Hypermaps - Beyond occupancy grids

Tobias Zaenker

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.
Espoo 12.8.2019

Supervisor

Prof. Ville Kyrki

Advisor

Dr. Francesco Verdoja

Aalto University
School of Electrical
Engineering

Aalto University
School of Electrical
Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Tobias Zaenker

Title Hypermaps - Beyond occupancy grids

Degree programme Space Science and Technology

Major Space Robotics and Automation **Code of major** ELEC3047

Supervisor Prof. Ville Kyrki

Advisor Dr. Francesco Verdoja

Date 12.8.2019 **Number of pages** 51 **Language** English

Abstract

Intelligent and autonomous robotic applications often require robots to have more information about their environment than provided by traditional occupancy maps. An example are semantic maps, which provide qualitative descriptions of the environment. While research in the area of semantic mapping has been performed, most robotic frameworks still offer only occupancy maps.

In this thesis, a framework is developed to handle multi-layered 2D maps in ROS. The framework offers occupancy and semantic layers, but can be extended with new layer types in the future. Furthermore, an algorithm to automatically generate semantic maps from RGB-D images is presented.

Software tests were performed to check if the framework fulfills all set requirements. It was shown that the requirements are accomplished. Furthermore, the semantic mapping algorithm was evaluated with different configurations in two test environments, a laboratory and a floor. While the object shapes of the generated semantic maps were not always accurate and some false detections occurred, most objects were successfully detected and placed on the semantic map. Possible ways to improve the accuracy of the mapping in the future are discussed.

Keywords hypermaps, semantic maps, object detection, mapping

Contents

Symbols and abbreviations

Abbreviations

CNN	Convolutional Neural Network
IPA	Institut für Produktionstechnik und Automatisierung
PAM	Point Access Method
PCL	Point Cloud Library
ROS	Robot Operating System
SAM	Spatial Access Method
SLAM	Simultaneous Localization and Mapping
SSD	Single Shot Detector
YAML	YAML Ain't Markup Language
YOLO	You Only Look Once
WKT	Well-Known Text

1 Introduction

Navigation is one of the main tasks of mobile robotics. Successful navigation often requires a map of the environment. Traditionally, occupancy grids are used, which indicate the likelihood of areas in the map being occupied or free. However, with the advance of more intelligent and autonomous robots, the information provided by such a map might not be sufficient. For example, a service robot might need to know where the kitchen is or where to find cups.

A solution to provide intelligent systems with more advanced information is provided by semantic mapping. A semantic map contains not only spatial information, but also a qualitative description of the environment, by using natural language to describe it.

But not only semantic information could be useful for a map. Some applications might have the need for other information in a map, such as temperatures or noise levels. A robot with temperature sensitive equipment could for example want to avoid areas with high temperature levels.

The term "hypermap" was used in [1] for maps that incorporate multimedial information, such as sounds, videos and texts, into a map. Hypermaps should provide spatial navigation (access information in a specified area), thematic navigation (look for information by keywords) and temporal navigation (look for information by time frame). However, commonly used robotic frameworks, like the robot operating system ROS, have not implemented these types of maps, and still rely mostly on occupancy grid maps.

The first aim of this thesis is to develop a framework to handle multi-layered 2D maps, called hypermaps, in ROS. The framework should offer layers for occupancy grids and semantic maps, but be easily extendable with new layer types in the future. Furthermore, it should enable spatial as well as thematic queries on all layers.

The second aim is to create a mapping process to automatically generate semantic maps from RGB-D camera images. To achieve this, object detection should be performed using a neural network, and the detected objects should then be transformed to 2D shapes on a semantic map.

In Section 2, the theoretical background on map representations and structures used to perform efficient queries is given. Furthermore, research in the area of semantic mapping and object detection is presented. In Section 3, the developed framework as well as the semantic mapping algorithm is presented. Section 4 describes the software tests performed to evaluate the framework. In Section 5, the simulations and experiments for evaluating the semantic mapping process are presented. In Section 6, the results are summarized and possible further improvements and research are suggested.

2 Background

In this section, the scientific background used in the framework and for the semantic mapping is described. In Section 2.1, different ways of representing maps are explained. In Section 2.2, research in the area of semantic mapping is presented. Section 2.3 explains methods for objects detection, which is used to obtain semantic information. Section 2.4 describes algorithms to perform segmentation on point clouds, which is needed to find out the location of detected objects. In Section 2.5, methods to efficiently perform spatial queries are presented.

2.1 Map representations

In mobile robotics, navigation is one of the main tasks to perform, which means guiding the robot towards a goal [2]. The first step to successful navigation is the localization of the robot in the environment. In addition to sensors, this requires a model of the environment, a map. There are multiple ways to represent maps, which differ in precision and complexity [3]. While the described methods could be applied in 3D, 2D maps are more common, as the complexity of the maps increases drastically in 3D, and a planar map is mostly sufficient for navigation.

2.1.1 Continuous representation

Continuous-valued maps allow for an exact representation of the environment. One way of achieving that is to represent obstacles as polygons. The map complexity depends on the number and shape of obstacles. Often, simplifications are performed to reduce it, e.g. by replacing complex polygons with simpler ones or only displaying features relevant to the sensors and navigation.

Another way of simplifying a continuous map is using a line representation. Instead of polygons, obstacles are constructed from a set of infinite lines. While this does not preserve all information, such a map can still be useful for localization purposes. An example of a map using line representation is found in [4]. Instead of using infinite lines, the map consists of line segments, which are extracted from laser scanner range data.

Polygon based maps can be used for navigational purposes. In [5], a method for navigating a robot in an environment of polygonal obstacles is described. A navigational graph is built from the obstacles, which can then be searched for a path to the destination. The obstacles are obtained through information acquired by a laser scanner.

There is also newer work on navigation with polygonal maps. In [6], a method that constructs only necessary portions of the visibility graph from the obstacles is developed, which leads to an improved path finding performance in dynamic environments. Figure 1 shows an example of a polygonal map, including the generated visibility graph as well as the found path.

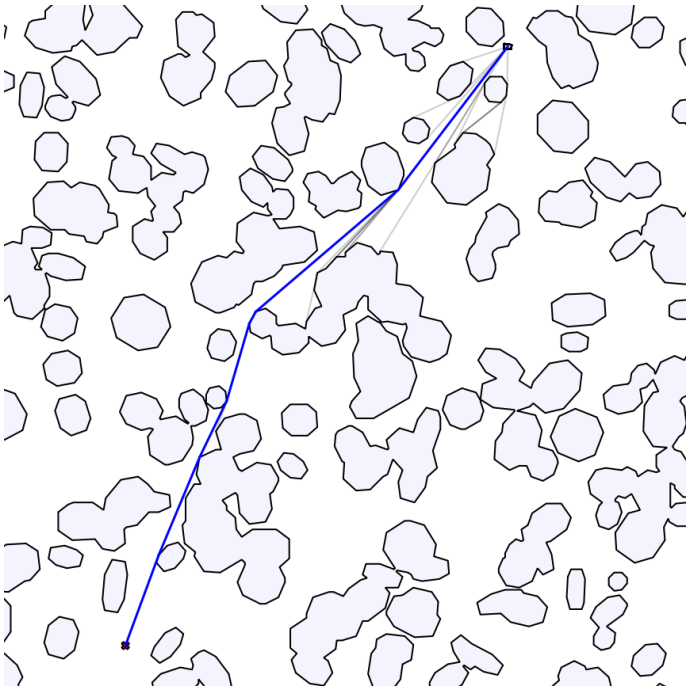


Figure 1: Example of navigation in a polygonal map [6].

2.1.2 Space decomposition

Instead of trying to find a continuous representation of the environment, an often used strategy is to decompose the space into discrete cells. One possible way of doing this without information loss is to use exact cell decomposition, in which cells are chosen in a way to cover all of the space that is not covered by obstacles. While this can work well in simple environments, the complexity of the map increases with the complexity of the obstacles.

Exact cell decomposition maps can also be utilized for navigational purposes. In [7], an algorithm to generate the exact cell decomposition of a map is described. The method creates a connectivity graph connecting neighboring cells, which can be searched for a path between two states. Figure 2 shows an example of a generated exact cell decomposition with this algorithm.

To avoid varying complexity, a fixed decomposition can be performed, in which the map consists of a set of fixed-sized squares. Each cell can store whether it contains an obstacle or is free. This kind of map is called an occupancy grid. Due to the discretization of space, cells could however be only partly covered by obstacles. In order to represent this, the assigned value to a cell usually indicates the probability of an obstacle at that cell.

Occupancy grid maps are the most common types of maps used in robotics [3]. One reason for that is that automatic mapping of occupancy grids from sensor data like laser scanners is well researched. An algorithm to generate occupancy grid maps is presented in [8]. This algorithm provides a solution to the Simultaneous Localization and Mapping (SLAM) problem, as it generates a map while providing an estimate of the location of the robot in the map. It was implemented by the

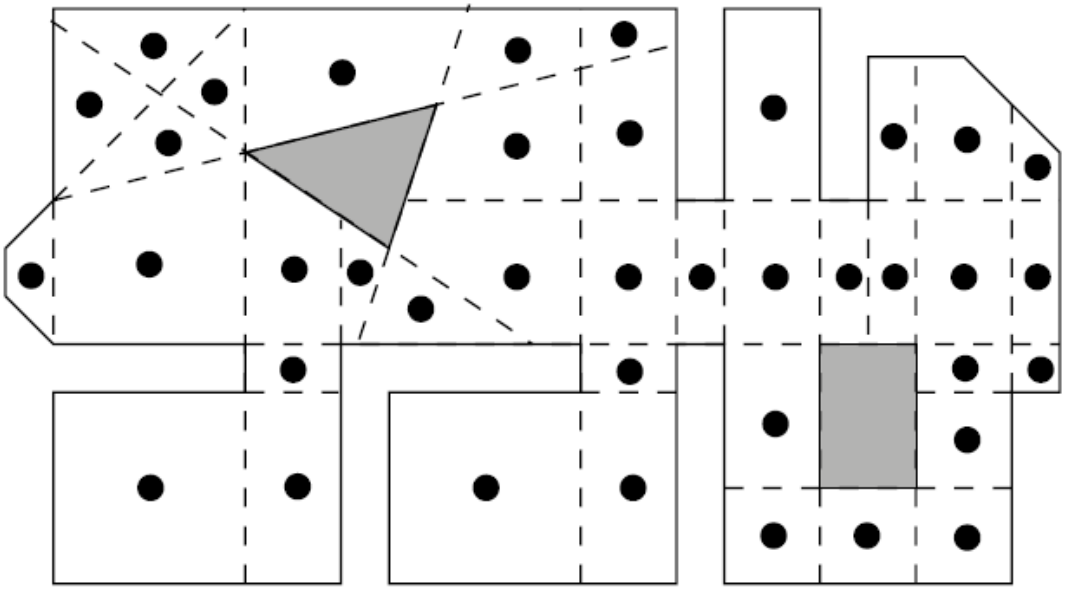


Figure 2: Example of exact cell decomposition [7].

