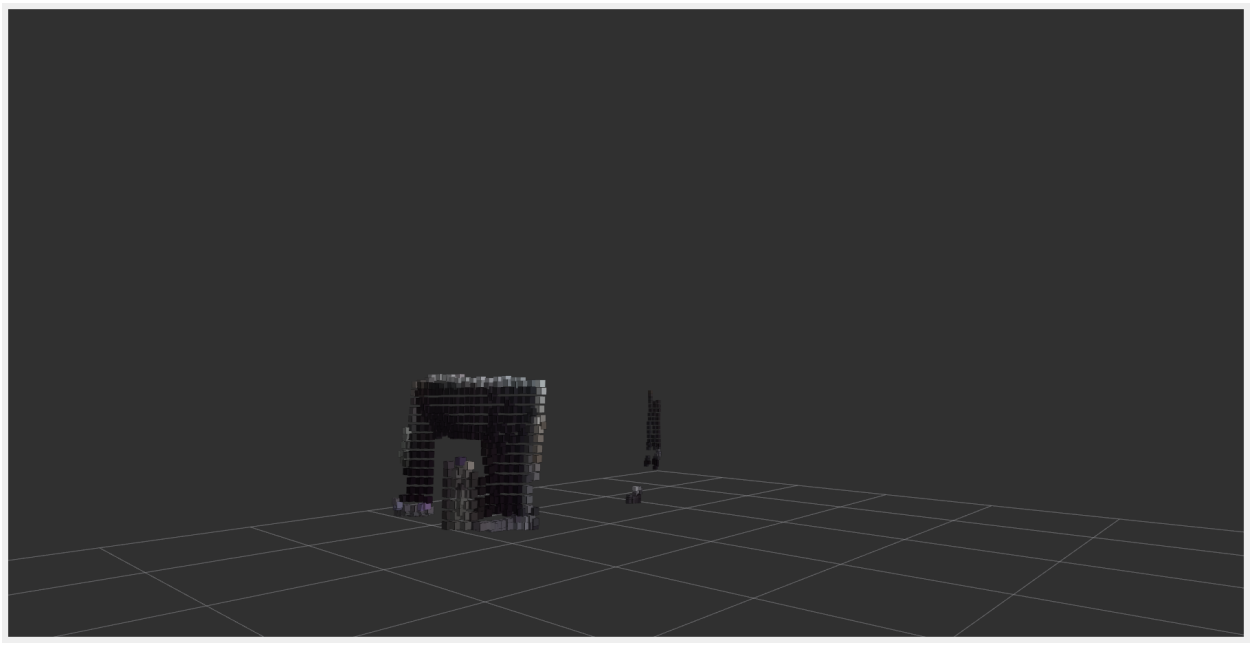

Figure C.1.6: Object segmentation steps: Crop Filter, ROI filter, Voxelization.

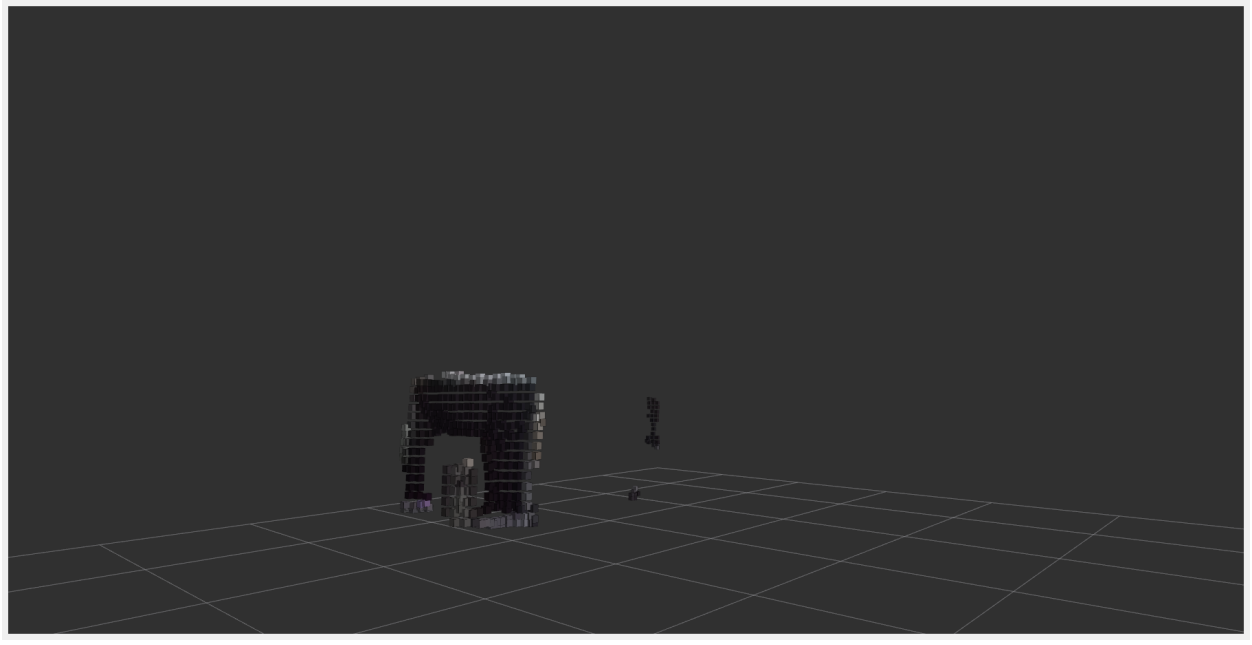

Figure C.1.7: Object segmentation steps: Crop Filter, ROI filter, Voxelization, SOF filter.

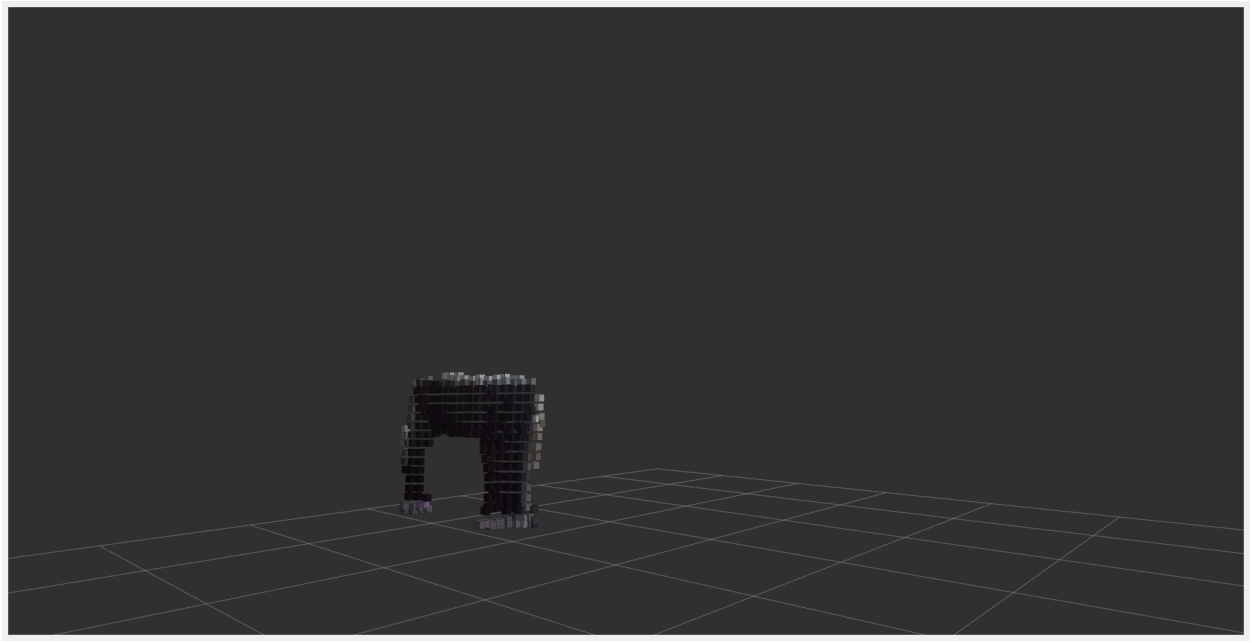

Figure C.1.8: Object Segmentation steps: Crop Filter, ROI filter, Voxelization, SOR filter, Clustering.

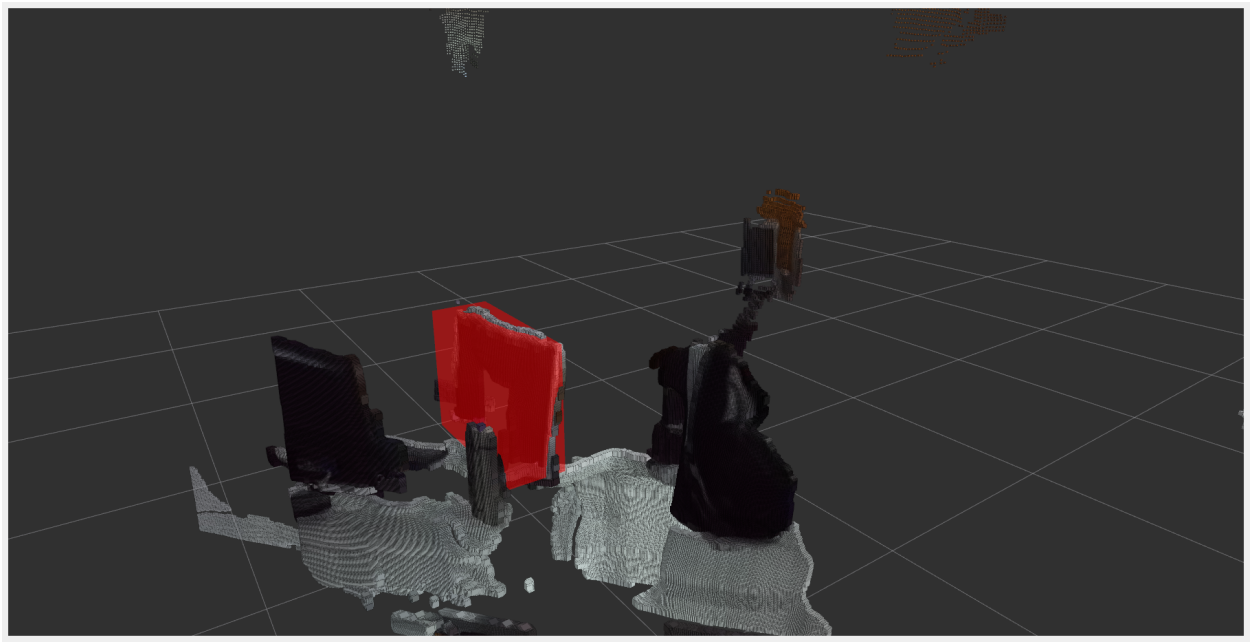

Figure C.1.9: Object segmentation steps: Crop Filter, ROI filter, Voxelization, SOF filter, Clustering, 3D AABB Estimation.

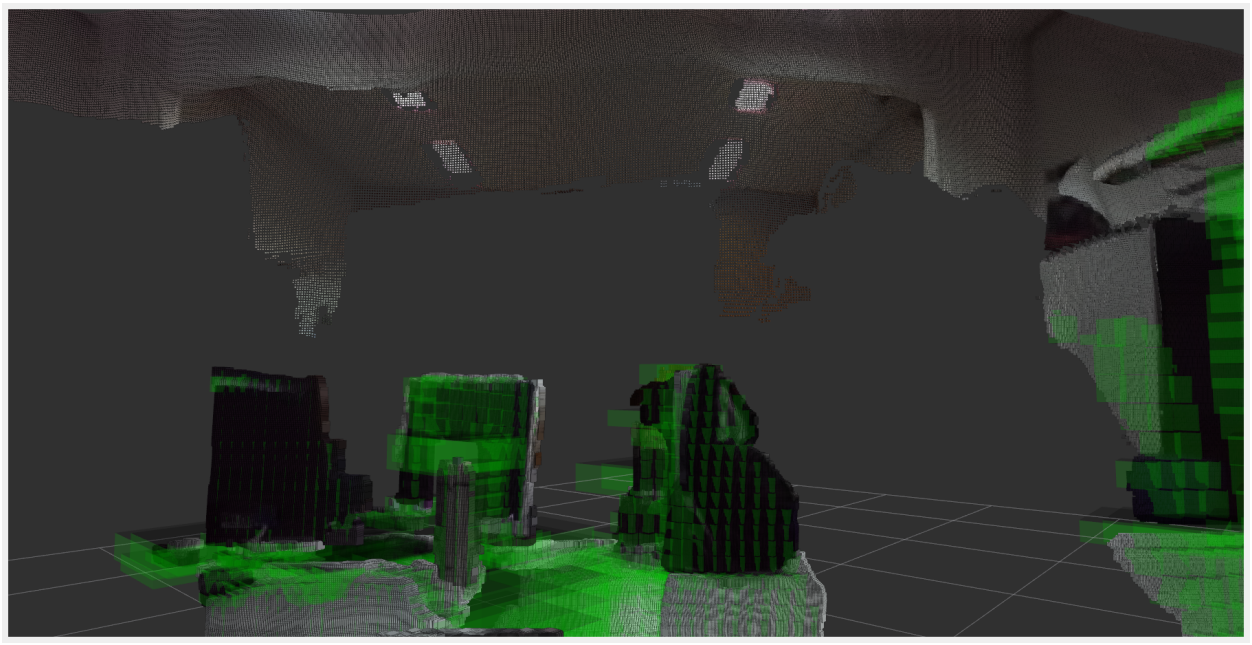

Figure C.1.10: Binned depth map view 1.

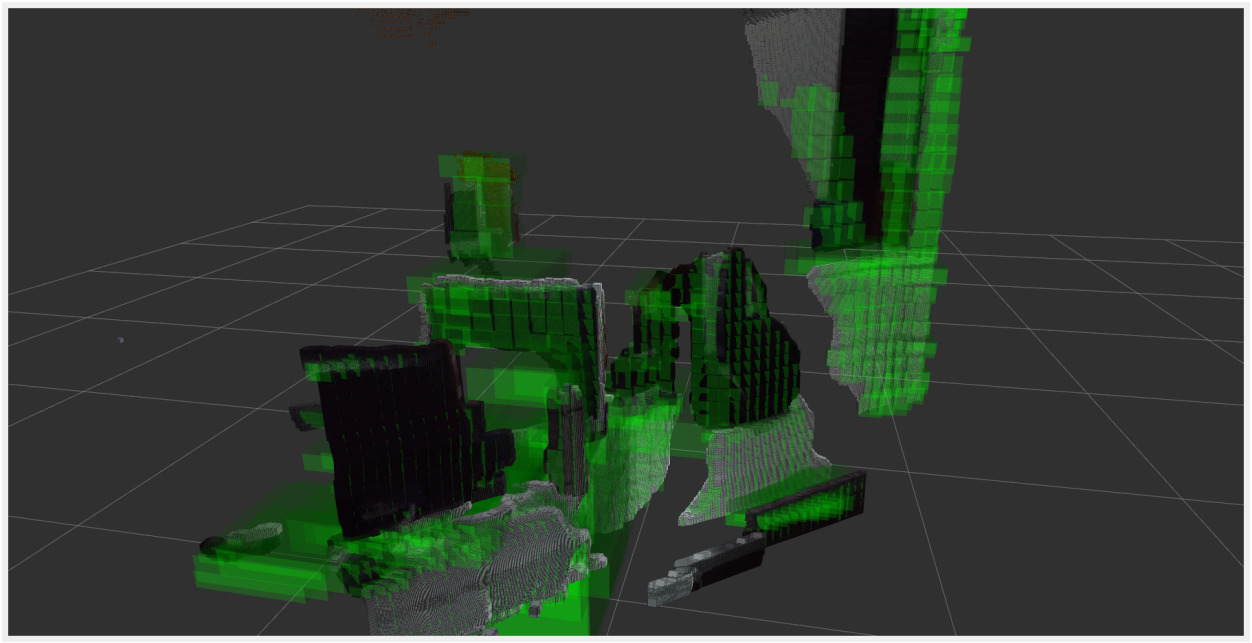

Figure C.1.11: Binned depth map view 2.