OpenSLAM GMapping library [1] and is commonly used for occupancy grid mapping. Figure 3 shows an example of a map generated with this algorithm, where black pixels indicate occupied cells, white pixels free cells and grey pixels unknown cells.

2.1.3 Topological maps

The map types so far were storing geometrical representations of the environment. Alternatively, a graph consisting of nodes and connection between these nodes can be used as a model of the environment. This type of map is called topological map. For example, each room and corridor of a building can be a node. If two nodes are connected, it must be possible to directly travel between the nodes (e.g. if two rooms are connected by a door).

Topological maps are often combined with metric maps. In [9], a method to generate a topological map on top of a grid-based map is proposed. That way, the geometric accuracy of metric maps is combined with the more efficient planning capabilities of topological maps.

Topological maps can also prove useful for navigation. In [10], images of the environment are stored in nodes in a graph. Features are extracted from the images in order to match similarities. Similar images that are likely to have been taken at close distance are connected in the graph. Localization can be performed by finding the node that has the best similarity to the current observation. When navigating to a goal node, a path in the graph can be found, and the robot can adjust its movement to move from node to node. Figure 4 shows a generated appearance based graph,

[1] https://openslam-org.github.io/gmapping.html

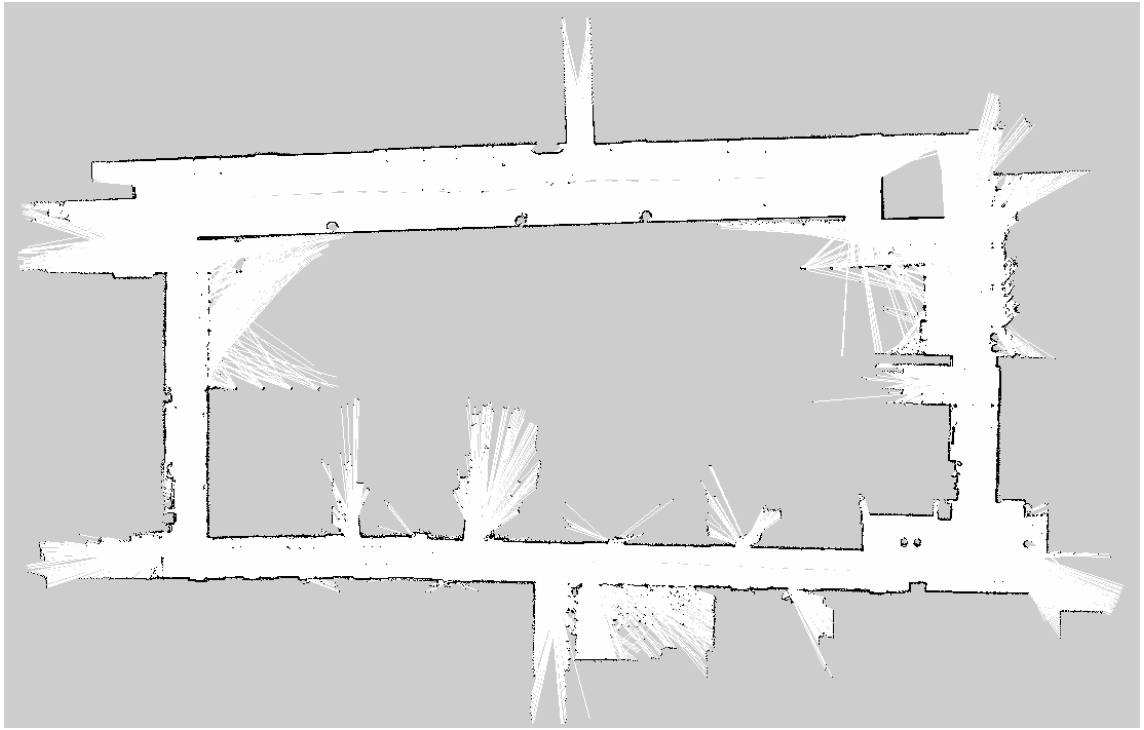


Figure 3: Example of an occupancy grid map.

with the circles representing the images and the lines connecting similar images.

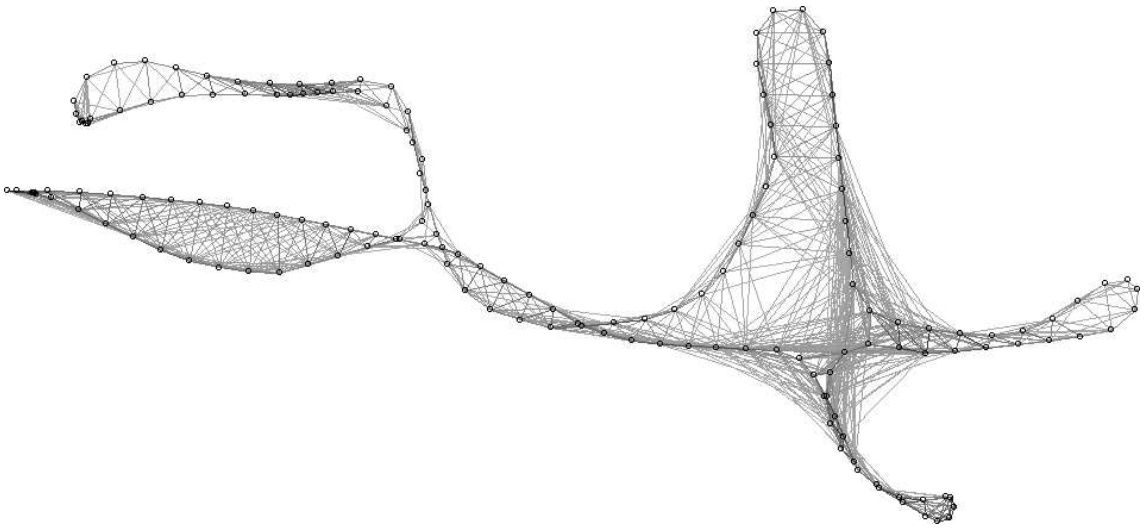


Figure 4: Example of a topological map [10].

2.2 Semantic maps

Traditionally in robotics, maps incorporate only geometrical information about the environment, i.e. the structure and position of obstacles. In semantic maps on the

other hand, labels are assigned to the environment, in order to describe qualitative features of it. The labels could either be direct descriptions for objects found in the environment, e.g. “chair”, “plant” and “desk”, or more abstract descriptions of places, like “office”, “floor” and “kitchen”.

In [11], multiple approaches for semantic maps are compared and classified. Following that work, Section 2.2.1 describes the classification based on perception type and Section 2.2.2 the classification based on map scale. In Section 2.2.3, the use of topological maps for semantic mapping is explained.

2.2.1 Type of perception

One classification category is the type of perception a semantic mapping approach uses, which is either based on single cues or multiple cues. Cues can be for example recognized objects, places or shapes, and can come from a variety of sensors, e.g. cameras or laser scanners. Most research focuses on single cues (e.g. only recognizing objects), but some research has been performed on utilizing multiple cues. For example, Constante et al. [12] propose a method to transfer previously learned place recognition knowledge to new scenarios and, in that, a method to fuse multiple visual cues.

The single cue methods are further divided into scene annotation and pixel-wise point cloud annotation methods. In the former methods, labels are assigned to places or objects and placed on a map. These maps usually extend existing geometric maps. An example for scene annotation is [13]. In this work, a geometric occupancy grid map was built using FastSLAM. For the semantic part of the map, the position and extend of the objects was guessed by placing bounding boxes on top of the geometric map.

Pixel-wise point cloud annotation methods on the other hand assign labels to pixels in a point cloud map. For example, Nüchter et al. [14] perform object detection on a point cloud and assign each point a label, which describes whether it is part of the background (e.g. wall or ceiling) or one of the detected objects.

For the semantic maps in this thesis, scene annotation was used: an existing occupancy grid map was expanded with a layer of labeled shapes.

2.2.2 Scale

Another classification category is the scale of the maps [11]. The methods are first divided into indoor and outdoor interpretations. Most research focuses on indoor scenarios, e.g. offices. However, some research has also been performed on semantic mapping for larger-scale outdoor data. For example, Sengupta et al. [15] use street-level images to assign labels such as vegetation, building or road to an outdoor map.

The indoor interpretations are further divided into single scene and large scale approaches. The former looks only at a single frame (e.g. one camera image or point cloud), and performs classification or object detection on that frame. For example, Mozos et al. [16] classify rooms based on RGB-D images. Trevor et al. [17] on the other hand focus on performing object segmentation on point clouds. Both could

be considered pre-stages of larger scale semantic mapping, which could combine the acquired information to larger maps.

Most research focuses on large scale indoor maps that combine a larger set of collected data to a map. For example, the approaches of [13] and [14] (see Section 2.2.1) are both examples for large scale indoor semantic mapping.

The semantic maps in this thesis are also large scale indoor maps. Multiple camera views are combined to build the maps. That way, a mobile robot with a camera can be utilized to map an environment.

2.2.3 Topological semantic maps

The semantic maps explained so far assigned labels either to places or existing pixels. Another approach is to build a semantic map as a topological map (see Section 2.1.3). For example, in [18] a topological map is built consisting of places which are connected by paths. Each place has properties such as the number and type of objects or the shape and size assigned. These properties are used to draw a conclusion about the room category, e.g. single office or meeting room.

2.3 Object detection

In order to automatically generate a semantic map from a camera image stream, the objects occurring in the image have to be detected and then converted to a representation on the map. The object detection part can be achieved using a neural network. Most object detection approaches [19, 20, 21] aim to find labeled bounding boxes indicating the position of objects in the image. Convolutional Neural Networks (CNNs) are most commonly used for image classification tasks, that is, assigning a label to an image. A possible approach to utilize CNNs for object detection is to divide the image into regions and then apply the CNN. However, if the objects can have different sizes and aspect ratios, the number of possible regions is too large to compute the classification for all of them. In 2014, R-CNNs were proposed as a method to overcome this problem [19]. The idea of R-CNNs is to first generate a limited number of category-independent regions proposals and then perform a categorization on each of the proposals with a CNN.

While the limited number of regions makes R-CNNs a feasible approach for object detection, it is not suitable for real-time detection, as forwarding all regions separately through the neural network is still computationally expensive. Several improvements were proposed e.g. Fast R-CNN [22] and Faster R-CNN [23], which allow for greater speed by forwarding the whole image once through the network to extract features and then reusing these features for the different regions.

All variants of R-CNN are region based methods. An alternative approach is offered by single-shot object detectors like SSD [20] and YOLO [21]. Instead of first proposing regions and then performing classification, bounding boxes and class probabilities are proposed simultaneously in a single step. While this approach slightly decreases the accuracy, the detection speed is increased by an order of magnitude, offering real-time object detection at 30 frames per second.

The previously presented methods aim to determine the rough position of objects in a picture by proposing bounding boxes. These boxes may contain a lot of pixels that do not belong to the object. A more accurate detection is determined by instance segmentation methods. Additionally to proposing a bounding box, these methods also give masks as output that specify which pixels belong to the object. Mask R-CNN [24], an extension to Faster R-CNN, provides an additional mask output with only a small overhead compared to Faster R-CNN.

Extending single-shot object detectors for instance segmentation is more difficult. R-CNNs generate features of the image, which are then used to determine possible bounding boxes. These features can be reused to generate the masks. Since single-shot detectors generate bounding boxes in a single step, additional mask output is more expensive. A proposed solution is YOLACT, which divides the instance segmentation into two steps: The first step is generating a set of prototype masks for the image, and the second step is suggesting a set of linear combinations of these masks, one for each instance [25].

Instance segmentation provides a more accurate representation of an object in an image than object detection. However, it is still limited to 2D image data. Modern robots often work with RGB-D cameras, which would allow for object detection in 3D space. Few research focuses on this kind of object detection. Possible extensions for YOLO are Complex-YOLO [26] and YOLO3D [27]. These detectors work on point cloud data and propose 3D bounding boxes, which is useful if the 3D position of objects is needed.

2.4 Point cloud segmentation

As discussed in Section 2.3, most object detection methods work in image domain and often only determine the rough region of the object. In order to identify which points in 3D space belong to the identified object, segmentation can be performed. Segmentation aims to divide the point cloud into a set of clusters, where each cluster contains related points.

In this thesis, point clouds were handles using the Point Cloud Library (PCL) [2]. PCL offers integrated algorithms for the segmentation of point clouds. In Section 2.4.1, PCL's region growing algorithm is presented.

2.4.1 Region growing

For the region growing algorithm, the normals and curvatures of each point have to be known. As first seed, the point with the smallest curvature is picked as starting point for a region, as a low curvature indicates the point is in a flat area. Then, the nearest neighbours of the seed point are checked. If the angle between the normals of the points is lower than a set threshold (angle threshold), the point is added to the region. If the curvature of the new point is below a set threshold (curvature threshold), the point is also added to the seed set, which means its nearest neighbours are checked in the next loop iteration. The process is repeated until the seed set is

[2] http://pointclouds.org/

empty. Then, the region is added to the segment list. The process is repeated until all points in the point cloud have been added to a region. [28]

2.5 Structures for storing spatial data

In order to perform efficient queries on higher level maps like semantic maps, it can be useful to store information in data structures that allow good performance on such queries. Two general types of queries can be differentiated: Spatial queries, that look for information by a location or area, and thematic or keyword queries, that request location or additional information by keywords, e.g. labels in case of a semantic map. Efficient keyword queries can be achieved by using map data structures, for example hash maps. For efficient spatial queries, various data structures exist. Three possible approaches (grid files, quadtrees and R-trees) are presented in this section.

Two general types of spatial indexes can be differentiated: Point access methods (PAMs) and spatial access methods (SAMs). The former only allows storing and accessing points, while the latter stores objects with extent. [29]

2.5.1 Grid file

The grid file [30] is a PAM. For the grid file, the space is divided into cells by axis aligned hyperplanes (lines in case of 2D maps). Neighboring cells can be grouped into blocks. The grid file is build by starting with a single cell. When adding points, a cell is split whenever it exceeds a set capacity. This method works well if the stored points are static and uniformly distributed. Adding and removing data can lead to cells being split or merged, which is computationally expensive [29].

2.5.2 Quadtrees

The quadtree [31] is also a PAM. It uses a tree structure, where each node can have exactly four children. The root contains the whole space. If a node has children, they divide the space of the parent node into four equally sized quadrants. Whenever a quadrant exceeds the set maximum number of stored points, it is divided into children. Due to the space always being split into four equal quadrants, the tree can become unbalanced if the points are distributed unevenly [29].

2.5.3 R-trees

An R-tree [32] is a data structure that stores multidimensional rectangles. Therefore, unlike gridfiles and quadtrees, it is a SAM. Each entry consists of the object and its enclosing rectangle. Leaf nodes contain a number of these entries, while other nodes store a number of child nodes together with the enclosing rectangles of all the objects in that node. Each node has a minimum and a maximum number of entries. Once the maximum number is exceeded, a split into two child nodes is performed. Different algorithms on which child node to choose to insert a new object and on how to split a node exist. The R*-tree variant proposed in [33] needs some additional effort while constructing the tree, but performs faster on queries.

3 Design and Implementation

In the section, the design and implementation details of the hypermap framework and the semantic mapping node is described. Section 3.1 presents the basic concepts of ROS, which was used for implementing the framework, as well as its visualization program rviz. In Section 3.2, the requirements for the framework are defined. Section 3.3 describes the structure of the framework, including its nodes, classes, the types of supported layers, messages, file structures as well as its rviz plugin. In Section 3.4, the algorithm used to generate semantic maps from RGB-D images is explained.

3.1 ROS

Since the developed framwork uses the robot operating system ROS, it is necessary to understand the basic structure of it. ROS [34] is not a traditional operating system, but a framework that builds a communication layer on top of existing operating systems.

3.1.1 Concepts

ROS introduces four main concepts: Nodes, messages, topics and services [34]. Nodes are processes that perform various tasks. Usually, each node is responsible for a single task, and can communicate to other nodes. For communication, messages are used. Messages are defined data structures that can consist of a number of primitive types as well as other message types and arrays. Messages can be transferred between nodes via topics. Topics have a unique name. Nodes can publish messages to topics and subscribe to topics to receive messages. A node can publish and subscribe to multiple topics and a topic can have multiple publishers and subscribers.

While topics work well as asynchronous communication system for most cases, some applications require synchronous communication. For this, ROS provides services. Services consist of a unique name as well as a pair of messages, a request and a response. If a node offers a service, another node can send a request to this service. It then waits until it receives the response from the service node.

3.1.2 Rviz

Rviz is a 3D visualization program provided by ROS. A variety of basic ROS message types (e.g. occupancy grids, point clouds and laser scans) can be natively displayed. For other messages, plugins can be created to display any type of message.

Internally, rviz uses Qt for its GUI and Ogre for the 3D display. Qt [3] is a cross-platform framework for GUI creation in C++. Ogre [4] is a 3D graphics rendering engine.

For displaying data, rviz provides a basic Display class. Displays for specific message types can derive from this class and specify how to display the data.

[3] https://www.qt.io/
[4] https://www.ogre3d.org/

Additionally, properties can be defined, which can be configured by the user to change the behavior of the display. Properties can contain primitive data types like boolean, integer or floating point values, but also more complex structures, such as vectors, colors or quaternions.

3.2 Framework requirements

The following table lists all requirements for the framework. Only functional requirements were defined. Each requirement received an ID, which are referenced in the framework tests (Section 4).

ID	Requirement
F1	The framework shall be able to combine multiple separately saved maps of different types, if the transformation between the maps is known.
F2	The framework shall be able to initiate a mapping process.
F3	The framework shall provide an occupancy grid layer.
F4	The framework shall provide a semantic map layer.
F5	The framework shall provide a map server node.
F6	The framework shall provide the ability to save the the information of each layer and the transformations between the layers to a single file.
F7	The map server node shall be able to load a map from a file saved by the framework.
F8	The map server node shall publish a visual representation of the map that can be displayed in rviz.
F9	The map server shall publish messages that represent each layer, given by the layers.
F10	The map server shall provide a service to retrieve the string representation of a layer for a desired coordinate.
F11	The map server shall provide a service to retrieve the integer value of a layer for a desired coordinate.
F12	The map server shall provide a service to retrieve the coordinates of a given string representation for a layer.
F13	The map server shall provide a service to retrieve the coordinates of a given integer representation for a layer.
F14	The map server shall provide tf transforms from all layers with respect to the base layer.
F15	Each map layer shall provide a method that returns a string representation for a given coordinate.
F16	Each map layer shall provide a method that returns an integer value for a given coordinate.
F17	Each map layer shall have an rviz display for its visualization.
F18	Each map layer shall provide a method to save its data to one or more files.
F19	Each map layer shall provide a method to load its data from one or more files.

F20	The occupancy grid layer shall provide messages of the types nav_msgs/OccupancyGrid and nav_msgs/MapMetaData.

3.3 Framework design

The Hypermap framework consists of three ROS packages. The package “hypermap” provides classes to manage the multi-layered map as well as the map server node. The package “hypermap_msgs” defines messages and services used by the framework. The package “hypermap_rviz_plugin” implements plugins needed to display map layers in rviz. Both the “hypermap” and the “hypermap_rviz_plugin” package depend on “hypermap_msgs”, but are independent of each other.