C.2 RVIZ SMAP Map Representation

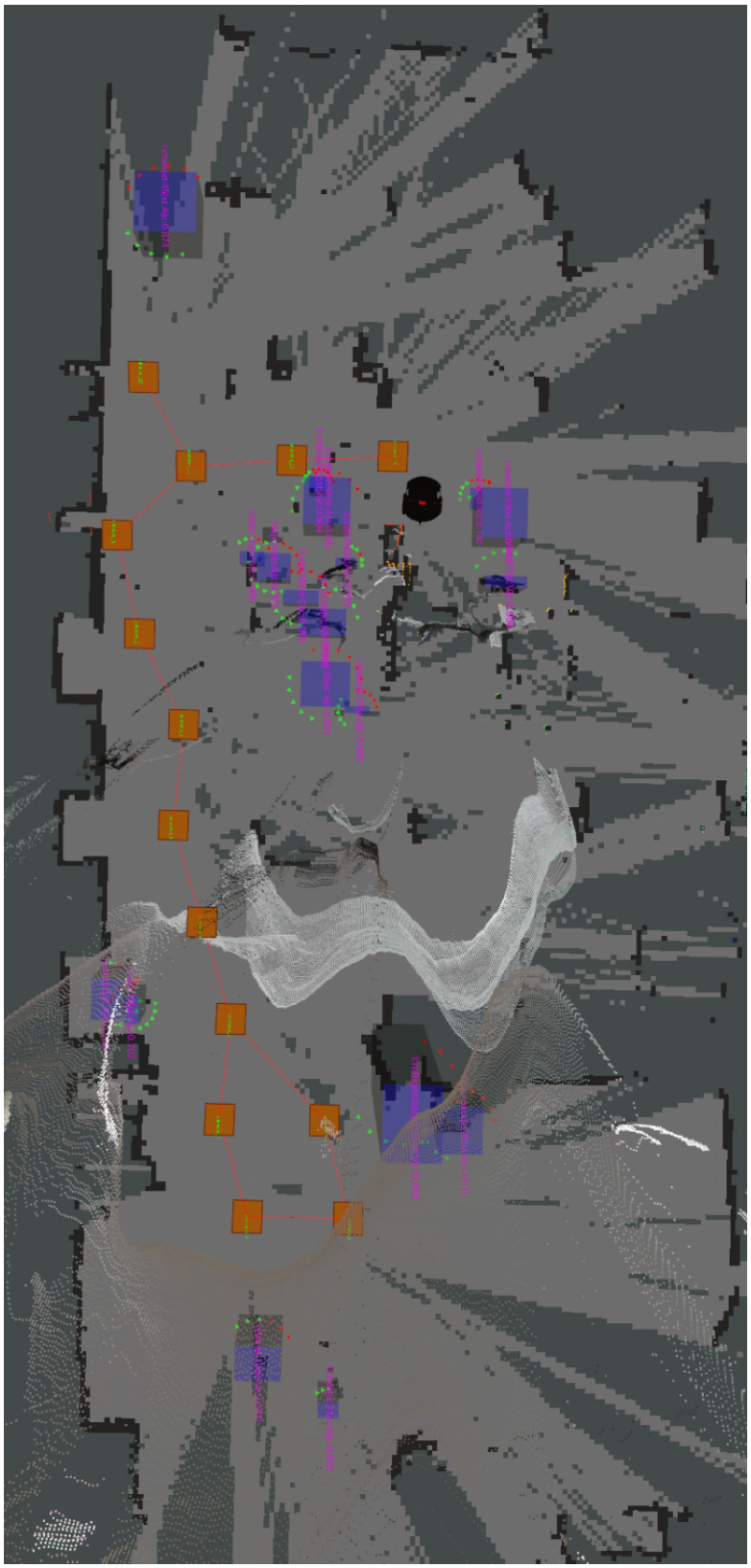

Figure C.2.12: SMAP map of the MRL. The map shows the precise location (see Table 5.2, page 53), and good volume estimation of the entities analyzed. The map corresponds to the region illustrated by Figures C.4.26, C.4.27 and C.4.29 (page 92). Figures 5.2 and 5.3 explain the indicators present in the figure.

C.3 SMAP Grid Representation

Figure C.3.13: Occupancy grid ($0.05m/cell$).

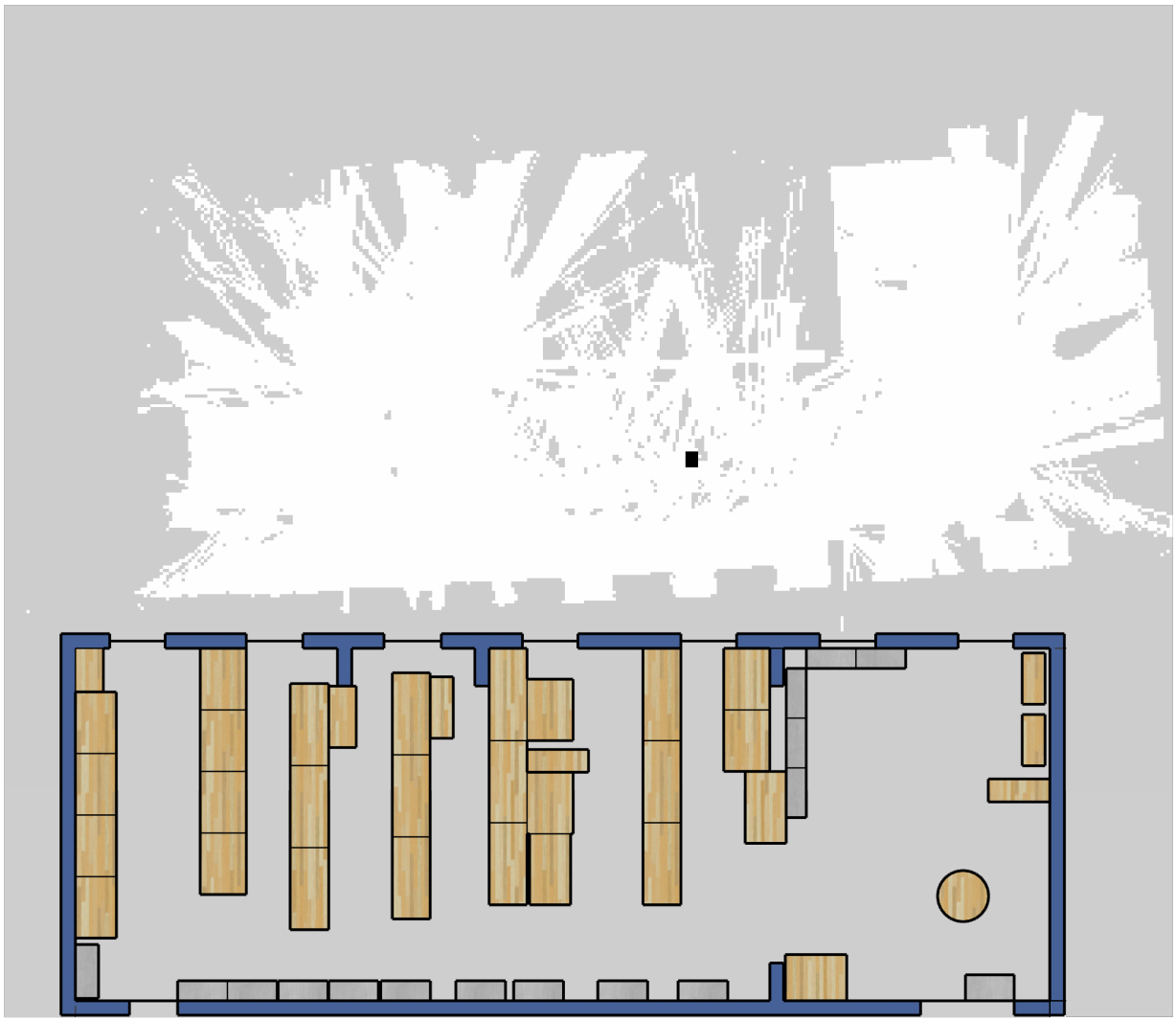

Figure C.3.14: SMAP 2D projection class *Backpack* ($0.05m/cell$).

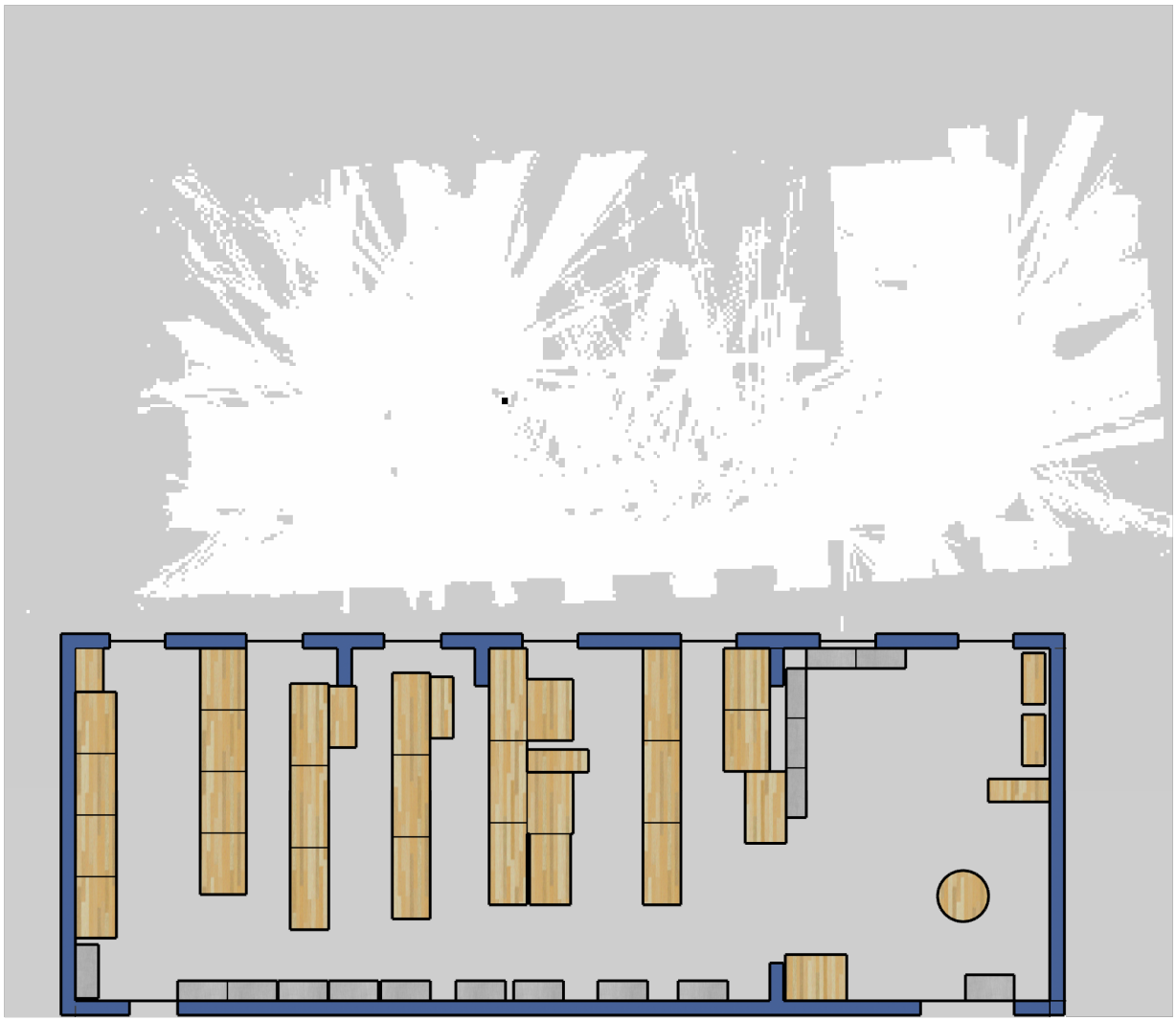

Figure C.3.15: SMAP 2D projection class *Bottle* ($0.05m/cell$).

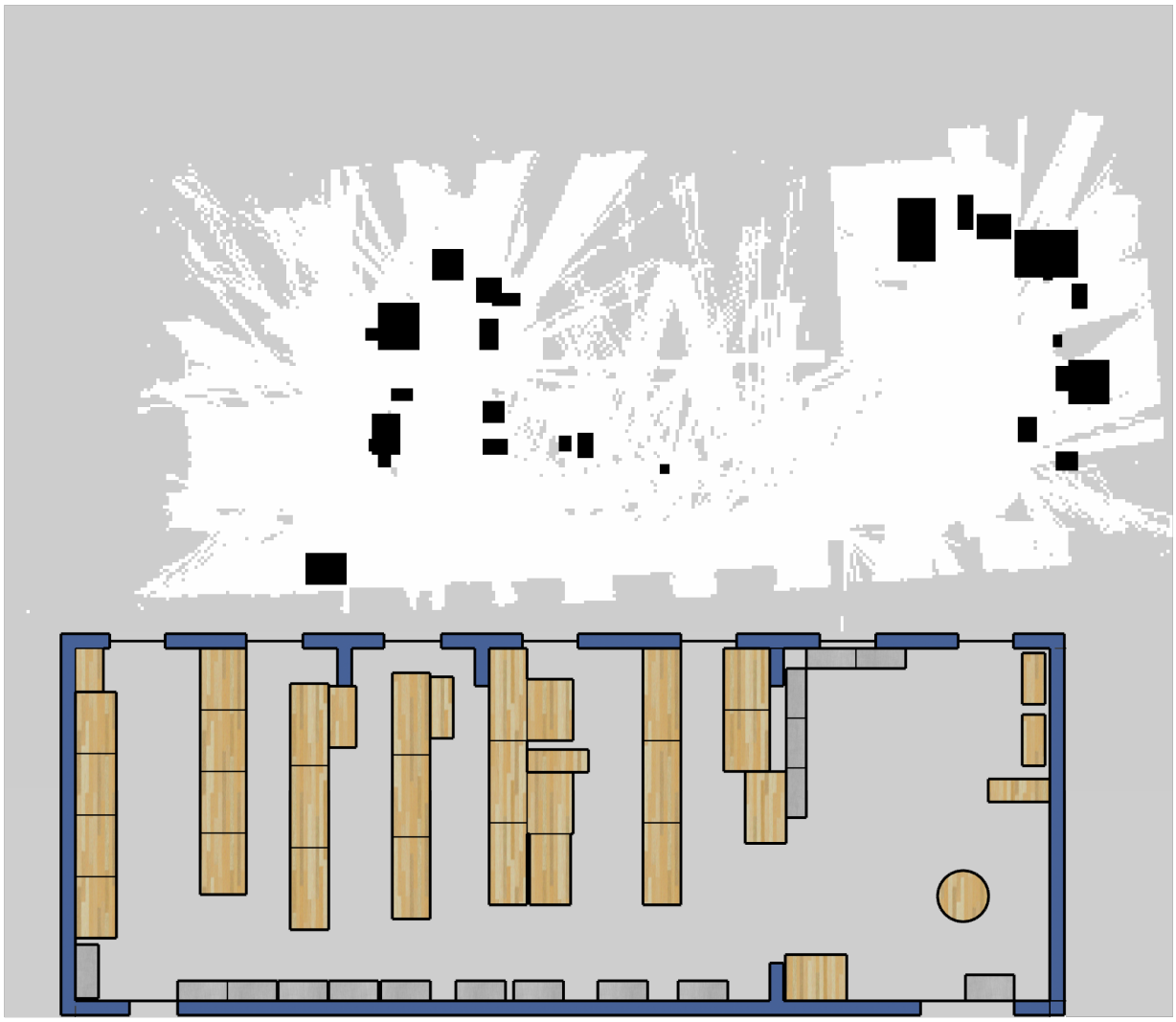

Figure C.3.16: SMAP 2D projection class *Chair* ($0.05m/cell$).