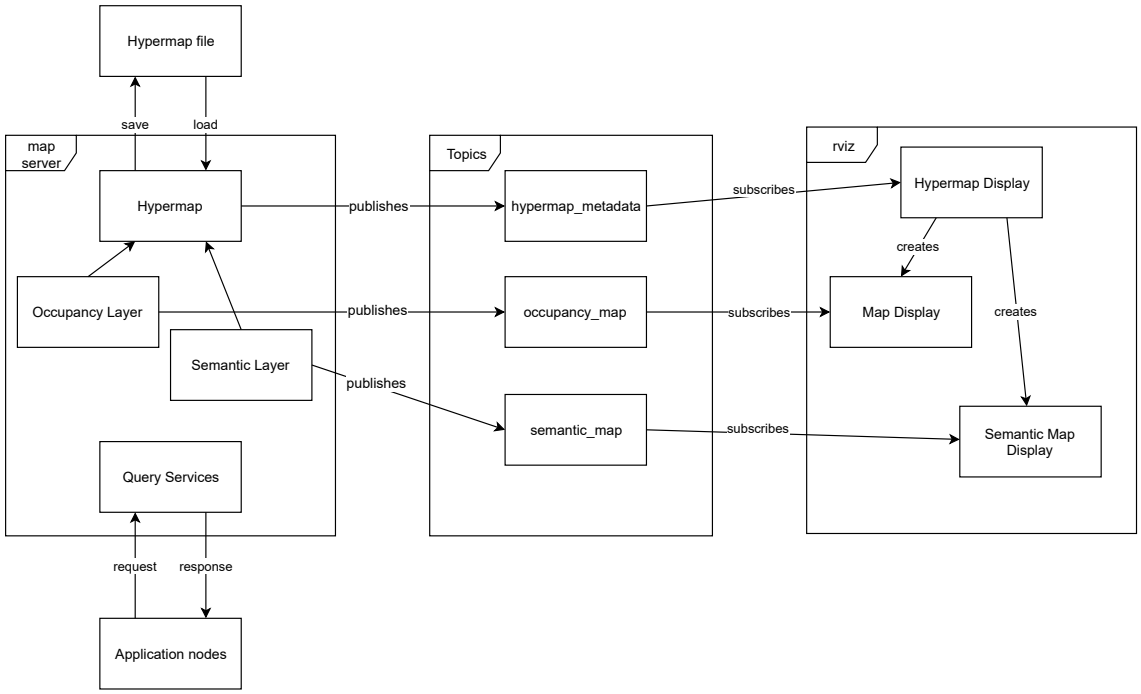


Figure 5: Framework overview.

Figure 5 shows an overview of the structure of the framework. The map server node stores the hypermap with its layers. It can save and load hypermap files and provides services, which allow other nodes to query the map for information. Furthermore, each layer publishes its content to a topic and the metadata of the hypermap is published. The hypermap display provided by the rviz plugin can subscribe to the metadata topic. It automatically creates the necessary displays for the layers, which subscribe to the respective topics.

Section 3.3.1 describes the functionalities of the map server node. In Section 3.3.2, the class structure of the framework is presented. Section 3.3.3 describes the implementation the the different layer types. Section 3.3.4 presents the messages defined by the framework. In Section 3.3.5, the structure of the hypermap files is explained. Section 3.3.6 describes the plugin used to display the maps in rviz.

3.3.1 Nodes

The framework provides a single node type “map_server”. This was specified by requirement F5. The name of the node defaults to “hypermap_server” in order to avoid conflicts with the ROS map_server package for occupancy maps. This map server node is supposed to load a map file from the file system into memory, publish necessary topics and transformations between layers, and provide services to interact with the map. It also provides a command-line interface to interact with the map and allows saving the map to the file system.

Maps can be loaded on the startup of the node by providing a command-line argument specifying a location of a map file (requirement F7). The file has to be a valid hypermap archive; the structure of the file is discussed in Section 3.3.5. Alternatively, a file location can be passed as a ROS parameter called “file”, which has to be specified as private parameter in the namespace of the node. This method allows to load two types of files. In order to load hypermap archives, the private parameter “load_map” has to be set to true. Alternatively, a hypermap configuration file can be loaded if the parameter “load_map_config” is true. If both “load_map” and “load_map_config” are true, an error is thrown.

The node provides services to enable two types of queries on a map. The first type (spatial query) allows requesting the string or integer representation of a layer at a specified point or in a specified area (requirements F10 and F11). The second type (keyword query) gives all locations of a specified string or integer representation on a layer (F12 and F13). It is also possible to limit the latter query on a specified area. Each query request has to contain the name of the layer the request is for, and the point, area (as polygon) or representation (string or integer), depending on the query type. The keyword queries contain an additional polygon type parameter (area). If left empty, the query is performed on the whole map; otherwise, the search is limited to the specified area.

The command-line interface provides functionality to load and save maps (F6 and F7), submit spatial or keyword queries and some additional commands to interact with the node. Table 2 lists all available commands.

3.3.2 Classes

Figure 6 shows an overview of the classes used in the hypermap framework. The main class for the framework is the Hypermap class. A node handle to the current node is stored, which is used for publishing and subscribing to other ROS nodes. Furthermore, an archive file is stored to provide access to hypermap files on the file system. The archive file type is provided by a C++-wrapper for the libzip library, and enables reading and writing from and to zip-compressed archives. All layers of the map are stored in a vector, and a map is used for the translation of the layer name to the index in the vector. The class provides functions to forward the queries described in Section 3.3.1 to the layers. These functions transform the passed points and polygons to the layer frame. Additionally, functions to save and load files and to publish all required data are provided.

Each layer must derive from the class MapLayerBase. This class contains pure

Table 2: Command-line interface commands.

Command	**Function**
help	List all available commands with a short description.
load [file_path]	Load a hypermap archive from the file system.
loadConfig [file_path]	Load a hypermap config file from the file system.
save [file_path]	Save currently loaded map as hypermap archive to the file system.
getString [layer] [location]	Get the string representation at location on layer.
getStrings [layer] [area]	Get a list of string representations in area on layer.
getCoords [layer] [string]	Get a list of coordinates of string on layer.
getCoordsArea [layer] [string] [area]	Get a list of coordinates of string on layer in area.
republish	Force republishing of data of all layers as well as coordinate transforms between layers, if available.
exit	Shut down the map server node.
clear	Clear all loaded map data.
getLayerCount	Returns the number of layer of the currently loaded map.

virtual functions for all query types, which have to be implemented by all non-abstract subclasses. The name and coordinate frame of the layer is stored, as well as the parameters "subscribe_mode" and "enable_update" (see Section 3.3.5). Furthermore, a pointer to the parent hypermap is stored, which is needed to provide access to the node handle as well as requesting and storing layer files.

3.3.3 Layer Types

Two types of layers were implemented: The occupancy grid layer (requirement F3) and the semantic layer (F4). This subsection presents the internal structure of the two layer types.

Occupancy Layer

The occupancy grid layer stores its data internally as OccupancyGrid message, which is a message type provided in the nav_msgs package of ROS. This representation was chosen to provide compatibility of the occupancy layer with nodes that use ROS

Figure 6: Class diagram.

occupancy grids (F20). The message contains a header, a field with metadata and a byte array which represents the occupancy data. The metadata contains the time the map was loaded, the resolution in m/cell, the width and height in cells and the origin of the map as Pose message, which, since the map is 2D, can be expressed as x and y offset combined with a rotation relative to the coordinate frame origin.

In order to respond to spatial queries, coordinates have to be transformed to map indices. To achieve this, the origin position of the map is first subtracted from the requested coordinates. The computed x- and y-difference is then converted to polar coordinates (distance and angle). The map rotation angle is then subtracted from the computed angle. Finally, the coordinates are converted back to Cartesian coordinates, which are the divided by the map resolution. The resulting values, rounded down to the nearest integer, is the two dimensional cell index representing the given coordinates.

The data is stored in a one dimensional array, row by row. The two dimensional index can be converted to an array index my multiplying the y index with the map

width and adding the x index. The occupancy is stored as a signed 8-bit integer. Occupancies should be between 0 and 100 and express the probability of occupation of that cell in percent. A value of -1 indicates an unknown occupancy probability.

Queries for integer representations return the occupancy probability. If a string representation is requested, the query returns "Free" if the occupancy probability is 0, "Occupied" if it is 100, and "Unknown" otherwise.

The occupancy layer publishes two topics: "NAME_map" of the type OccupancyGrid and "NAME_map_metadata" of the type MapMetaData, both defined in the nav_msgs package. It also provides a service "NAME_static_map", which return the map as OccupancyGrid on request. Those are the topics and services provided by the map server of the map_server package, but with the additional NAME prefix, which is replaced by the name of the layer. This avoids name conflicts and allows to have multiple occupancy layers.

Semantic Layer

The semantic layer stores semantic objects in a map. Each object has a unique ID, which is used as key for the map. The main attributes of the semantic objects are their name, which is stored as a string, and their shape, which is a polygon. Additionally, the bounding box and position are stored. The axis aligned bounding box of an object is described by two corner points and is used for spatial queries. The position is calculated as the centroid of their shape and is used as a return value when queried for the location of an object.

The polygonal representation for the semantic layer was chosen as it provides multiple advantages over grid maps: It is possible to differentiate between individual objects, the same space can be occupied by multiple objects, and the size of the map is scale independent.

For geometric representations, the semantic layer utilizes the boost geometry library [5]. The library offers definitions for various geometric shapes, templated to be usable with any point type. For the semantic layer, 2D Cartesian points with double-precision floating point coordinates are used. Boost geometry furthermore offers algorithms to solve geometric problems, e.g. calculating intersections, unions, or whether a point lies inside another geometry.

One of the functionalities used by the semantic layer is the implementation of an R-tree index for spatial queries (see Section 2.5.3). The R-tree uses the bounding boxes of the objects as key, while their ID is stored as value. When queried to find objects within an area, all objects whose bounding boxes intersect with the area are returned. To determine whether the objects intersect, the polygonal shape of each returned object is then checked for intersection. A set of all IDs fulfilling the query is returned.

To enable queries by name (keyword queries), an additional map with names as keys and sets of IDs as values is used. Since names are not unique, each name can be associated with multiple IDs.

[5] http://boost.org/libs/geometry

For keyword queries restricted to an area, spatial and keyword queries are performed separately. The set intersection of the two generated ID sets is then created to determine the objects that fulfill both conditions.

The semantic layer publishes a topic "NAME_semmap" of the type SemanticMap defined in the hypermap_msgs package. The message structure is described in Section 3.3.4.

3.3.4 Messages

Message and service types are defined in the hypermap_msgs package. Five types of messages are defined, of which two are used by the hypermap directly, and three are used by the semantic layer.

LayerMetaData

This message describes the metadata of a layer. It is used as part of the hypermap metadata.

- **class_name:** Layer class. Should be either OccupancyGridLayer or SemanticLayer.
- **name:** Unique name of the layer.
- **frame_id:** Coordinate frame of the layer.
- **subscribe_mode:** True if layer is in subscribe mode.
- **enable_update:** True if updating the layer is enabled.

HypermapMetaData

The hypermap metadata describes the structure of the hypermap. It is published by the map server.

- **layer_cnt:** Number of layers.
- **node_name:** Name of the node; this is the namespace for published topics and services.
- **layers:** List with layer metadata.

SemanticObject

This message describes a single semantic object. It is used as part of a semantic map or a semantic map update.

- **id:** Integer ID of the object. In a semantic map, each object should have a unique ID.

- **shape:** Polygon specifying the shape of the object.
- **point:** Point specifying the position of the object. Should be the centroid of the object.
- **name:** Class of the semantic object as string.

SemanticMap

The semantic map message is published by the semantic layer. It describes a map containing of multiple semantic objects.

- **header:** Header containing frame and time stamp of the map.
- **objects:** List of semantic objects contained in the map.

SemanticMapUpdate

This message describes changes to a semantic map. It can be used to modify the semantic layer of a hypermap.

- **to_add:** List of semantic objects to be added to the map.
- **to_modify:** List of IDs of objects that should be updated.
- **updates:** List of semantic objects that contain the updates.
- **to_delete:** List of IDs of objects to be deleted from the map.

3.3.5 Map file structure

Hypermaps are saved as compressed zip archives. The archive must contain a configuration file called "hmap_config.yaml". This file consists of a YAML [6] node "layers", which is a sequence of mappings. For each layer, the following attributes have to be specified:

- **class:** Specifies the layer type. Currently available classes are OccupancyGridLayer and SemanticLayer.
- **frame_id:** The coordinate frame in which the layer is specified.
- **name:** The name of the layer. Each layer has to have a unique name.
- **load_file:** Boolean attribute that specifies whether the layer should be loaded from a file, or an empty layer should be created.

Additionally, the following optional attributes can be provided:

[6] https://yaml.org/

- **file:** The name of the file to load the layer from. Has to be provided only if load_file is true. Note that this name is passed to the load function of the layer; this function may load multiple different files depending on the type.
- **subscribe_mode:** Boolean attribute. If set to true, it indicates that the layer should be built by subscribing to topics published by an external node, for example a mapping node. The default value is false.
- **enable_update:** Boolean attribute. Specifies whether the layer should accept update messages from external nodes. The default value is true.

If the layers are specified in different coordinate frames, the file should additionally contain a node 'transforms', which specifies the transformations between the layer frames.

Occupancy files

Occupancy grid layers are saved the same way as occupancy maps in ROS, that is, a YAML file containing the metadata of the layer, and an image file containing the occupancy values.

In order to maintain compatibility with ROS occupancy maps, the metadata is structured the same way as for maps in the map_server package. A description of the parameters can also be found on the ROS wiki page for the map server [7].

- **image:** The file name of the map image file consisting of the occupancy values.
- **resolution:** Resolution of the map images in meters per pixel.
- **origin:** Array of three values: x and y offset as well as yaw rotation of the map.
- **occupied_thresh:** Value between 0 and 1. Pixels with higher value than this threshold have an occupancy of 100.
- **free_thresh:** Value between 0 and 1. Pixels with a lower value than this threshold have an occupancy of 0.
- **negate:** If true, the interpretation of colors is reversed (black pixels are free and white pixels occupied).
- **mode:** Must be trinary, scale or raw. If not specified, defaults to trinary. Affects how pixels are interpreted.

The map image files are read with the stb image library [8]. The library supports various formats (JPEG, PNG, TGA, BMP, PSD, GIF, HDR, PIC and PNM). Images are converted to 2-channel greyscale-alpha images. Any degree of transparency is

[7] http://wiki.ros.org/map_server
[8] https://github.com/nothings/stb

interpreted as unknown cell. Otherwise, the value of the greyscale channel (an integer between 0 and 255) is used for computing the occupancy of a cell.

If the negate setting is activated the cell value c is inverted ($c = 255 - c$). In raw mode, this c value is directly used as occupancy. For other modes, the occupancy value p is computed as $p = (255 - c)/255$.

In trinary mode, only three occupancy values are used: -1, 0 and 100. Any value p greater than the occupied threshold p_{occ} has an occupancy 100. For p smaller than the free threshold p_{free}, the occupancy is 0. All other values get the occupancy -1.

In scaled mode, if p is between the free and the occupied threshold, the ratio r is calculated as $r = (p - p_{free})/(p_{occ} - p_{free})$. The occupancy is then computed as $100 \cdot r$, rounded to the nearest integer. Note that this is different from the formula used by the map server package, which is $99 \cdot r$. This was changed because multiplying by 100 provides a better linear scaling of occupancy values between the two thresholds.

For writing images, png++ is utilized, a C++-wrapper for libpng. Therefore, maps are always saved as PNG. Maps are saved as greyscale-alpha images in scaled mode, with an occupancy threshold of 1 and a free threshold of 0.

Semantic files

Semantic layers are saved as single YAML files. These files consist of a list of semantic objects. Each object has the following attributes:

- **name:** The semantic label assigned to the object.
- **shape:** The spatial structure of the object, specified in Well-Known Text (WKT) [9] representation. Only polygon shapes are supported.

3.3.6 Rviz plugin

The package hypermap_rviz_plugin provides a class HypermapDisplay, which derives from the DisplayGroup class. The DisplayGroup class is provided by rviz and allows storing and managing child displays of any type.

The display subscribes to a topic of the type HypermapMetaData. On receiving metadata, it creates a display for each layer specified in the data. The display type is determined by the layer type. For occupancy layers, the MapDisplay provided by rviz is used, as it can display the published OccupancyGrid messages. For semantic layers, a display is provided by the plugin. In case more layer types are added at a later point, corresponding displays should be added to this package, if a suitable display does not already exist.

Each layer display receives the name specified in the layer metadata and can be enabled and disabled separately. Also, the settings specified by the layer display type can be accessed. If new metadata is received (e.g. layers are added or removed), the displays are adjusted automatically.

[9] https://www.opengeospatial.org/standards/sfa

Semantic map display

The semantic map display class is derived from the basic rviz Display class. It subscribes to SemanticMap messages, which contains a list of objects, consisting of a polygonal shape, a position (which should be the centroid of the shape) and a label. The plugin has two boolean properties that allow to enable and disable the display of the shapes and the labels separately. The shapes are displayed by using the ManualObject class of Ogre, which allows the creation of objects with custom geometry. Since only collections of simple geometries like points, lines and triangle can be natively displayed, the polygons are converted to a list of triangles using the earcut triangulation algorithm. Each class receives a unique color taken from the Glasbey lookup table, which is a table aiming to contain a distinct set of colors for categorical images [35]. The labels are displayed at the centroid of the shapes, and are always facing towards the camera view.

Additionally, the float property "char height" can be adjusted to change the size of the labels on the map. Furthermore, each detected class can be enabled or disabled to be displayed separately.

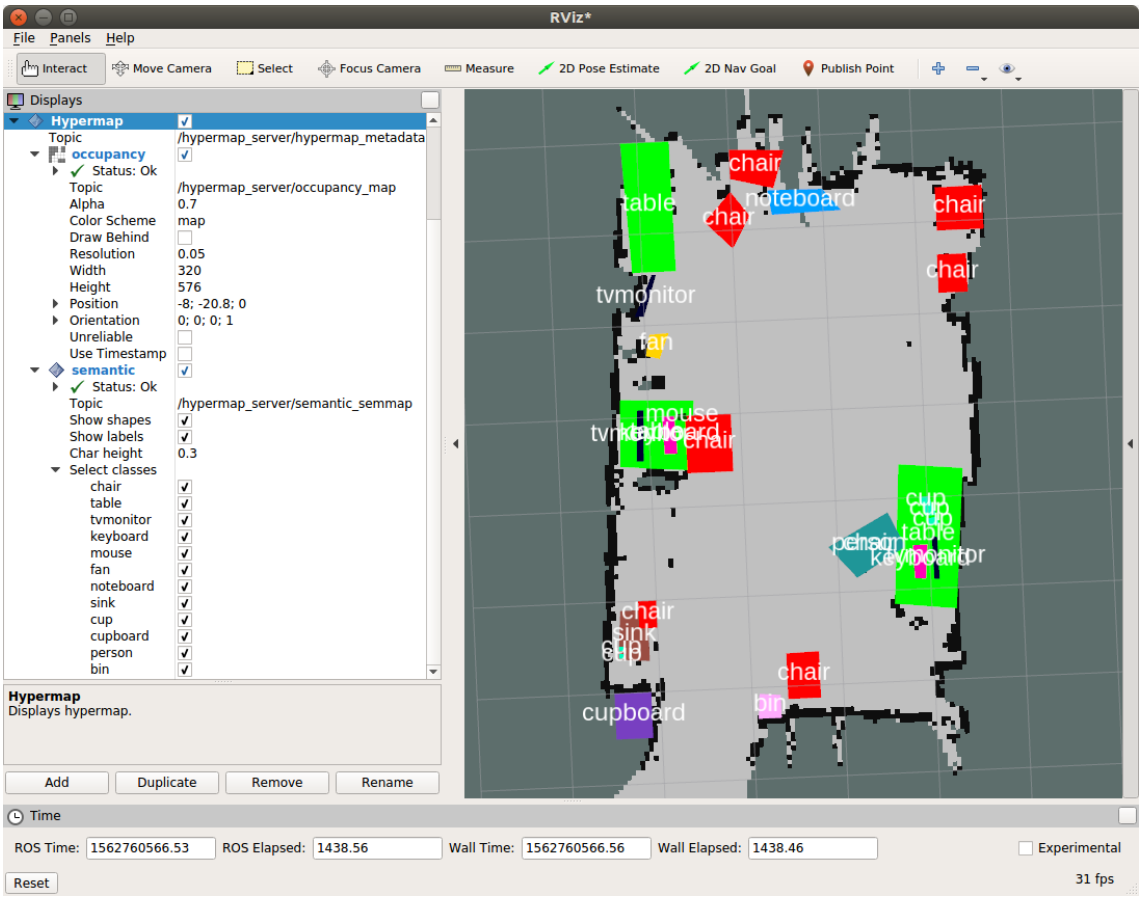


Figure 7: Rviz plugin.