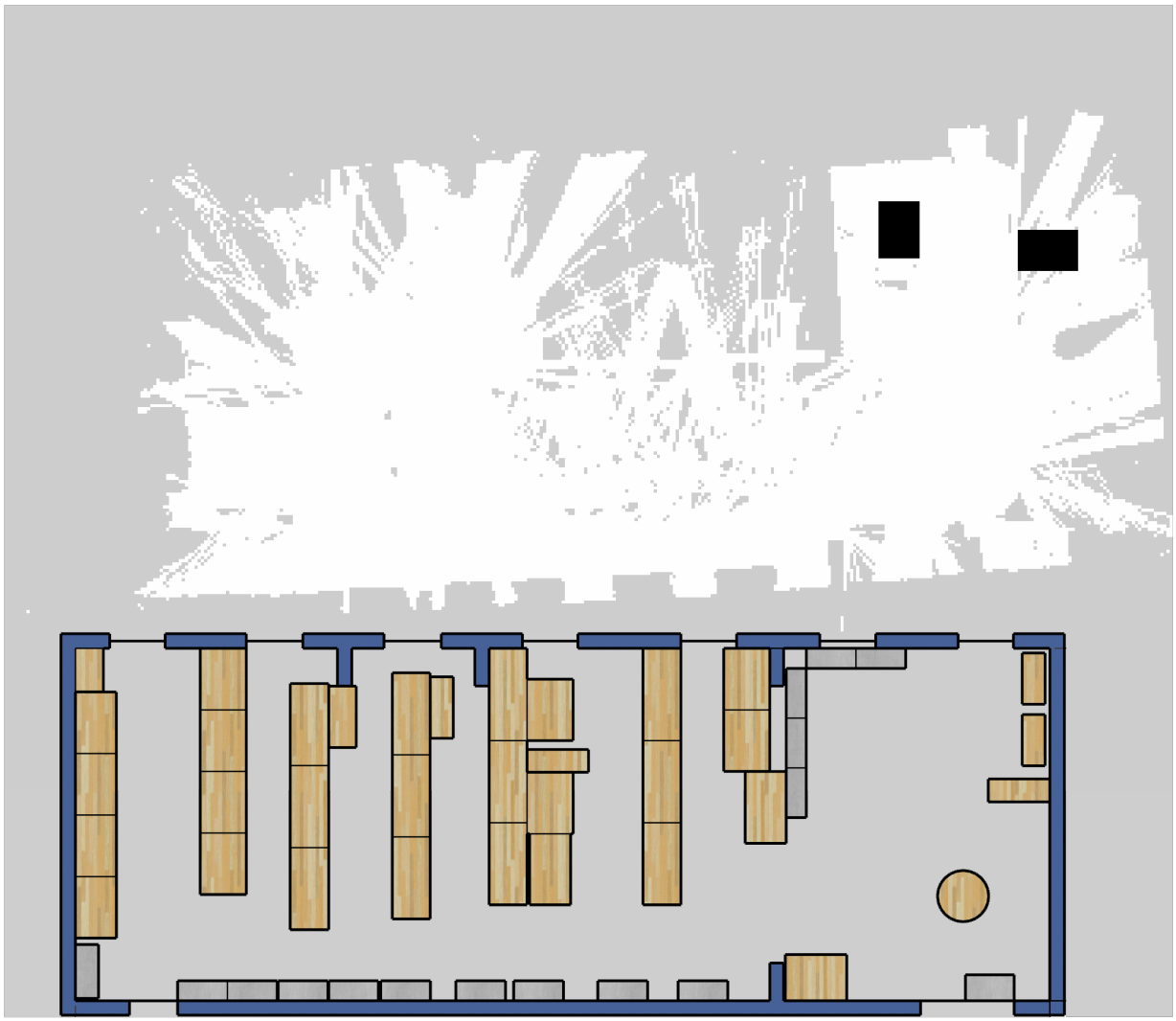

Figure C.3.17: SMAP 2D projection class *Couch* ($0.05m/cell$).

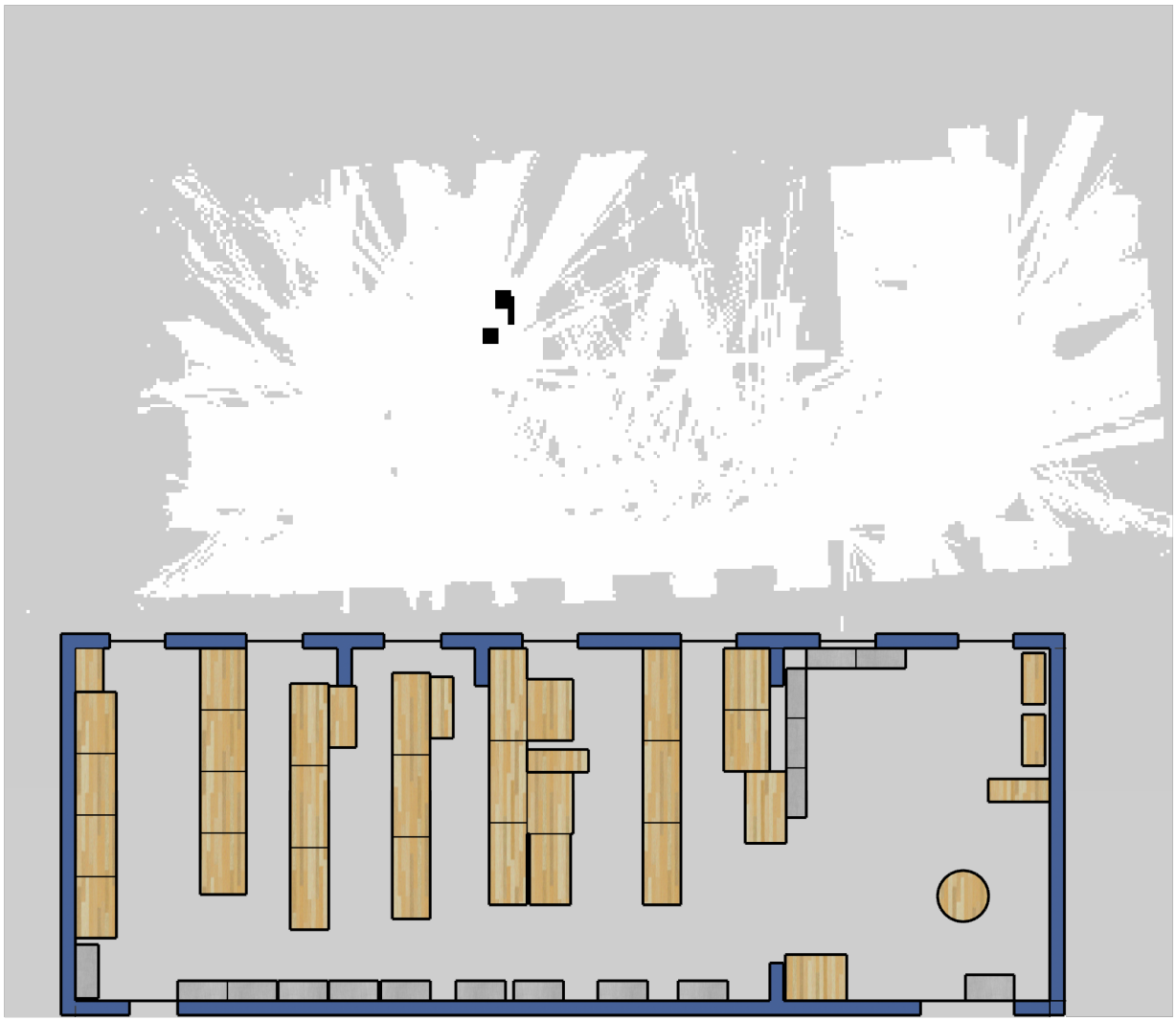

Figure C.3.18: SMAP 2D projection class *Keyboard* ($0.05m/cell$).

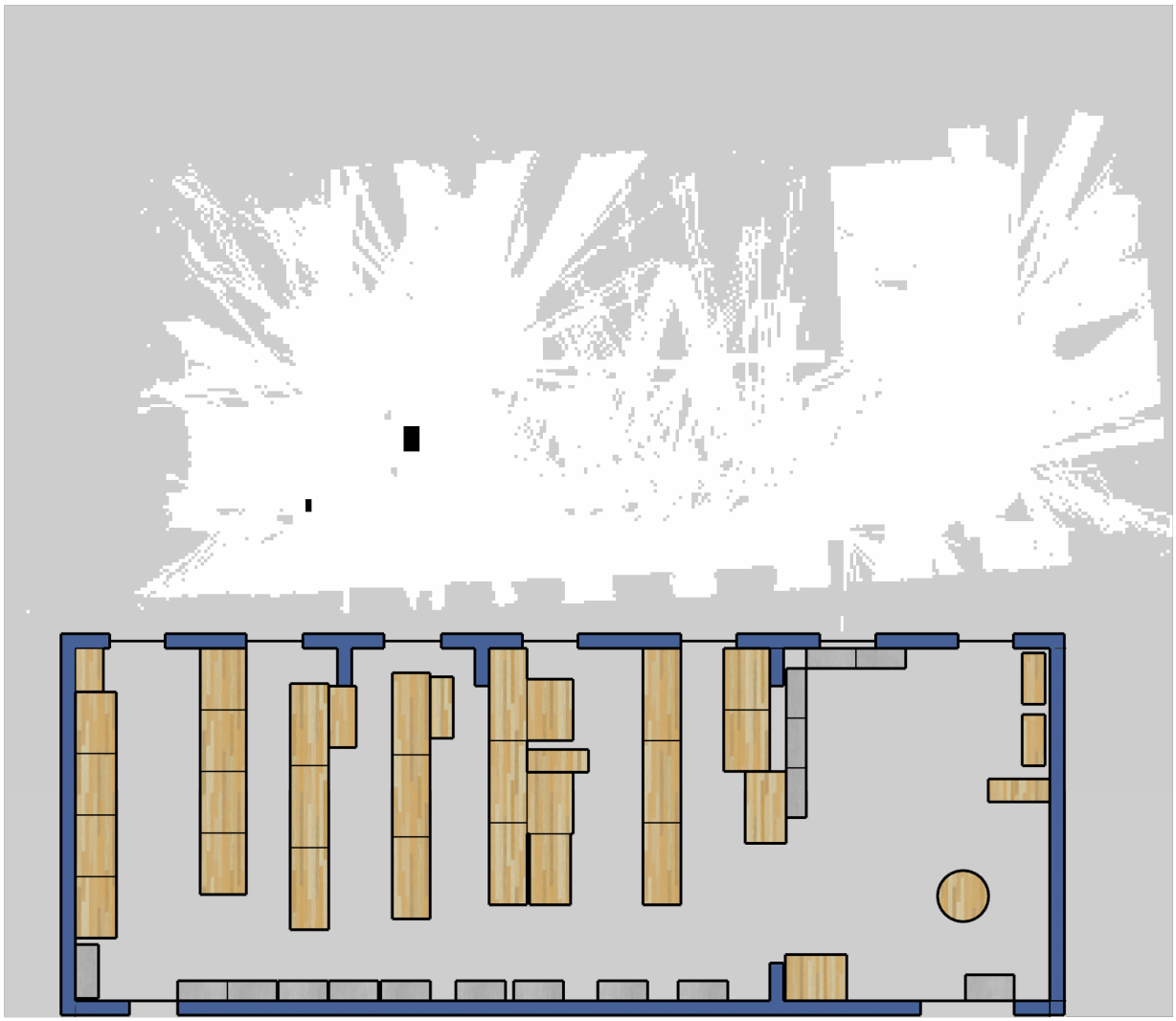

Figure C.3.19: SMAP 2D projection class *Suitcase* ($0.05m/cell$).

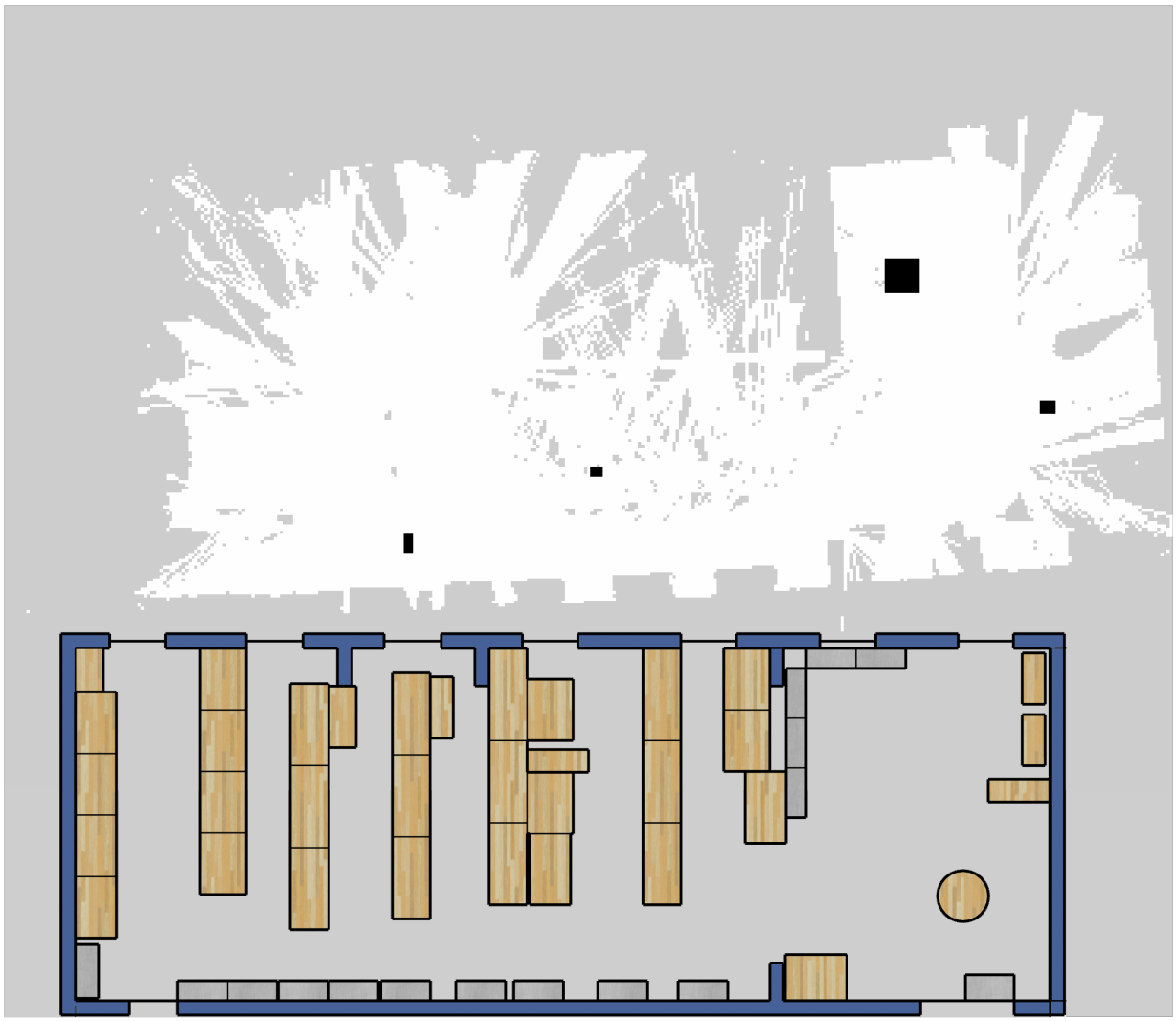

Figure C.3.20: SMAP 2D projection class *Table* ($0.05m/cell$).

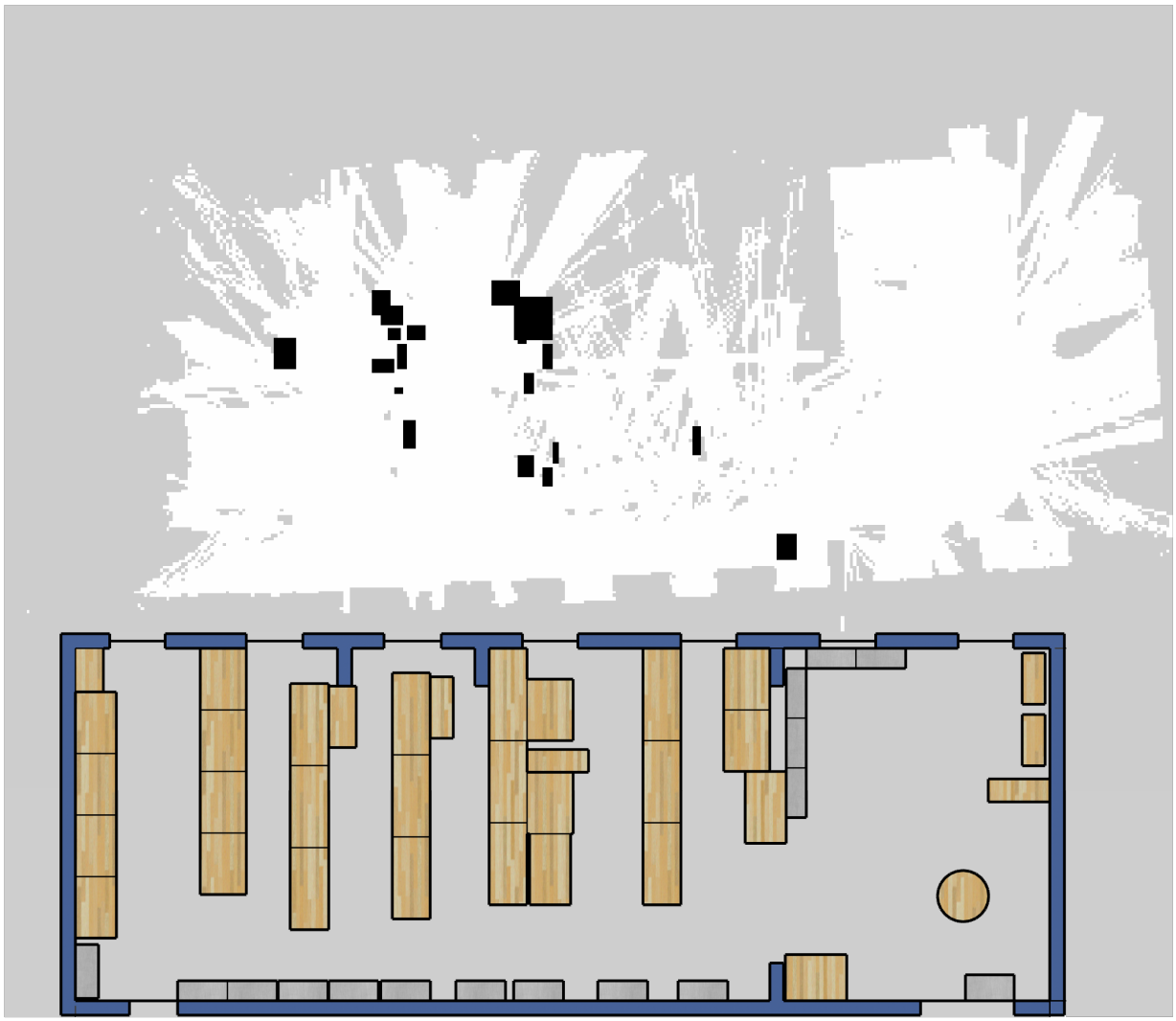

Figure C.3.21: SMAP 2D projection class TV ($0.05m/cell$).

Figure C.3.22: SMAP 2D projection class *Umbrella* ($0.05m/cell$).

C.4 Mobile Robotics Lab (MRL)

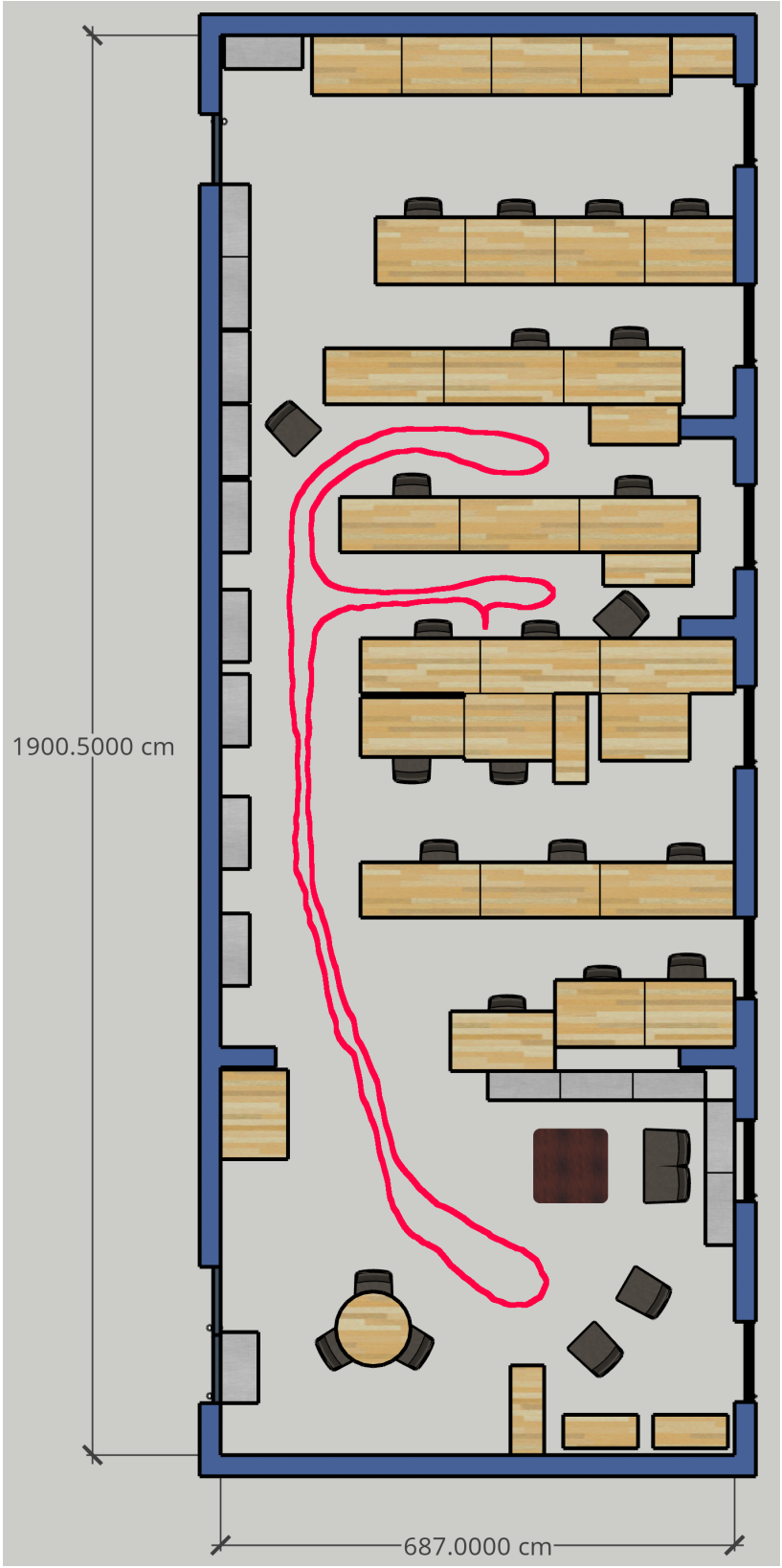

Figure C.4.23: Mobile Robotics Lab (MRL) plant. The red trace is the path traveled by the robot in experiment 1 referred to in Section 5 (page 45). The robot starts and finishes in the same location between two working stations.

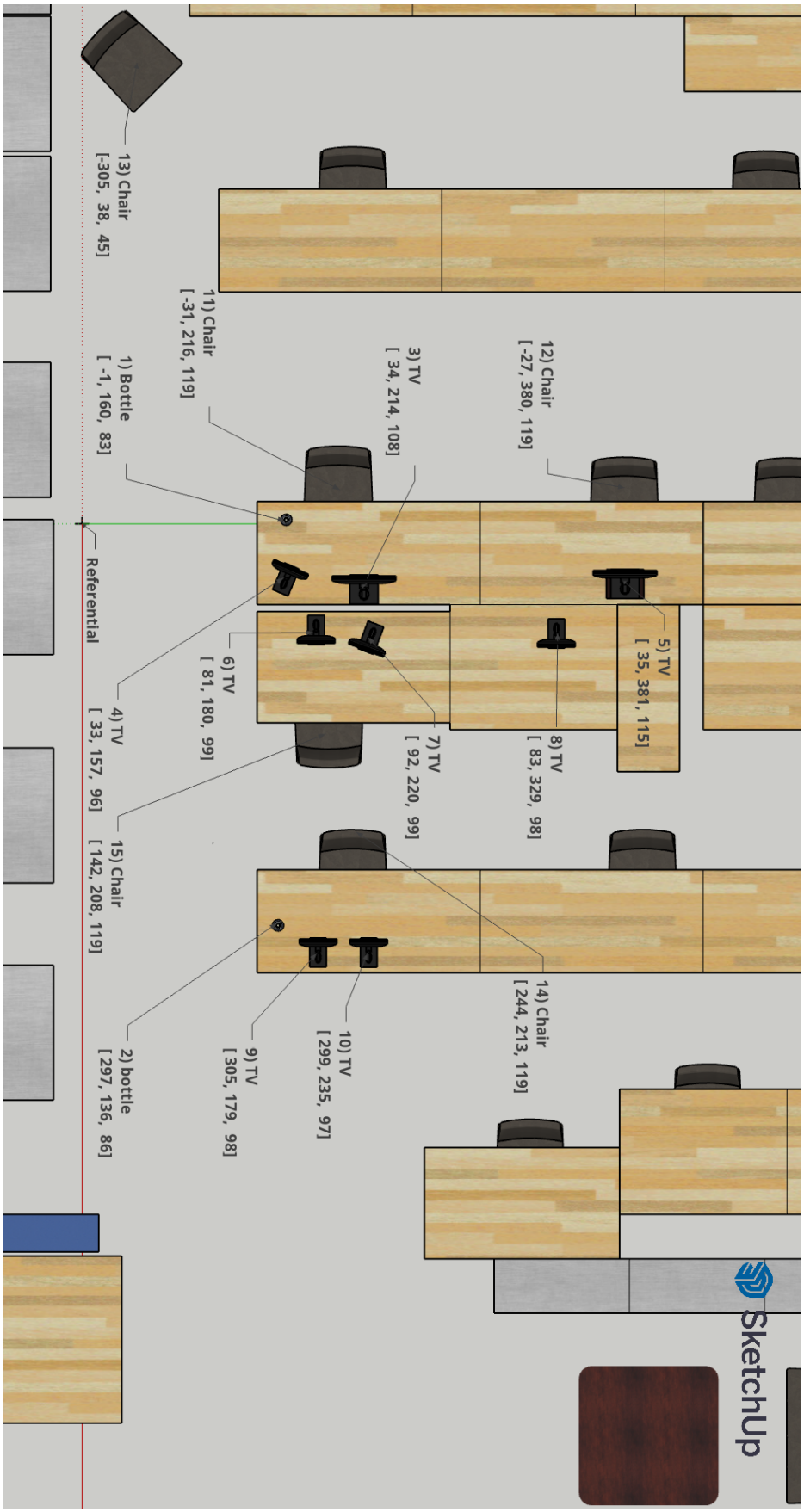

Figure C.4.24: Mobile Robotics Lab (MRL) object locations. Figures C.4.26, C.4.27 and C.4.29 illustrates the correspondent location in MRL.

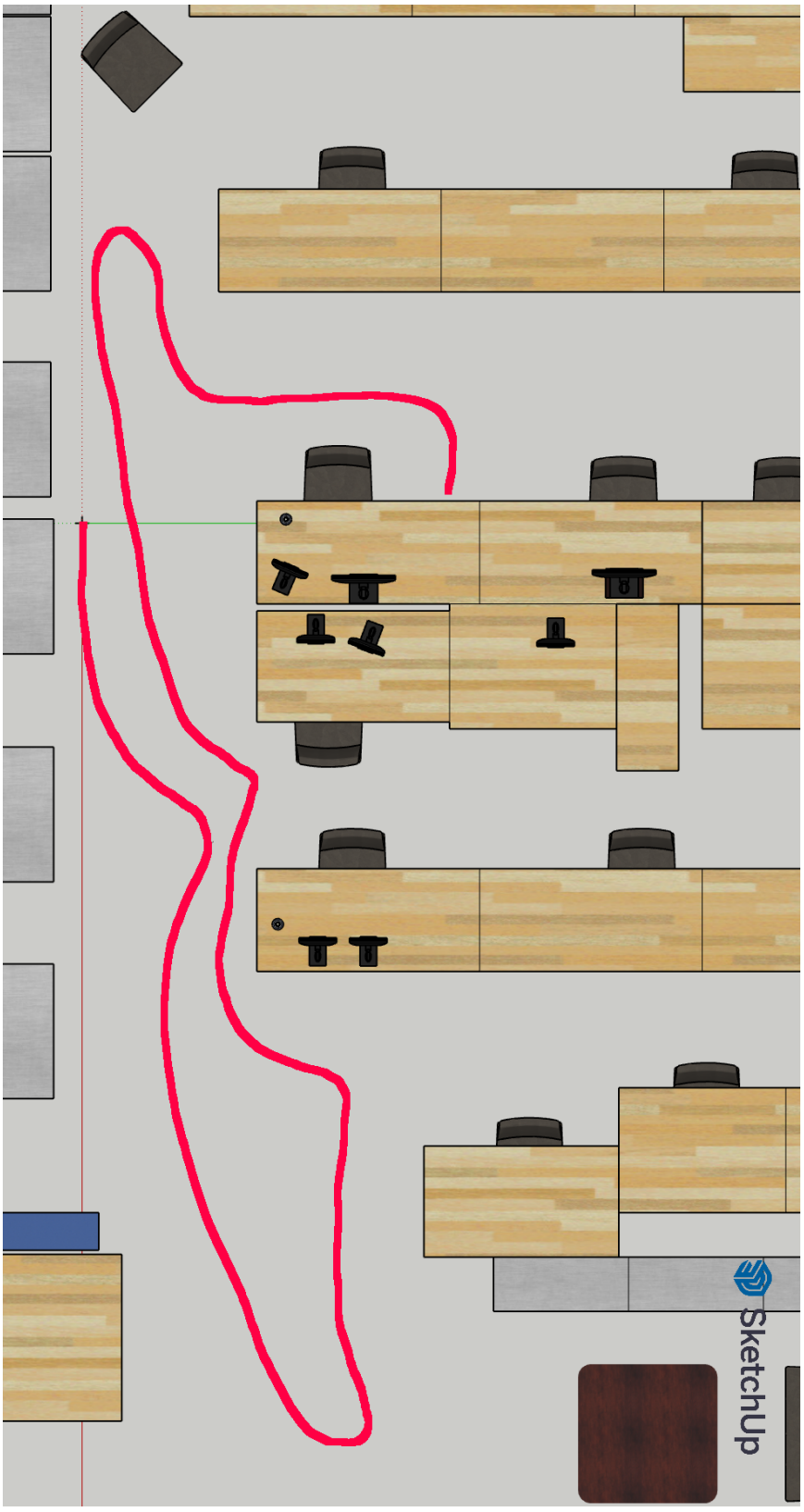

Figure C.4.25: Experiment 2 path. The red trace is the path traveled by the robot in experiment 2 referred to in Section 5 (page 45). The robot starts near a cabinet and finishes between two working stations.

Figure C.4.26: Mobile Robotics Lab (MRL) photo 1.

Figure C.4.27: Mobile Robotics Lab (MRL) photo 2.

Figure C.4.28: Mobile Robotics Lab (MRL) photo 3.

Figure C.4.29: Mobile Robotics Lab (MRL) photo 4.

C.5 ROS Diagrams

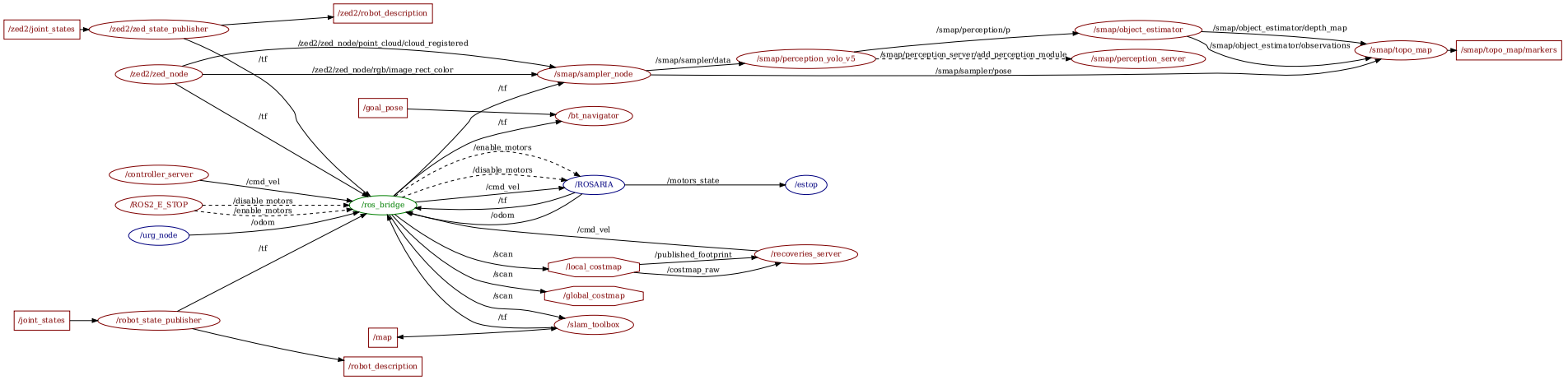

Figure C.5.30: ROS node diagram. Blue nodes belong to ROS Noetic, red nodes represent ROS Foxy. Solid lines represent topics and dashed lines represent services.