Figure 7 shows the plugin on an example hypermap with an occupancy and a semantic layer. On the left side, the settings for the hypermap display can be

accessed, including all settings for the layer displays. On the right side, the complete map is displayed.

3.4 Semantic mapping

3.4.1 Methodology

Since semantic maps were defined as labeled polygonal maps, the mapping process is challenging. Most common mapping algorithms work with occupancy grids. Each cell stores an occupancy probability. With new sensor information, the probability of the cells can be updated independently. However, updating the belief of polygonal shapes with noisy and possibly contradicting sensor evidence is more difficult. A common approach to obtain polygonal obstacle maps is to first perform mapping on occupancy grids and then convert occupied cells to polygons that approximately cover the occupied area.

For labeled polygonal maps, this approach has several disadvantages. First, different object classes have to be tracked. A solution for achieving that with grid maps would be to maintain a separate grid map for each class. However, this still leaves the disadvantage that information about separate objects of the same class is lost. For example, it would be difficult to differentiate between three chairs standing directly next to each other when extracting polygons from the grid maps.

Therefore, it was decided to perform mapping in the polygonal map environment. The polygonal map consists of uncertain semantic objects. While uncertainty could occur in the shape as well as in the class, it was decided to track each class separately, thereby removing the necessity to keep track of different class probabilities within each object.

For this method, it is assumed that a sensor measurement consists of a list of object hypotheses, where each hypothesis consists of a class and a polygonal shape, as well as the complete observed area. The latter is necessary to be able to remove evidence of falsely detected objects. The process of generating these sensor measurements from RBG-D camera images is described in Sections 3.4.2 to 3.4.4.

For the purpose of the semantic mapping, semantic objects were defined to have three main properties: a class, a list of collected shape evidence and an existence probability. From the list of shapes, the most likely shape of the object has to be derived. The best way of achieving that depends on the quality of the evidence: If the evidence is likely to contain the complete object, it can be sufficient to pick one of the shapes that is supported the most by all shapes. If the evidence often covers only part of an object, combining multiple shapes (e.g. by computing the convex hull) might be the better approach.

For incoming new evidence, the objects that this evidence could belong to have to be identified. A useful metric to select whether a shape correlates to an existing shape can be the Jaccard index, which is defined as the area of the intersection divided by the area of the union of two shapes [36]. Therefore, the Jaccard index is a value between 0 and 1, and a higher Jaccard index indicates a better similarity of the shapes.

If an object is identified as possible fit to the new shape, the shape can be added to the shape list of the object and the existence probability of the object can be increased. Otherwise, a new object can be created. If the existence probability exceeds a set threshold, the object can be considered part of the map.

With the method as described so far, the existence probability of objects can only increase. In order to be able to remove falsely detected objects, there has to be a way to reduce the probability in case an object is not detected. For this purpose, the visibility area can be used: For each object in the map within the visibility area of a sensor measurement which has not received new evidence, the existence probability can be reduced.

This method was implemented for the semantic mapping node. Implementation details can be found in Section 3.4.5.

3.4.2 Mapping process

The semantic mapping process can be divided into three main tasks. First, the objects have to be detected and labeled in the images or point clouds. Then, these object hypotheses have to be transformed to shape hypotheses on the map in the appropriate coordinate frame. Lastly, multiple shape hypotheses from different camera view have to be combined to build a coherent semantic map of the environment. Figure 8 shows the process of extracting shape hypotheses from RGB-D camera images, which is described in detail in the following subsections.

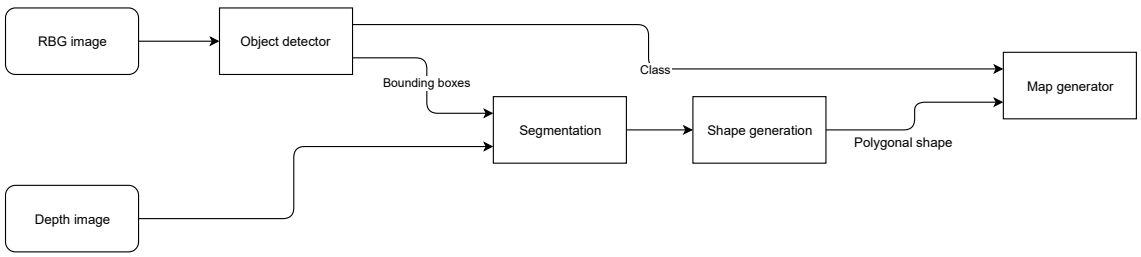


Figure 8: Semantic mapping process.

3.4.3 Object detection

The object detection is achieved by a neural network. As discussed in Section 2.3, possible methods are simple bounding box object detection algorithms, more complex instance segmentation variants, or object detection in 3D. While 3D object detection would provide the easiest way to convert the detected objects to the map frame, they are also the least researched and tested type of object detectors. Well implemented and pre-trained networks are difficult to find, and also training and test datasets are rarer. For example, the most common dataset for training and analyzing object detection performance, COCO [37], only provides RGB data, not depth.

When using bounding box object detection, the problem of determining which pixels belong to the object and should be used to determine the shape on the map arises - the box could contain several pixels belonging to the background or other

objects. Using instance segmentation methods would only partly solve this problem: The mask is provided in the image frame. Some pixels close to the object that actually belong to the background could be included in the mask. While these pixels are close to the object in the image frame, their depth might differ significantly. Therefore, additional filtering would have to be performed.

For the current implementation, it was decided to use YOLO for object detection. An implementation for YOLO in ROS is available [38].

3.4.4 Shape generation

In order to convert the detected objects to shapes on the map, the bounding box messages are first synchronized with the point cloud messages. The points inside a bounding box are then segmented using the region growing algorithm provided by PCL. Out of the generated clusters, the biggest one is used for generating the object shape. All points belonging to that cluster are projected to the x-y-plane of the map frame. The projected point cloud is then downsampled using a voxel grid filter. Finally, the convex hull of them is determined.

The determined shape together with the detected class of the bounding box is then passed to the "add evidence" function of the map generator. This is repeated for all detected bounding boxes in a frame. Additionally, the visibility area of the robot is determined by projecting the complete point cloud on the x-y-plane and then generating the convex hull. This area is passed to the "remove evidence" function of the map generator.

3.4.5 Map generation

The map is stored as a list of semantic objects. As discussed in Section 3.4.1, a semantic object in the context of the used mapping method consists of a name, a list of possible shapes, a combined shape hypothesis for the object, as well as an existence probability. Additionally, bounding box and centroid information are stored. Each object has a unique identification number (ID). The objects are stored in a map with their IDs as key. The IDs are additionally stored in an R-tree, which uses the bounding boxes of the objects as key. The R-tree is used for spatial queries.

When a new shape is added as evidence to the map, all objects that the new shape could belong to are collected. Different methods were tested to collect these objects. One approach is to use objects with the same name of which the Jaccard index with the new shape is higher than a set threshold. Since this approach was problematic when dealing with thin shapes, an alternative method was to use all objects within a defined radius around the detected shape. After refining the segmentation algorithm to find larger segments, thin shapes were rarely detected, so that the former approach could be used. For the final tests, a Jaccard index threshold of 0.2 was used. Higher thresholds lead to most objects not being detected. The method was also compared to the radius method, where a search radius of 0.5 m was used. The result of this experiment can be found in Section 5.2.4.

The existence probability is not stored as a normal probability, but as a log probability. Therefore, instead of having probabilities between 0 and 1, a logarithmic

scale is used. This provides the benefit that probabilities can be combined by addition instead of having to use multiplication. New objects start with an existence probability of 1. If evidence is added to an object, the probability is increased by 1. Furthermore, for all objects within the visibility area, the probability is decreased by 0.2. To allow easier implementation, it is not checked whether evidence was added to an object in the same sensor measurement. Therefore, the probability increases effectively by 0.8 if evidence is collected.

If multiple objects are found as match for a new shape, options to either combine the found objects or add the shape to each object separately were considered. For the final tests, the objects were combined, as the latter approach could lead to duplicate objects at the same position. If two objects are combined, the shape lists are concatenated and the existence probability is determined as the sum of the existence probabilities of the combined objects, as the evidence of both objects now indicates the existence of the new object.

For generating the shape hypothesis of an object, both selecting only one shape and combining multiple shapes was tested. In order to select the best fitting shape, the centroid of each shape as well as the average centroid are stored and updated with the semantic object. The shape whose centroid is closest to the average is then selected as most likely shape. For the method of combining the shapes, the convex hull of all shapes in the list was computed. The result of this test can be found in Section 5.2.3.

Whenever the map is updated, it is published on the topic “semantic_map”. The existence probability threshold for adding an object to the published map was set to 2.

4 Framework tests

For testing the framework, each of the requirements defined in Section 3.2 was checked. Some requirements were tested manually, while for others, test code was designed.

F1 regarding the ability to combine separately saved maps was tested with several occupancy maps and manually created semantic maps. In order to combine the maps, a configuration file had to be written, as specified in Section 3.3.5. The configuration file was then loaded and the map saved with the map server command-line interface. The resulting archive was then reviewed in an archive viewer. It was shown that the resulting archive contained all necessary files.

F2 regarding the ability to initiate a mapping process is fulfilled by provided launch files. For occupancy grid mapping, a launch file that creates a hypermap with a single occupancy layer in subscribe mode was created. The occupancy layer then subscribes to the topic "map", which is published by the gmapping node. That way, the generated occupancy map can be saved from the map server node. For semantic mapping, the launch file creates a hypermap with both occupancy and semantic layer in subscribe mode. The semantic layer subscribes to the topic "semantic_map", which is published by the semantic mapping node.

F3 to F5 do not need testing, as the existence of the layers and the map server node was already shown in Section 3. The same is valid for F9 regarding the publishing of messages and F20 regarding the type of messages for the occupancy layer.

In order to test F6 and F7 regarding saving and loading map files as well as F18 and F19 (saving and loading of the layers), various hypermap test files were manually created. For testing the metadata load functionality, maps with only one occupancy or semantic layer as well as maps with both layers were created. The maps were loaded using the command-line interface of the map server and then saved again. The saved maps were then compared to the original maps. It was shown that the saved maps were the same as the loaded maps.

For the occupancy grid layer, occupancy maps with various formats (pgm, png with transparency, png without transparency) were used for testing, with the modes trinary and scale. All tested formats could be successfully loaded.

The semantic map loading and saving was first tested with a manually created dummy map with a few objects. In the later stages, the automatically generated maps were also used. Both the manually created and the automatically generated maps could be loaded by the map server.

F8 and F17 regarding the visualization of the layers in rviz did not need to be tested for the occupancy grid layer, since a display for the published messages is already included in ROS.

For the semantic layer, the visualization plugin was tested with the created test maps, as well as during the mapping process. Adjusting settings such as which classes to display and changing the fixed frame of rviz were tested and it was verified that the display was updated correctly.

The hypermap display plugin that subscribes to the metadata was tested by loading maps with different layers while the plugin was loaded in rviz. It was verified

that the layers are updated correctly if the map changes.

In order to test the functionality of the node services (F10 to F13 and F15 to F16), a test suite was created based on the rostest package, which utilizes the Google Test framework to enable Unit tests. For the tests, a map server node was created and a test map was loaded. Then, the result of different query types was compared to the expected outcome, which was determined manually from the map. All unit tests succeeded.

F14 regarding the providing of tf transforms was tested with a manually created map. Occupancy and semantic map were specified in different frames, and a transform was added to the configuration file. On the map, queries were performed using the command-line interface and it was checked whether the transformations were applied correctly.

Conclusion

The framework tests showed that the framework fulfills the set requirements. Files are loaded and saved correctly, different types of spatial and keyword queries work on both the semantic and the occupancy layer, and the map with all its layers can be displayed in rviz.

5 Semantic mapping experiments

In this section, the experiments performed to evaluate the semantic mapping algorithm are presented. Section 5.1 describes the conducted simulation experiments. In Section 5.2, experiments with the robot Care-o-bot 4 are presented.

5.1 Simulations

In order to test if the mapping algorithm is working as intended, the semantic mapping was first evaluated in a simulated environment. Two simulation environments were tested: The first one uses Gazebo [10], a robot simulation toolbox. The "Institut für Produktionstechnik und Automatisierung" (IPA), which developed the Care-o-bot 4 that was used for the robot experiments, provides sample environments like a kitchen and a floor with a simulated Care-o-bot robot for Gazebo. The second one is the Gibson environment [39] developed at Stanford University, which includes multiple virtualized floors and buildings and provides a ROS binding with a simulated TurtleBot.

Both environments provide a movable robot equipped with an RGB-D camera setup, which is needed for the semantic mapping. First, it was tested if the object detection network works with the simulated images. In the Gazebo environment, some textured elements, like an oven, a microwave and a refrigerator, could be recognized. Figure 9 shows the environment with oven and microwave recognized. Objects with missing textures were usually not recognized. In the Gibson environment, the camera

[10] http://gazebosim.org/

image was very blurry, which led to very few recognitions. Therefore, the Gazebo environment was used for the first test phase, as at least some objects were detected reliably.

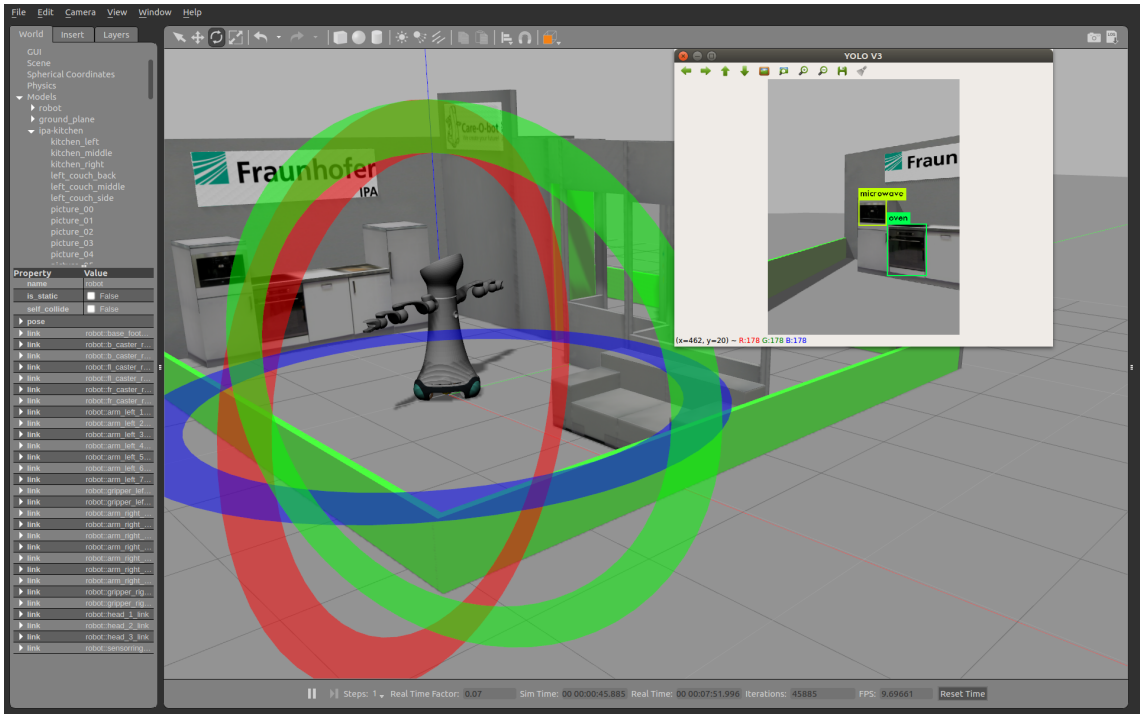


Figure 9: Simulated kitchen in Gazebo.

One problem in the simulated environment arose when converting the detected point cloud to a polygon on the map. Since the detected objects were flat surfaces with a texture, the resulting polygons were often just single lines on the map. With lines, even a small deviation in position or orientation leads to no overlapping area. Therefore, if the overlap percentage was used to determine if two shapes belong to the same object, detections of the same object were often not correctly grouped together. However, when using a search radius to combine the shapes, the detected objects could successfully be placed on the map.

5.2 Experiments with Care-o-bot

The experiments in the simulated environment showed the general suitability of the approach. However, real environments are usually more chaotic and generate noisier data. To test whether the algorithm can handle these additional challenges, experiments with data recorded with a Care-o-bot 4 robot were conducted. Section 5.2.1 describes the setup for these experiments, including a description of the robot and the environments as well as the methods used to analyze the quality of the generated maps. Section 5.2.2 presents the results acquired from analyzing the segmentation procedure. In Section 5.2.3, two methods for generating the shape of the objects are compared. Section 5.2.4 compares two methods of selecting which evidence to group.

In Section 5.2.5, the results of the experiments are discussed.

5.2.1 Experiment setup

Care-o-bot 4

For the experiments, a Care-o-bot 4 [11] produced by the IPA was used. The robot is equipped with three laser scanners to provide omnidirectional obstacle detection. The base can be moved in any direction. Spherical joints also allow for 360° rotations without moving the robot. Additionally, the robot contains multiple RGB-D cameras. For the conducted mapping experiments, the camera placed on the sensor ring below the head of the robot was used, as it provided the most complete view of the environment in front of the robot. Furthermore, the robot has two arms with grippers, which were not used for the experiments.

Data acquisition

For acquiring experiment data, the rosbag tool of ROS was used to record relevant sensor data, which was then used for the tests. This provides the advantage over online testing that different methods can be tested with the same input data. The recorded sensor data for the first tests included the laser scanner, the odometry, transform topics, the raw RGB camera image and the depth point cloud. Since the raw image and the point cloud contain a lot of data leading to large file sizes for the records, these two were replaced with the compressed RBG and depth images for later recordings. The depth image was then converted to a point cloud while running the tests.

A problem that arose during testing was that when recording RGB and depth image simultaneously, the RGB image was not received anymore if the camera was pointed towards an open area with mostly far away points. The issue was resolved by reducing the resolution of the used Asus XTion camera from 640x480 to 320x240.

Two recordings were chosen to analyze the mapping process, one of a laboratory and one of a floor. Occupancy maps were created using the SLAM implementation of the gmapping package [12] from the laser scanner and the odometry. This package uses the OpenSLAM Gmapping implementation that was described in Section 2.1.2.

Map quality analysis

In order to evaluate the quality of the automatically generated semantic maps, ground truth semantic maps for these two locations were created manually by evaluating the camera and point cloud data.

Additionally to a visual comparison, the maps were compared with a Matlab script using the following algorithm.

1. For each shape of specified or all classes in the ground truth and the generated map: Calculate the centroid of the polygon.

[11] https://www.care-o-bot.de/en/care-o-bot-4.html
[12] http://wiki.ros.org/gmapping

2. For each shape of the classes in the ground truth: Find the closest centroid of the same class in the generated map.

3. Translate the found shape so that the centroids align.

4. Calculate the Jaccard index of the shapes.

5. The average of the closest centroid distances and the calculated Jaccard indices are used as measures for the quality of the map.