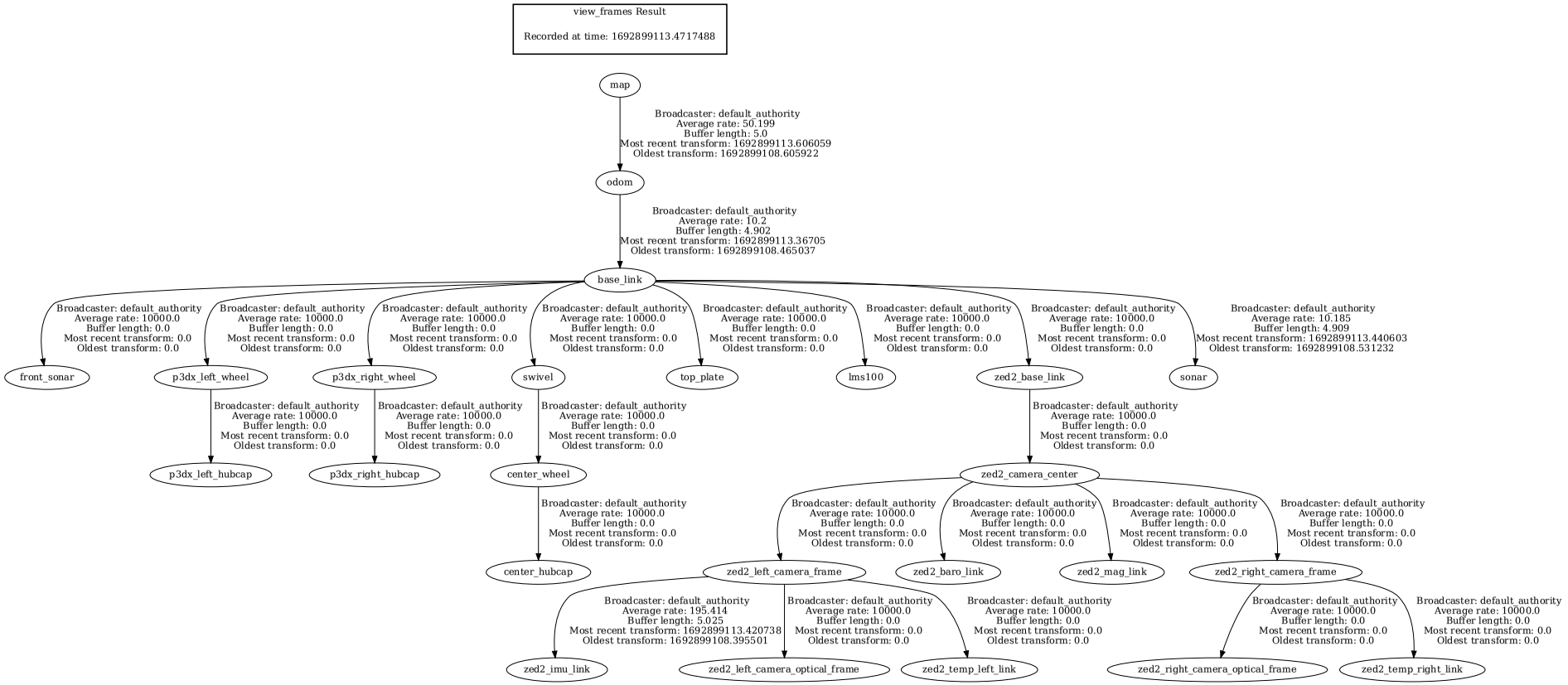

Figure C.5.31: ROS TF tree diagram.

C.6 P3-DX Plataform and Sensors

Figure C.6.32: (a) ZED2 Stereo Camera, (b) Jetson AGX Xavier, (c) RGB-D Orbbec Astra, (d) Hokuyo URG-04LX laser range finder, (e) P3-DX Platform.

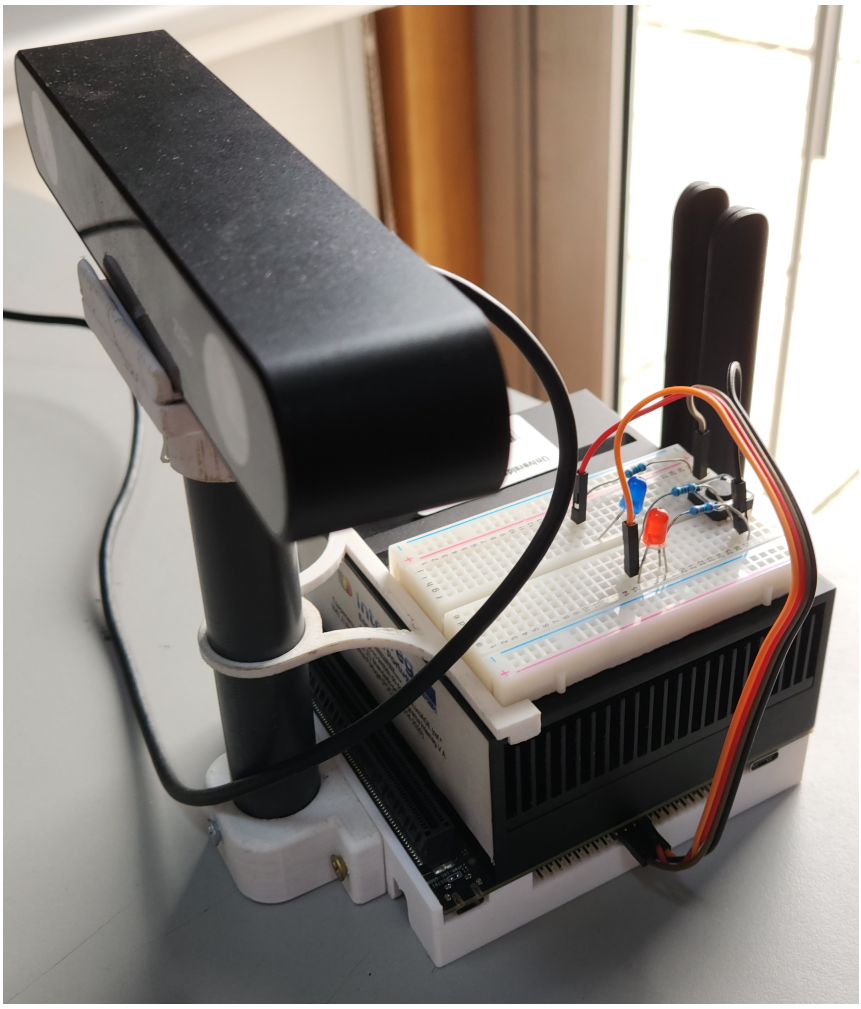

Figure C.6.33: Jetson AGX Xavier with e-top circuit and ZED 2 camera.

Figure C.6.34: ZED 2 stereo camera.

Figure C.6.35: Hokuyo URG-04LX laser range finder.

Figure C.6.36: Intel 8265 M.2 wireless network.

C.7 Box Plot

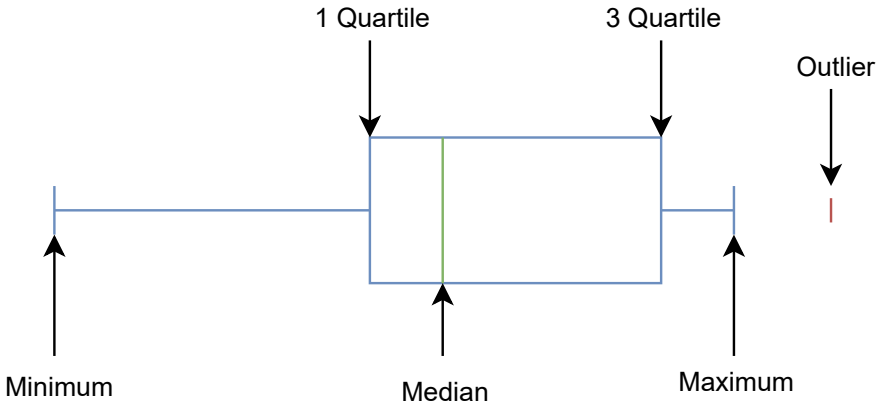

Figure C.7.37: Box plot summary illustration. Box plots are utilized to represent data distribution in a compact format. It presents the minimum and maximum values as well as the 4 quartiles $Q1$, $Q2 \equiv Median$, $Q3$, $Q4 \equiv Maximum$.

Appendix D

System Parameters

This appendix presents a list of the values adopted by the system of all parameters mentioned in this document.

Parameter	Value
Samplig (section 3.1, page 14)	
ZED2 Stereo Camera	
Image Resolution	672×376 pixels
$grab\_frame\_rate$	10 [Hz]
pub_frame_rate	10 [Hz]
Point Cloud Resolution	672×376 points
$point\_cloud\_freq$	10 [Hz]
cam_pose	$\begin{bmatrix}0.10 & 0.0 & 0.84 & 0.0 & 0.0 & 0.0\end{bmatrix}$
Object Detection (section 3.2, page 14)	
N_{yolov5}	80
Geometric Segmentation (section 3.3, page 15)	
RoI	
R_{min}	0.2 [m]
R_{max}	1.7 [m]
Voxelization	
S_{leaf}	0.03 [m]
SOR	
$mean_k$	8 samples
$std\_multiplier$	0.3
Clustering	
$cluster_{min}$	50 points
$cluster_{max}$	10000 points
$cluster_{tol}$	0.065 [m]
Confidence Estimation	
obj_ths	0.2 [m]

Table D.1: System parameters.

Parameter	Value
Binnecd Depth Map (section 3.4, page 23)	
BDM dimensions	24×16 cells
Data Association (section 3.5, page 25)	
Topological Map	
V_d	1 [m]
E_f	0.95
Visibility Histogram	
δ_{base}	2
l_f	5
$\mathbf{w}_{1\times l_f}$	$\begin{bmatrix}0.5 & 1.5 & 6 & 1.5 & 0.5\end{bmatrix}$
l_{min}, l_{max}	-10, +10
Semantic Update	
$p(\hat{\mathbf{x}})$	1
Positive Observations	
o_e	0.2
o_t	0.5
Negative Observations	
o_{valid}	0.5
$f_{ohc}(\mathbf{v})$	$\frac{\sum_{e=1}^{\#\mathbf{v}} 1_{\mathbf{v}_{e>0.7}}}{\sum_{e=1}^{\#\mathbf{v}}(1_{\mathbf{v}_{e>0.7}} + 1_{\mathbf{v}_{e<0.4}})}$
$f_{oc}(tp_1, tp_2, tp_3)$	$\frac{4 \cdot tp_1 + 1 \cdot tp_2 + 1 \cdot tp_3}{6}$

Table D.2: System parameters.

Appendix E

Extra Results

E.1 Object Registration Position Error

Tables E.1.1, E.1.2 and E.1.3 present the *ground truth* centroid of objects in the world and compare them with estimated positions. Figure C.4.24 illustrates the *ground truth*. The mean error of each object of the 3 tables was compiled into the Table 5.2 (page 53).

Reference	Class	Ground Truth	Position	Error
1	Bottle	[-0.01, 0.16, 0.83]	[-1.5,-0.3, 0.7]	[1.49,-0.14, 0.13]
2	Bottle	[2.97, 1.36, 0.86]	-	-
3	TV	[0.33, 1.57, 0.96]	[0.4, 1.6, 0.9]	[0.07,-0.03, 0.06]
4	TV	[0.34, 2.14, 1.08]	[0.4, 2.2, 1.1]	[-0.06,-0.06,-0.02]
5	TV	[0.35, 3.81, 1.15]	[0.5, 3.6, 1.1]	[-0.15, 0.21, 0.05]
6	TV	[0.81, 1.80, 0.99]	[0.7, 1.9, 0.9]	[0.11,-0.10, 0.09]
7	TV	[0.92, 2.20, 0.99]	[0.9, 2.1, 0.9]	[0.02, 0.10, 0.09]
8	TV	[0.83, 3.29, 0.98]	[0.5, 3.6, 1.1]	[0.33,-0.31,-0.12]
9	TV	[3.05, 1.79, 0.98]	[3.0, 1.9, 0.9]	[0.05,-0.11, 0.08]
10	TV	[2.99, 2.35, 0.97]	[2.9, 2.4, 0.9]	[0.09,-0.05, 0.07]
11	Chair	[-0.31, 2.16, 1.19]	[-0.3, 2.1, 0.3]	[-0.01, 0.06, 0.89]
12	Chair	[-0.27, 3.80, 1.19]	[-0.2, 3.7, 0.4]	[-0.07, 0.10, 0.79]
13	Chair	[-3.05, 0.38, 0.45]	[-3.0, 0.5, 0.5]	[-0.05,-0.12,-0.05]
14	Chair	[2.44, 2.13, 1.19]	[2.4, 2.1, 0.5]	[0.04, 0.03, 0.69]
15	Chair	[1.42, 2.08, 1.19]	[1.4, 2.1, 0.5]	[0.02,-0.02, 0.69]

Table E.1.1: Experiment 1.

Reference	Class	Ground Truth	Position	Error
1	Bottle	[-0.01, 0.16, 0.83]	-	-
2	Bottle	[2.97, 1.36, 0.86]	-	-
3	TV	[0.33, 1.57, 0.96]	[0.3, 1.6, 0.9]	[0.03,-0.03, 0.06]
4	TV	[0.34, 2.14, 1.08]	[0.3, 2.2, 1.0]	[0.04,-0.06, 0.08]
5	TV	[0.35, 3.81, 1.15]	-	-
6	TV	[0.81, 1.80, 0.99]	[0.8, 2.0, 0.9]	[0.01,-0.20, 0.09]
7	TV	[0.92, 2.20, 0.99]	[0.8, 2.3, 0.9]	[0.12,-0.10, 0.09]
8	TV	[0.83, 3.29, 0.98]	-	-
9	TV	[3.05, 1.79, 0.98]	[2.8, 2.0, 0.9]	[0.25,-0.21, 0.08]
10	TV	[2.99, 2.35, 0.97]	[2.8, 2.6, 1.0]	[0.19,-0.25,-0.03]
11	Chair	[-0.31, 2.16, 1.19]	[-0.4, 2.0, 0.5]	[0.09, 0.16, 0.69]
12	Chair	[-0.27, 3.80, 1.19]	[-0.4, 3.7, 0.3]	[0.13, 0.10, 0.89]
13	Chair	[-3.05, 0.38, 0.45]	[-2.9, 0.4, 0.4]	[-0.15,-0.02, 0.05]
14	Chair	[2.44, 2.13, 1.19]	[2.3, 2.3, 0.4]	[0.14,-0.17, 0.79]
15	Chair	[1.42, 2.08, 1.19]	[1.4, 2.0, 0.5]	[0.02, 0.08, 0.69]

Table E.1.2: Experiment 2.

Reference	Class	Ground Truth	Position	Error
1	Bottle	[-0.01, 0.16, 0.83]	-	-
2	Bottle	[2.97, 1.36, 0.86]	-	-
3	TV	[0.33, 1.57, 0.96]	[0.5, 1.6, 0.9]	[-0.17,-0.03, 0.06]
4	TV	[0.34, 2.14, 1.08]	[0.5, 2.1, 1.1]	[-0.16, 0.04,-0.02]
5	TV	[0.35, 3.81, 1.15]	-	-
6	TV	[0.81, 1.80, 0.99]	[0.9, 1.9, 1.0]	[-0.09,-0.10,-0.01]
7	TV	[0.92, 2.20, 0.99]	[1.0, 2.1, 0.9]	[-0.08, 0.10, 0.09]
8	TV	[0.83, 3.29, 0.98]	-	-
9	TV	[3.05, 1.79, 0.98]	[3.0, 2.0, 0.9]	[0.05,-0.21, 0.08]
10	TV	[2.99, 2.35, 0.97]	[3.0, 2.3, 0.9]	[-0.01, 0.05, 0.07]
11	Chair	[-0.31, 2.16, 1.19]	[-0.3, 2.0, 0.6]	[-0.01, 0.16, 0.59]
12	Chair	[-0.27, 3.80, 1.19]	[-0.3, 3.6, 0.0]	[0.03, 0.20, 1.19]
13	Chair	[-3.05, 0.38, 0.45]	[-2.9, 0.6, 0.5]	[-0.15,-0.22,-0.05]
14	Chair	[2.44, 2.13, 1.19]	[2.4, 2.1, 0.5]	[0.04, 0.03, 0.69]
15	Chair	[1.42, 2.08, 1.19]	[1.6, 1.9, 0.5]	[-0.18, 0.18, 0.69]

Table E.1.3: Experiment 3.

E.2 Expandable Classification Test

To test the *Expandable Classification* of the system two nodes were created (yolo_v5_odd and yolo_v5_even) based on the yolo_v5 node detector; those nodes are basically the same as the yolo_v5 except for the fact that they only publish detections with even or odd class indexes[1] according to Pseudocode E.5 (see Tables E.2.4, and E.2.5). This method was chosen because each detector tends to apply the same computational load to the system while testing the ability of the system to add a new detector (therefore new classes) in runtime. A simple experiment was conducted with both yolo_v5_even and yolo_v5_odd deployed in independent containers and the execution timings of the system were recorded. Figure E.2.1 shows the results and the timings are basically the same, except for the *Geometric Segmentation* that had a huge performance impact.

[1]YOLOv5 list of classes: https://github.com/ultralytics/yolov5/blob/master/data/coco.yaml

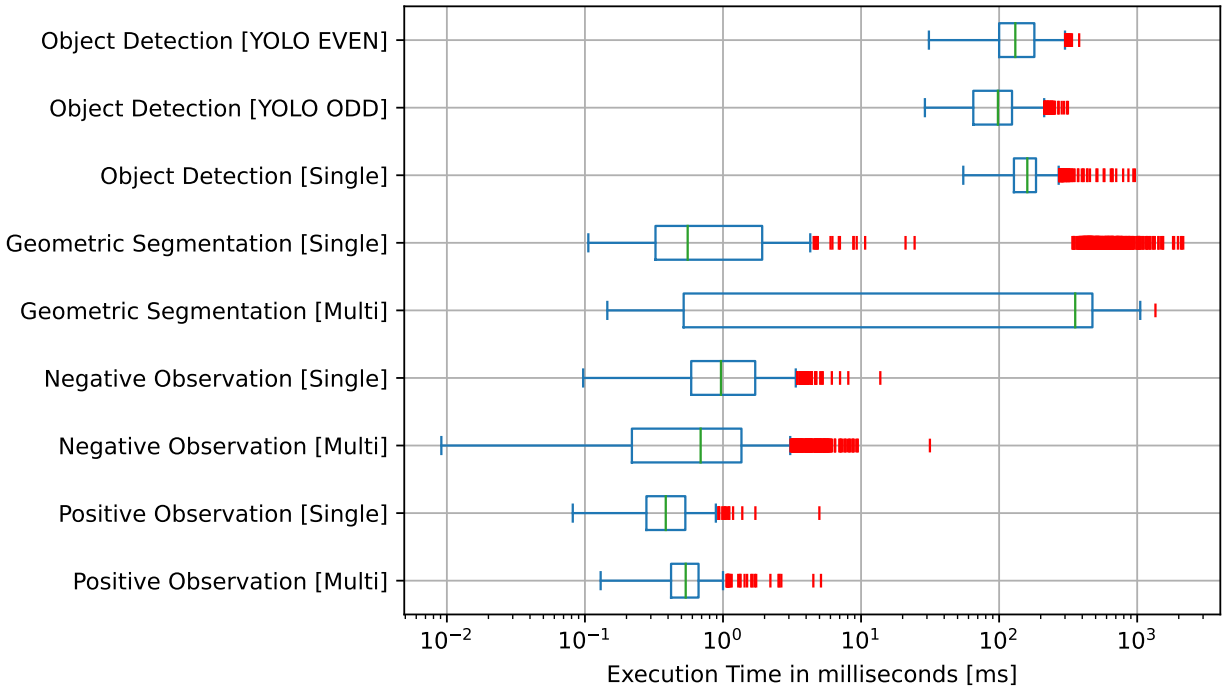

Figure E.2.1: Multiple detectors performance impact. "*Single*" timing when running with one detector, "*Multi*" timing when running with two detectors.

Pseudocode E.5: Even odd model index selection

```
for j in range(8):
    k = 2*j+1; // Indexes to be rejected k*5 -> (k+1)*5-1 [EVEN]
    k = 2*j; // Indexes to be rejected k*5 -> (k+1)*5-1 [ODD]
    if(c >= k*5 and c <= (k+1)*5-1):
        ignore detection;
```

YOLO Even	YOLO Odd
(5) bus	(0) person
(6) train	(1) bicycle
(7) truck	(2) car
(8) boat	(3) motorcycle
(9) traffic light	(4) airplane
(15) cat	(10) fire hydrant
(16) dog	(11) stop sign
(17) horse	(12) parking meter
(18) sheep	(13) bench
(19) cow	(14) bird
(25) umbrella	(20) elephant
(26) handbag	(21) bear
(27) tie	(22) zebra
(28) suitcase	(23) giraffe
(29) frisbee	(24) backpack
(35) baseball glove	(30) skis
(36) skateboard	(31) snowboard
(37) surfboard	(32) sports ball
(38) tennis racket	(33) kite
(39) bottle	(34) baseball bat

Table E.2.4: Even odd classes part 1.

YOLO Even	YOLO Odd
(45) bowl	(40) wine glass
(46) banana	(41) cup
(47) apple	(42) fork
(48) sandwich	(43) knife
(49) orange	(44) spoon
(55) cake	(50) broccoli
(56) chair	(51) carrot
(57) couch	(52) hot dog
(58) potted plant	(53) pizza
(59) bed	(54) donut
(65) remote	(60) dining table
(66) keyboard	(61) toilet
(67) cell phone	(62) tv
(68) microwave	(63) laptop
(69) oven	(64) mouse
(75) vase	(70) toaster
(76) scissors	(71) sink
(77) teddy bear	(72) refrigerator
(78) hair drier	(73) book
(79) toothbrush	(74) clock

Table E.2.5: Even odd classes part 2.

Appendix F

Auxiliary Tools

To help in the development and the portability of the system to future projects related to semantic mapping, a few standard and well-known tools in the Robotics community were used. Those tools were important to drastically reduce the development time in this dissertation. This section will introduce and describe the importance, functionalities, and contribution of each development tool to this thesis.

F.1 ROS

Robot Operating System (ROS) is an open-source middleware released in 2007 by the company *Willow Garage* (latter turned into *Open Robotics*). Despite the name "*Robot Operating System*", actually, it is not an OS and is based primarily on Linux. ROS is a consolidated robotics middleware in research applications because of its ecosystem, integrated with a set of tools essential for development and deployment. In 2015, the second version of the middleware was released, ROS 2. ROS 2 carries all the advantages of its predecessor but with the ambition to become popular in industrial applications, as well as to enable the deployment of fully distributed robotics software systems. It became closer to working as a real-time system, with support for more OS (Ubuntu, Windows 10, MacOS, Debian, etc ...).

ROS is structured to work as a modular integrated system. Each functionality (*e.g. mapping, navigation*) is implemented in a node that is basically a system process. The Inter Process Communication (IPC) used is based on messages and can work in both synchronous and asynchronous modes using "*services*" and "*topics*", respectively. Figure F.1.1 illustrates the flow of messages. "*Services*" establish a synchronous communication between a server

[1]https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html

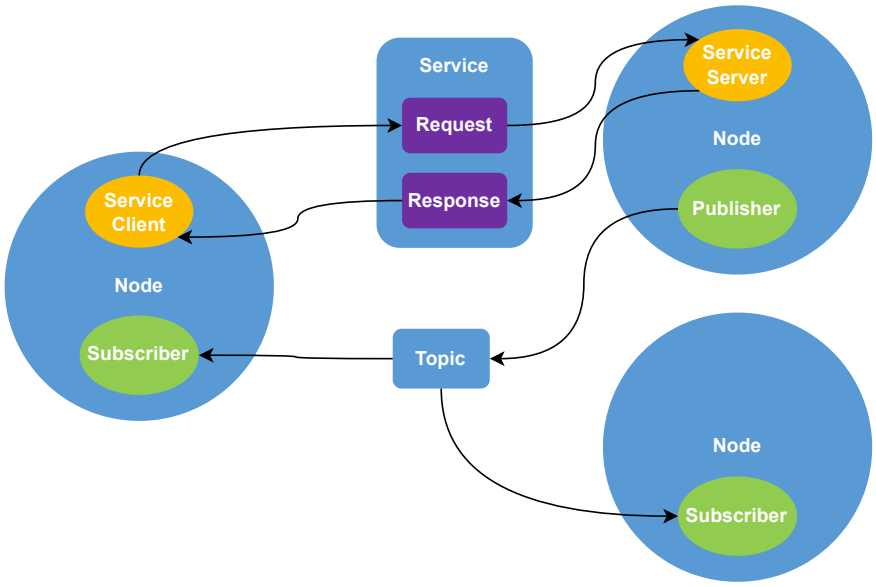

Figure F.1.1: ROS IPC Diagram. Image reproduced from ROS FOXY documentation[1].

node and a client node that is based on request/response pairs. Conversely, "*Topics*" follows the philosophy of "*publisher*" and "*subscriber*": a node can publish a message to a topic and all nodes subscribed to that topic will receive that message asynchronously.

F.2 NVIDIA Jetson

Jetson[2] is a family of System On Module (SOM) systems developed by NVIDIA aimed at low consumption and high computational power embedded systems, including mobile robots. Since it is a SOM, it is a complete computational unit containing Central Processing Unit (CPU), Graphics Processing Unit (GPU), memory, storage, and input/output ports.

Figure F.2.2: Jetson AGX Xavier Development kit. Image reproduced from [3].

Its CPU is developed by NVIDIA and based on an Advanced RISC Machines (ARM)

[2]https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/
[3]https://elinux.org/File:Xavier-module-dev-kit-3qrtr-1945px.png

architecture. The GPU is also developed by NVIDIA, is based on a Volta microarchitecture, and has a total computational power of 22 Tera Operations Per Second (TOPS).

The Jetson uses a custom Linux-based distro called Jetson Linux (popularly known as Linux For Tegra (L4T)) that is combined with the Jetpack Software Development Kit (SDK) [4] to provide hardware acceleration to the system using interfaces like *Tensor RT (TRT)* [5], CUDA Deep Neural Network (cuDNN) [6], and Compute Unified Device Architecture (CUDA) [7].

The Jetson AGX Xavier (Figure F.2.2) was the device chosen to power the robotic platform in this work. This device has been used recently in our lab for *edge computing* research [61, 62, 63]. A concept that consists of moving the heavy processing of modern applications closer to the machinery to take advantage of latency, bandwidth, and security if compared with *cloud computing*. In a mobile robot, this paradigm is key for lower latency perception applications and robot autonomy not relying on the cloud and potentially decreasing power consumption.

F.3 Docker

Docker is a tool to build, share, and run applications easily[8]. It is an "*operating system for containers*", and it works by creating a kernel-level interface between containers and real hardware using virtualization. The docker is available on the principal OS on the market and, because of the virtualization, it transforms applications into multi-platform applications.

With Docker, it is possible to create a container with an OS inside, which can be different from the host system OS, configure it, and install dependencies and applications. The created containers are extremely light because they only store the instructions needed to replicate the OS state (*e.g. Docker Image*), and can be shared and are ready to run in other systems, possibly different OS.

Docker was fundamental in this work to restore previous installation states and to share progress between the simulation environment in a development computer and the Jetson in the robot. In the future, it will be key to easily replicate the work done within this dissertation project.

[4] *Jetpack*: developer.nvidia.com/embedded/jetpack
[5] *TensorRT*: developer.nvidia.com/tensorrt
[6] *cuDNN*: developer.nvidia.com/cudnn
[7] *CUDA*: developer.nvidia.com/cuda-zone
[8] Docker: https://www.docker.com/
[9] https://docs.nvidia.com/ai-enterprise/deployment-guide-bare-metal/0.1.0/docker.html

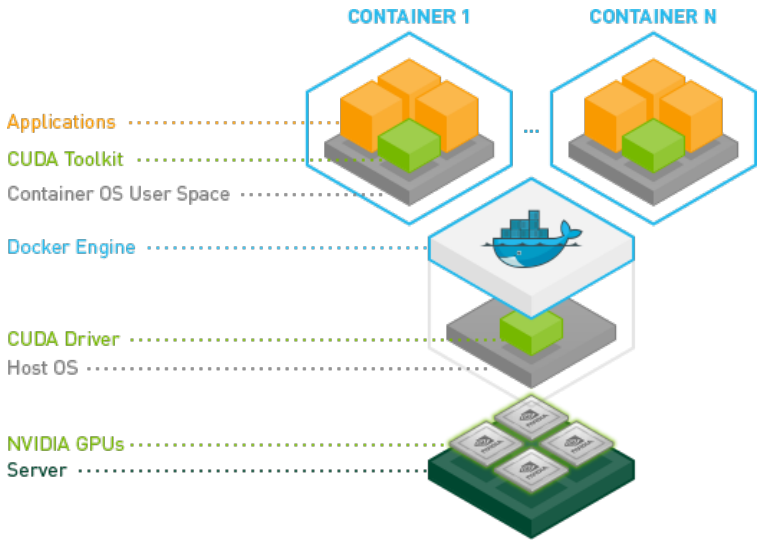

Figure F.3.3: Docker NVIDIA toolkit architecture diagram. Image reproduced from [9].

The use of docker in Jetson systems is almost a prerequisite due to limitations imposed by the ARM architecture, and the custom Linux distro (L4T) utilized in the system. As a consequence, the Jetson family of SOMs has a custom built-in docker engine supported by NVIDIA (see Figure F.3.3), and Jetson-compliant applications are meant to be developed inside containers to ensure minimal platform conflicts as possible.

F.4 Thread Timeout Lifetime Calculation

The *SMAP Geometric Segmentation Node* (section 4.1) generates multiple threads to handle the *Geometric Segmentation* task in parallel; It creates a new thread for each detected object (Pseudocode F.6); Each thread has a limited lifetime based on the number of active threads (th_{active}) defined by equation F.1.

$$timeout = 1000 \cdot e^{-\frac{th_{active}}{\tau}} \ (ms) \tag{F.1}$$

The parameter (τ) was set to a value of 1.738; This value gives a good compromise between rapid value decay and processing time margin. The thread timeout times are shown in figure F.4.4.

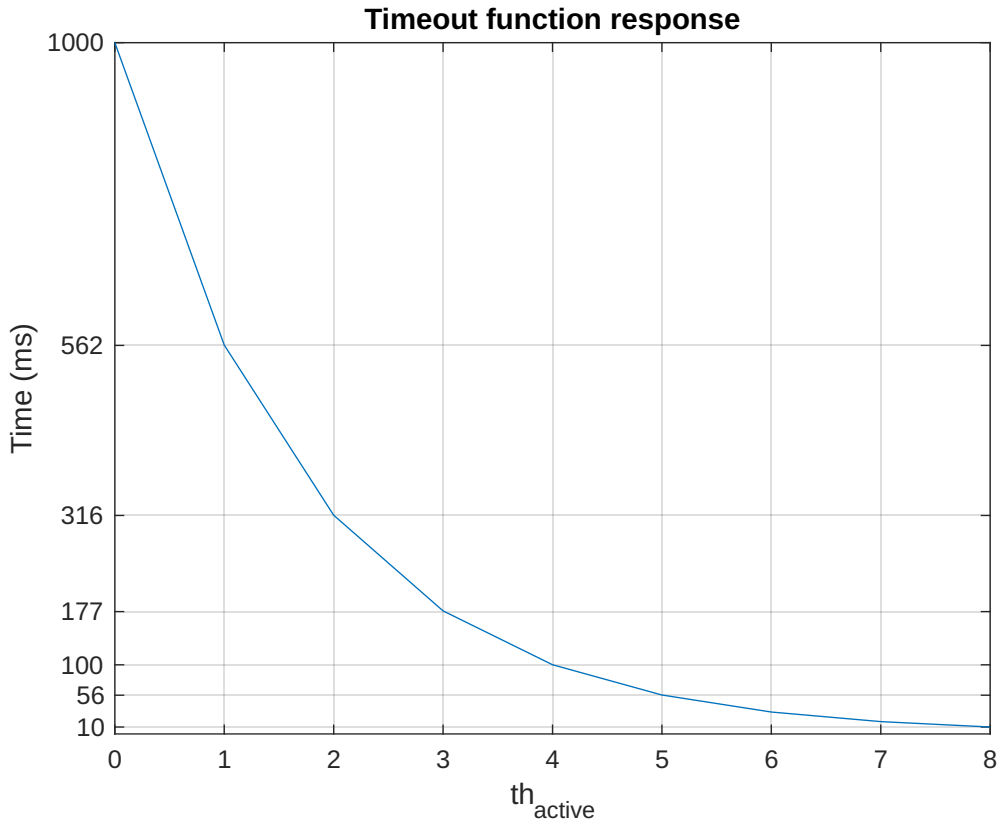

Figure F.4.4: Thread timeout function response

Pseudocode F.6: Object estimator callback

```
predictions_callback(predictions):
    launch depth map binning thread;
    for each object in predictions:
        while th_active >= 8:
            // Ensures that no more than 8 geometric segmentation threads
            // are launched
            wait 1 milliseconds;
        launch geometric segmentation thread;
```

F.5 Parameter Tuning Tool

The *geometric segmentation* pipeline has a lot of steps and parameters that impose compromises between computational cost and performance; to better understand the influence of each step and make fine adjustments to the parameters, a simple Graphical User Interface (GUI) was developed as a tool to help in the parameter fitting of this process. The GUI was developed using *Dear ImGui*[10], a self-contained open source C++ GUI library that is largely used to develop debug tools for real-time 3D applications. This library was chosen

[10] *Dear ImGui*: github.com/ocornut/imgui

because of its simplicity and minimal software integration requirements; the whole interface was implemented inside a single lambda function that is executed only when tunning the parameters. The interface enables the user to change parameter values, enable or disable steps of the pipeline, and plot the execution times of each step in real time; additionally, it can freeze the current input of the pipeline to provide more consistency in the timing comparisons between different step configurations. Figure F.5.5 and video I illustrates the interface.

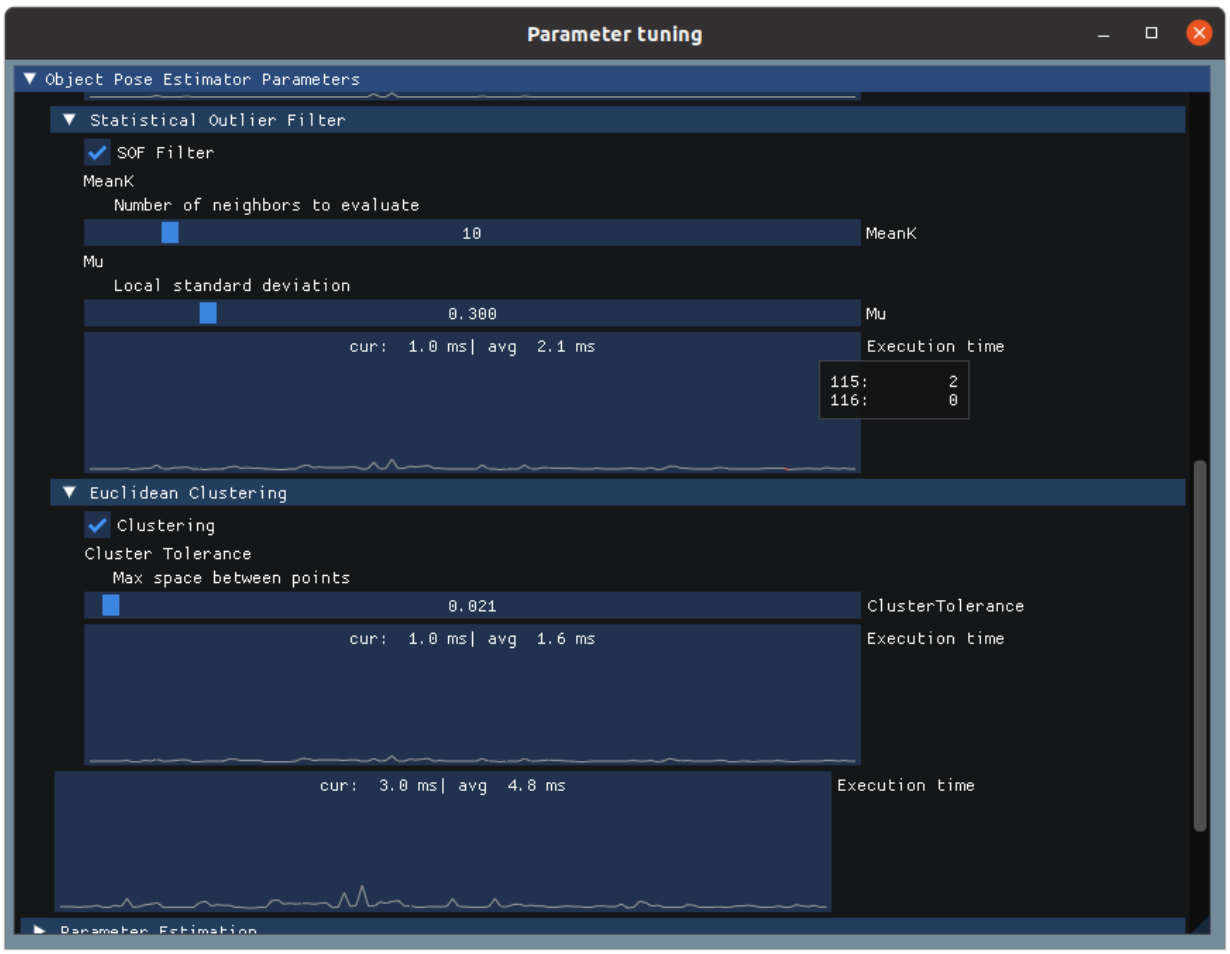

Figure F.5.5: Parameter tuning tool.

F.6 Map Exporter

Map Exporter is a ROS 2 node developed to help visualize the semantic map. The map generated by the Semantic Mapping (SMAP) package is a topological map that is represented in execution time in the *RVIZ2* ROS tool (see Figure F.6.6), the representation of the map after the execution is hardly comprehensible in terms of spatial disposition of the objects (Figure F.6.7). This node aligns and projects the objects into the 2D plane corresponding

to the occupancy grid map of the robot, which has more meaningful spatial characteristics. This is achieved by first removing the obstacles from the occupancy grid (see Figures F.6.8 and F.6.9), and then projecting each object of the topological map in a 2D plane; each projection generates 4 corner points that are iterated using *Bresenham's line algorithm*[11] to create lines between then with varying scales until a filled square-like structure is generated in the new grid map (Fig F.6.10). One new map is generated per object, and the new maps are generated following the standard ROS grid map structure[12].

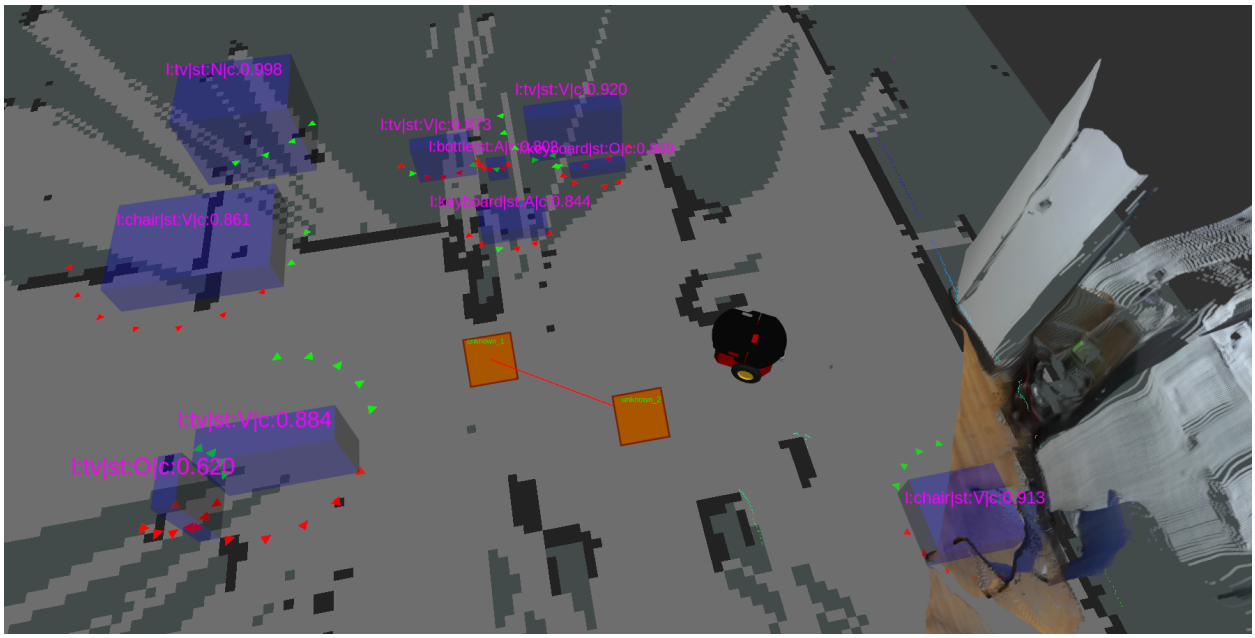

Figure F.6.6: SMAP topological map RVIZ representation. Figure C.4.27 (page 92) illustrates the correspondent location in MRL.

[11]Bresenham's line algorithm: https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm
[12]ROS map_saver: https://github.com/strawlab/navigation/blob/master/map_server/src/map_saver.cpp

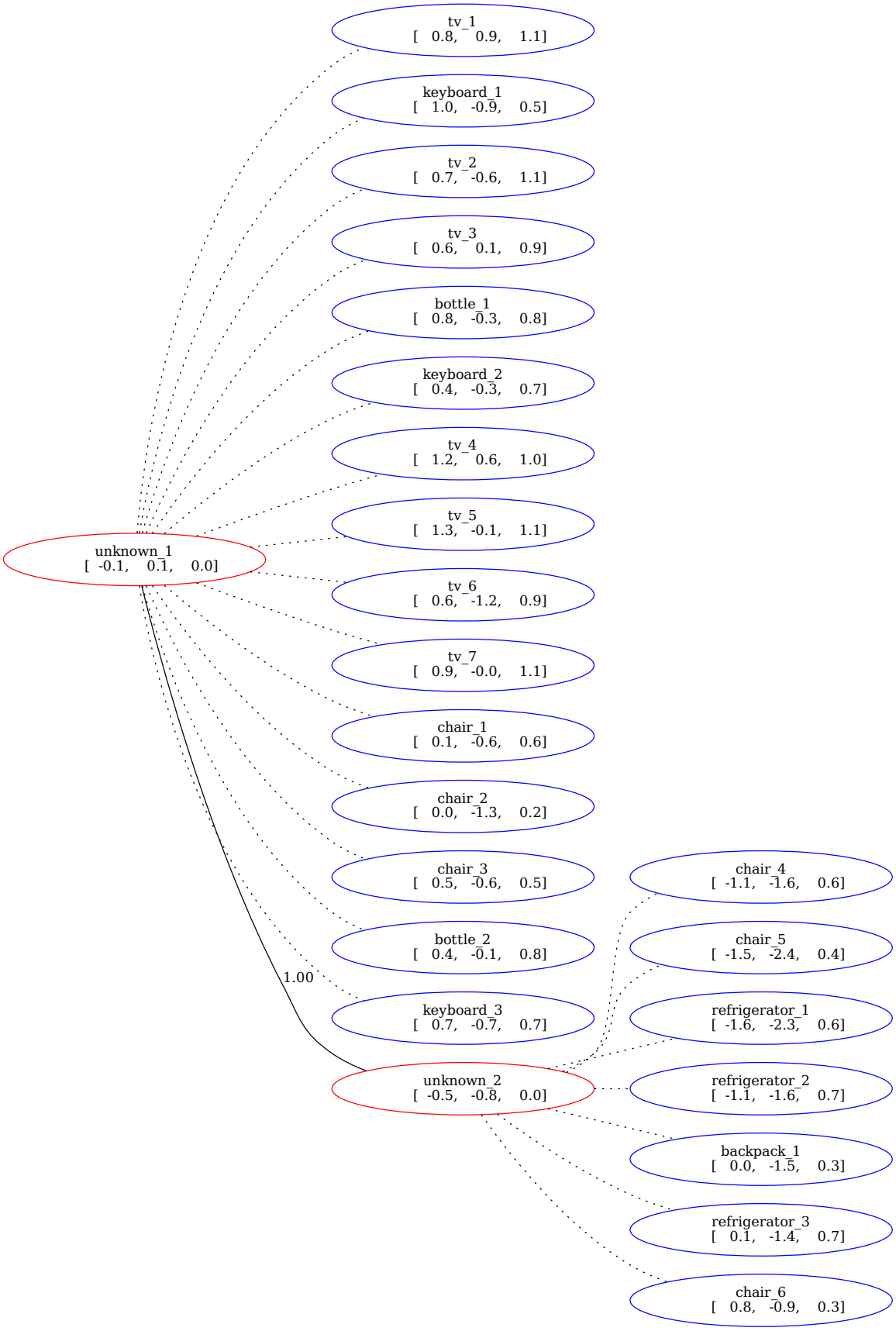

Figure F.6.7: SMAP topological map.

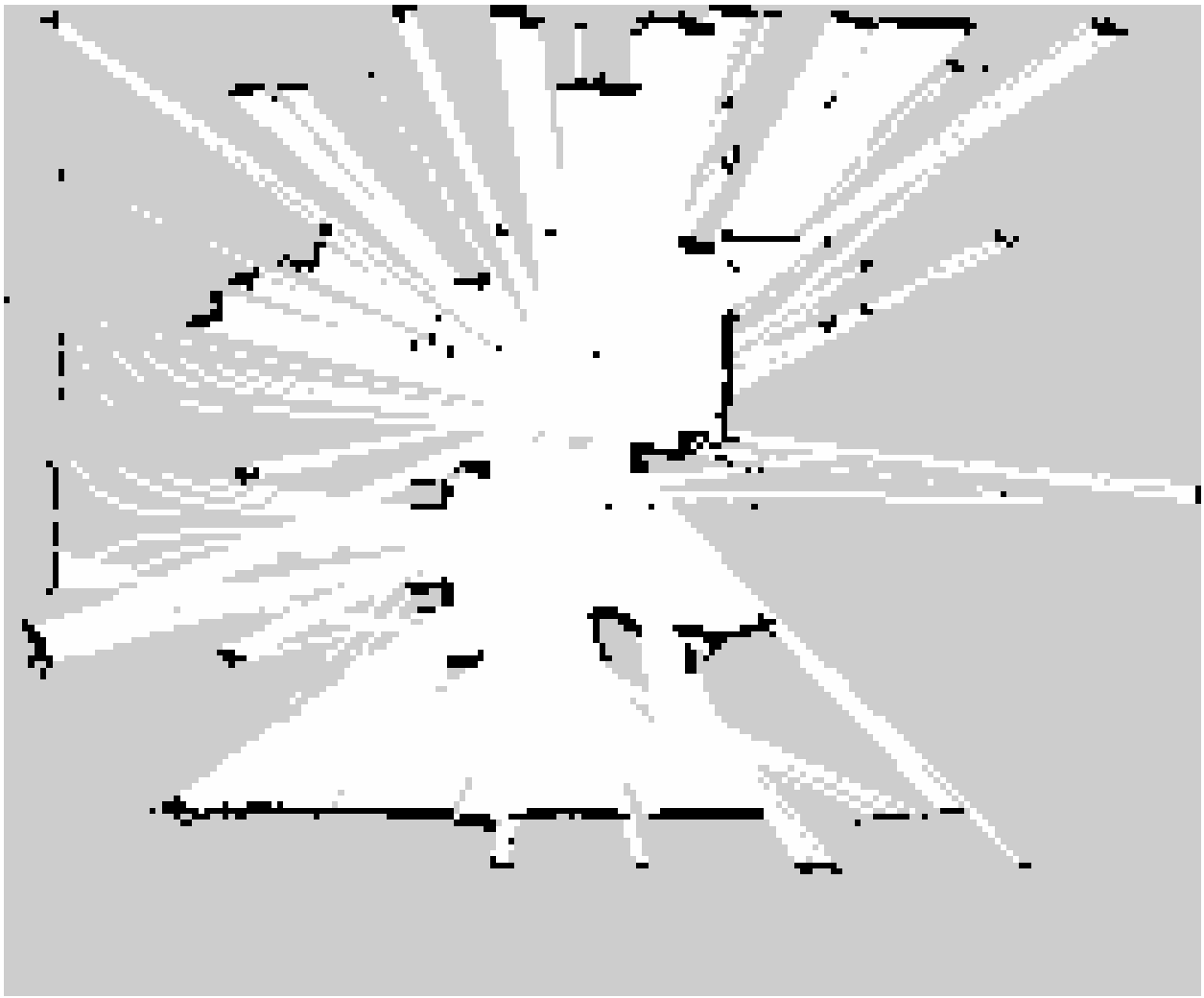

Figure F.6.8: Occupancy grid.

Figure F.6.9: Occupancy grid without obstacles.

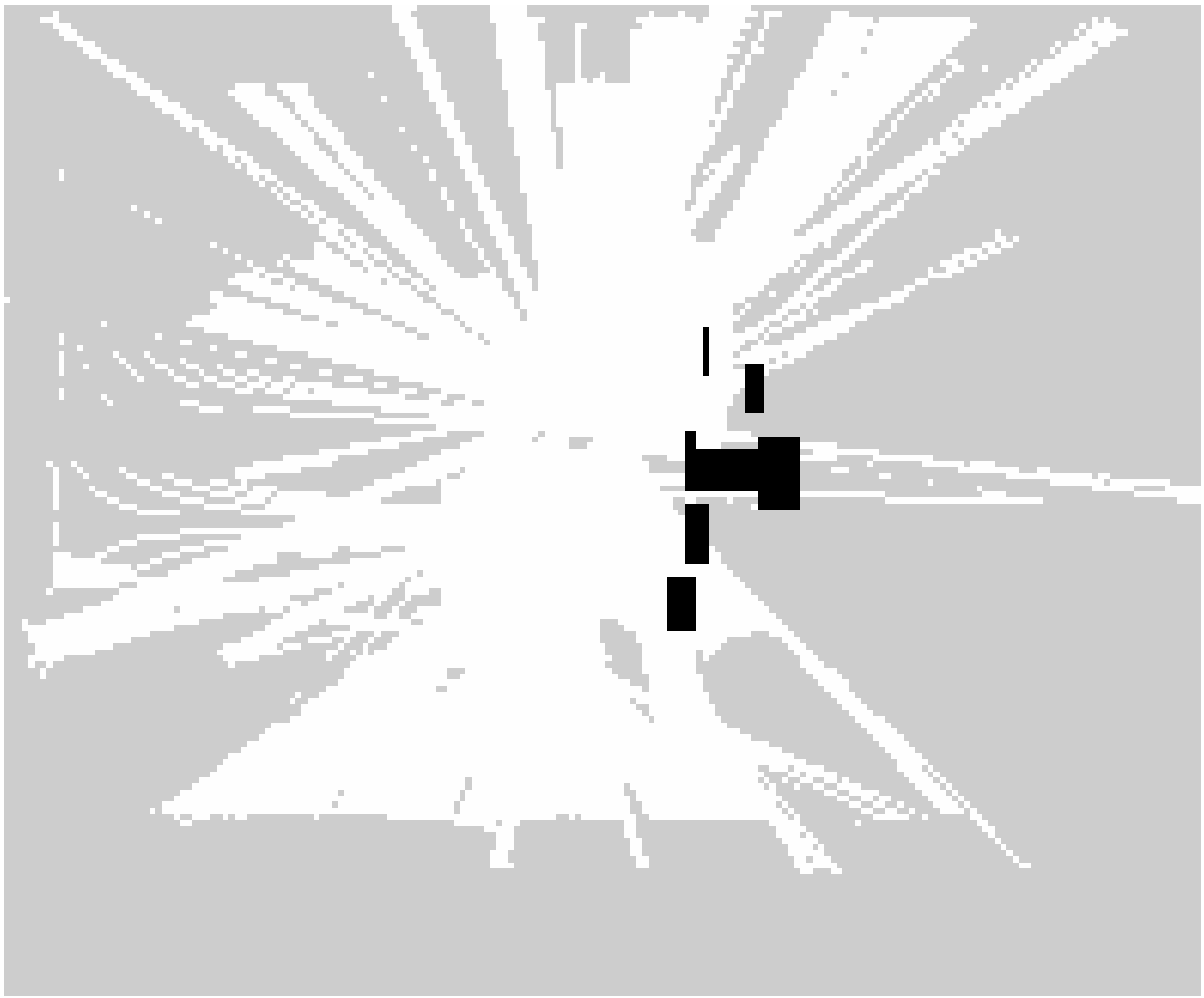

Figure F.6.10: 2D representation of the semantic map for the class TV.

Appendix G

Jetson AGX Xavier ROS 2 Emergency Stop

This is a Python script designed to implement a simple physical kill switch to disable the robot motors as fast as possible in case of an eventual emergency. The script is a ROS Foxy Node that utilizes the Jetson-GPIO library[1] to implement a simple interruption callback detected on Jetson pin 22 (figure G.1)[2]. The callback function make calls to the *ROS Aria* services "*/disable_motors*" and "*/enable_motors*" depending on the time in which the button is held.

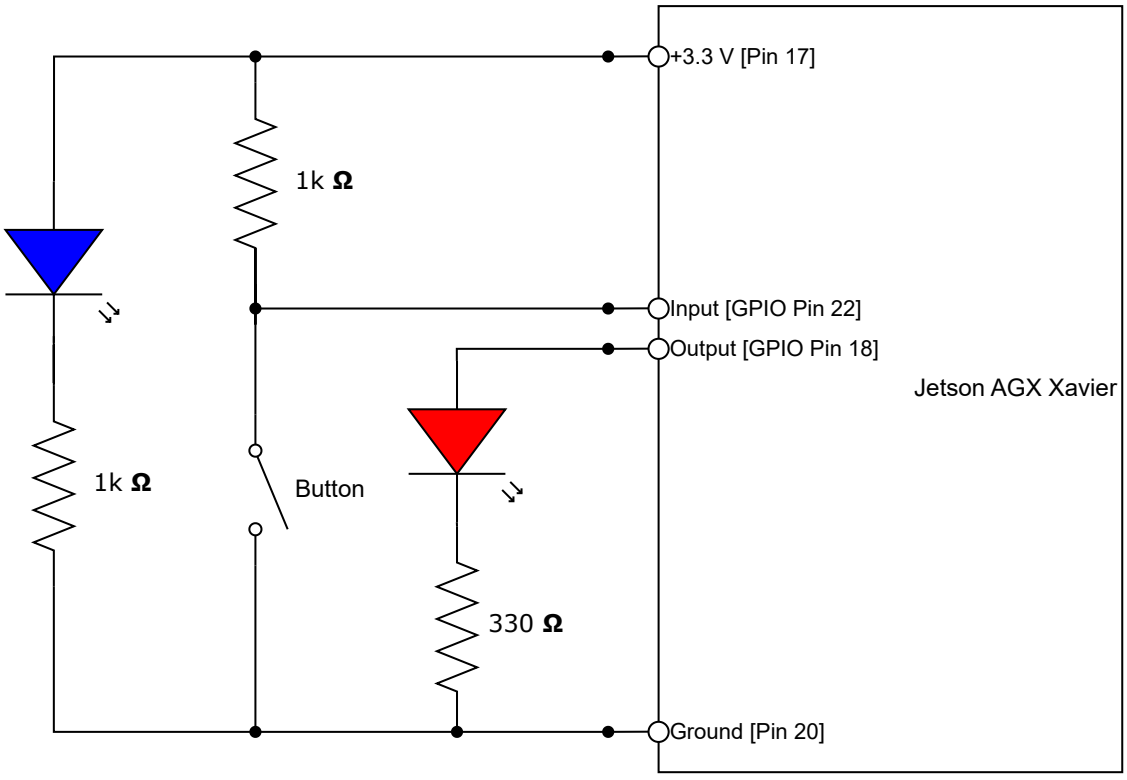

Figure G.1: Emergency stop circuit diagram.

[1]Jetson-GPIO: github.com/NVIDIA/jetson-gpio

[2]Jetson AGX Xavier pinout: jetsonhacks.com/nvidia-jetson-agx-xavier-gpio-header-pinout/

Button behavior:

- Disable motors when started
- Disable motors if the button is held for less than 2 seconds
- Enable motors if the button is held for more than 2 seconds

The circuit diagram shown in Figure G.1 shows two Light Emitting Diode (LED)s and one push button. The purpose of the blue LED is to validate the physical connections of the push button terminals. This is important since the circuit is not soldered. The red LED indicates whether the motors are enabled or disabled: ON means enabled and OFF means disabled.

Appendix H

Project Repositories

H.1 GitHub Repositories

ROS2 P3-DX Package: github.com/lucyannofrota/P3DX

Docker Image Building Scripts: github.com/lucyannofrota/p3dx-docker

SMAP Deploy: github.com/lucyannofrota/smap_deploy.git

H.1.1 SMAP Environments

Desktop Environment: github.com/lucyannofrota/smap-ros2-docker/tree/foxy

Jetson Environment: github.com/lucyannofrota/smap-ros2-docker/tree/jetson

H.1.2 SMAP Packages

SMAP Interfaces: github.com/lucyannofrota/smap_interfaces

SMAP Sampler: github.com/lucyannofrota/smap_sampler

SMAP Perception: github.com/lucyannofrota/smap_perception

SMAP Perception Wrapper: github.com/lucyannofrota/smap_perception_wrapper

SMAP YOLOv5: github.com/lucyannofrota/smap_yolo_v5

SMAP Core: github.com/lucyannofrota/smap_core

H.2 DockerHub Repositories

Jetson Noetic: hub.docker.com/r/lucyannofrota/jetson-noetic

Jetson Noetic + Foxy: hub.docker.com/r/lucyannofrota/jetson-noetic-foxy

Jetson Noetic P3-DX: hub.docker.com/r/lucyannofrota/p3dx-noetic

Jetson Noetic + Foxy P3-DX: hub.docker.com/r/lucyannofrota/p3dx-noetic-foxy
P3-DX Navigation: https://hub.docker.com/r/lucyannofrota/p3dx-navigation
SMAP Images: hub.docker.com/r/lucyannofrota/smap

- **Desktop Environment:** lucyannofrota/smap:env
- **Desktop Visualization (RVIZ2):** lucyannofrota/smap:desktop-deploy
- **Jetson Core Environment:** lucyannofrota/smap:jetson-env-latest
- **Jetson ZED Sampler:** lucyannofrota/smap:jetson-sampler-zed-latest
- **Jetson YOLOv5:** lucyannofrota/smap:jetson-yolov5-latest
- **Jetson Core Deploy:** lucyannofrota/smap:jetson-deploy-latest

Appendix I

Videos

Geometric Segmentation Demo: youtu.be/a5s0wMLJDj4

SMAP Demo: https://youtu.be/_PDy815FyG8

Appendix J

System setup tutorial

This Appendix presents a quick guide on how to deploy and test the SMAP Framework using the GitHub and Dockerhub repositories (Appendix H). The only requirement is to have Docker installed both in the robot and in the remote computer that will be used for visualization.

Docker Installation

The commands to install the Docker engine are available at:

https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository

It is strongly recommended to follow the *post install* tutorial available at:

https://docs.docker.com/engine/install/linux-postinstall/

Jetson Setup

The collection of images from the P3DX platform and SMAP packages listed in Appendix H are grouped into a single *docker compose*[1] file that downloads and launches all the services necessary to properly run the SMAP package in a Jetson host. Run the following commands line by line in a new terminal:

[1]Docker Compose is a script file that describes which Docker services[2]will be executed as well as their privileges and commands. Note: Depending on the Docker engine version the docker compose can be called using "*docker-compose*" or "*docker compose*".

[2]Docker service is the name given to a docker container that is executing some program (providing a service).

```
# Clone smap_deploy repository
git clone --branch jetson https://github.com/lucyannofrota/smap_deploy.git
# Enter the smap_deploy folder
cd smap_deploy
```

Use only one of the commands below to download the images and launch the services:

```
# Launch all services.
docker-compose up -d # Use flag '-d' to detach the containers.

# Launch only the necessary services.
docker-compose up -d p3dx p3dx-navigation smap-sampler smap-yolov5
↪  smap-deploy # Use flag '-d' to detach the containers.

# Launch the services to run even and odd yolo_v5 models.
docker-compose up -d p3dx p3dx-navigation smap-sampler smap-yolov5-even
↪  smap-yolov5-odd smap-deploy # Use flag '-d' to detach the containers.
```

The service "*smap-deploy*" already runs the SMAP node when launched; however, it is not capable of exporting the logs when the container is stopped. In order to have access to the logs, the service "*smap-env*" should be used instead. Since "*smap-env*" service does not launch the SMAP node, it has to be launched and stopped manually. Finishing the node manually using *ctrl+c* possibilities the logs to be exported to the folder './results'. The code below shows how to launch, attach, and execute the services with log access:

```
# Launch only the necessary containers with log access in the folder
↪  ./results.
docker-compose up -d p3dx p3dx-navigation smap-sampler smap-yolov5

# Attach the terminal to a new terminal inside the docker container
# Inside the container terminal launch smap_node
docker exec -it smap-env bash -c ". install/setup.bash && ros2 launch
↪  smap_node smap_launch.py"

# To stop the node and export the logs use "ctrl+c"
```

Jetson Teleop

```
docker exec -it p3dx-navigation bash -c ". install/setup.bash && ros2 run
↪  teleop_twist_keyboard teleop_twist_keyboard"
```

Remote RVIZ 2 Client

Thanks to the DDS protocol adopted in ROS 2, the process of connecting two machines running ROS has never been easier. The only requirement to visualize the SMAP topics in a remote host is that both computers need to be connected to the same network and use the same 'ROS_DOMAIN_ID' variable value. Similar to what was presented in *Jetson Setup* section, the code below describes how to download and launch the RVIZ2 service already configured with the same 'ROS_DOMAIN_ID' as the Jetson Services:

```
# Clone smap_deploy repository
git clone --branch desktop-rviz
↪  https://github.com/lucyannofrota/smap_deploy.git
# Enter the smap_deploy folder
cd smap_deploy

# Launch the RVIZ2 with all the P3DX and SMAP configurations.
docker compose up -d smap-desktop-rviz # Use flag '-d' to detach the
↪  containers.
```

Note: The user has to adjust the *docker-compose.yml* depending on the client host setup. The default is set to a host with NVIDIA drivers; in order to change the graphics driver or run without discrete graphics change or remove the following lines:

```
deploy:
  resources:
```

```
    reservations:
      devices:
        - driver: nvidia
          capabilities: [gpu]
```

Important Note: Because of the Quality of Service (QoS) ROS 2 policies, the nodes running and the order in which they start can affect the way the *tf tree* of the robot is built. Some inconsistencies can be experienced when rebooting the services "*p3dx*" and "*p3dx-navigation*" while the "*smap-desktop-rviz*" service is running. The "*smap-desktop-rviz*" service always needs to be started only after the complete initialization of services "*p3dx*" and "*p3dx-navigation*". The *tf tree* should be similar to Figure C.5.31, and can be checked using the following command:

```
docker exec -it smap-desktop-rviz bash -c ". install/setup.bash &&  cd
↪  /workspace/tf_tree && ros2 run tf2_tools view_frames.py"
```

Appendix K

Docker Image Compile Time

Image	Time
Jetson Noetic	02h58m36s
Jetson Noetic + Foxy	05h43m02s
Jetson Noetic P3-DX	00h06m30s
Jetson Noetic + Foxy P3-DX	01h25m04s
P3-DX Navigation	01h25m04s
ros2-zed	00h33m28s
Jetson ZED Sampler	00h04m17s
Jetson YOLOv5	00h16m55s
Jetson Core Environment	00h00m31s
Jetson Core Deploy	00h00m30s
Total	11h15m41s

Table K.1: Time necessary to compile the docker images developed.