These metrics were computed for various test runs with different parameters or algorithms. In order to determine if an observed difference in the metrics is significant, the p-value of a paired, one-sided Wilcoxon sign rank test was computed. The test has the null hypothesis that the difference between the two datasets comes from a distribution with zero median. If this null hypothesis can be rejected at a sufficient significance level (e.g. $p < 0.05$), it can be assumed that the deviation in the metrics is not only due to random variations.

Experiment environments

The first experiment environment was the home laboratory of the robot. Figure 10 shows the manually labeled ground truth map. It contains multiple chairs, tables with monitors, keyboards, mouses and cups, a sink, a cupboard, a bin, a fan and a noteboard. Table, cupboard, fan and noteboard are not recognizable classes for the used object detection network, and can therefore not expected to be found in the automatically generated maps, but were included to provide a more complete overview of the lab environment. In the recorded sequence, the robot moved for 143 seconds through the laboratory, moving from its home position to the door and back while rotating a few times.

The second experiment environment was the floor of the robot laboratory. Figure 11 shows the corresponding ground truth map. The most common types of objects are benches and potted plants, but the environment also contains some tables, chairs and monitors. The recorded sequence was 7 minutes long, during which the robot moved one time around the floor.

5.2.2 Segmentation analysis

In order to figure out the quality of the point cloud segmentation with different parameters, the segments were viewed in PCL's point cloud viewer. As described in Section 2.4.1, the used region growing algorithm has two parameters: The angle threshold θ_{th}, which determines the maximum angle between normals for a neighboring point to be considered as part of the same segment, and the curvature threshold c_{th}, which is the maximum curvature for a point to be added to the seed set. Multiple values for these parameters were tested and the results on the segmentation of different objects were compared. In this section, the resulting segmentation for two objects (a monitor and a chair) in the laboratory environment is shown. The

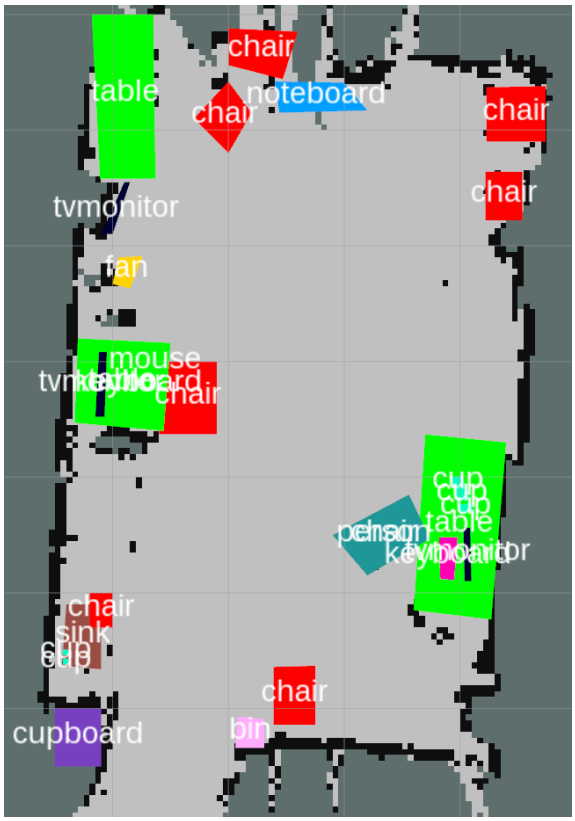


Figure 10: Laboratory map ground truth.

curvature threshold was set to 1.0, and the difference between an angle threshold of 3° and 20° is compared.

Figure 12 show the resulting segmentation of a chair, Figure 13 the segmentation of a monitor. Red pixels do not belong to any segment. For other colors, the color indicates the segment. Note that the segmentation was only performed in the area specified by the detection bounding box. With an angle threshold of 3°, the objects are divided into many small segments. With an angle threshold of 20°, the back of the chair is visible as one purple segment in Figure 12b, and the whole monitor in Figure 13b is one segment.

After testing different thresholds, $\theta_{th} = 20°$ and $c_{th} = 1.0$ were chosen as final values. While these might not be the optimal thresholds in all cases, they proved to provide reasonable segmentation on many objects.

5.2.3 Object shape generation

The goal of this test was to compare the map quality for two different ways of generating the shape of the objects. The first way is to select the one of the detected shapes as shape for the object. To select the shape, the average of the centroid of all shapes is computed. The shape for which the centroid is closest to the average is selected. In the following, this will be called the “best shape method”.

The second way is to compute the convex hull of all shapes that were collected

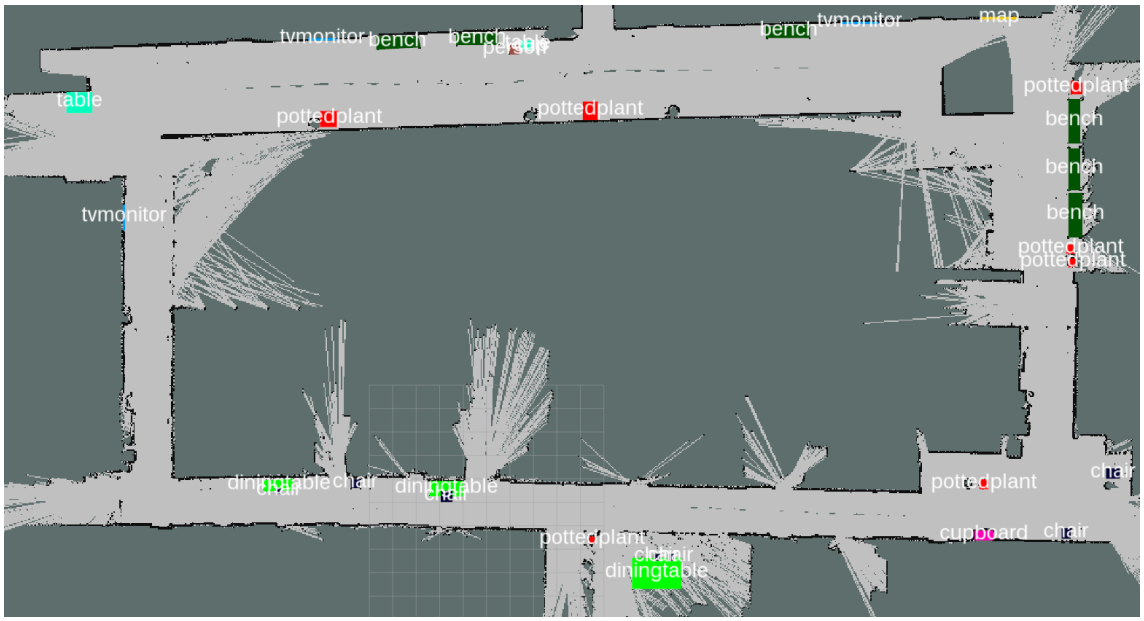
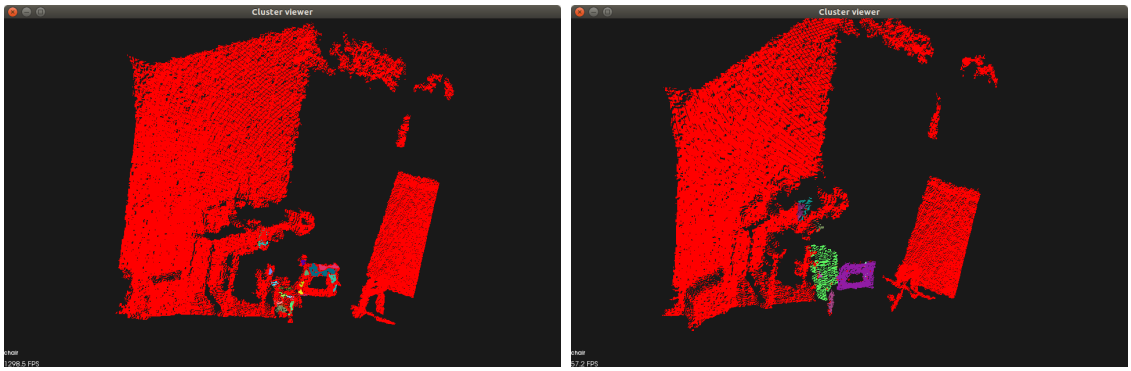


Figure 11: Floor map ground truth.



(a) $\theta_{th} = 3°$ (b) $\theta_{th} = 20°$

Figure 12: Chair segments.

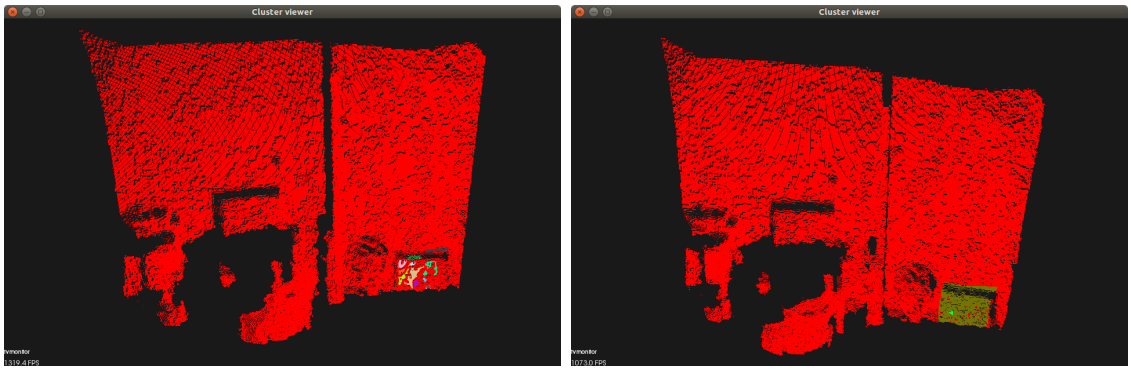
as evidence. This will be called the "convex hull method".

For both test, the Jaccard index was used to choose which shapes to group together. The minimum Jaccard index for a new shape to be considered evidence for an existing shape was set to 0.2.

Laboratory

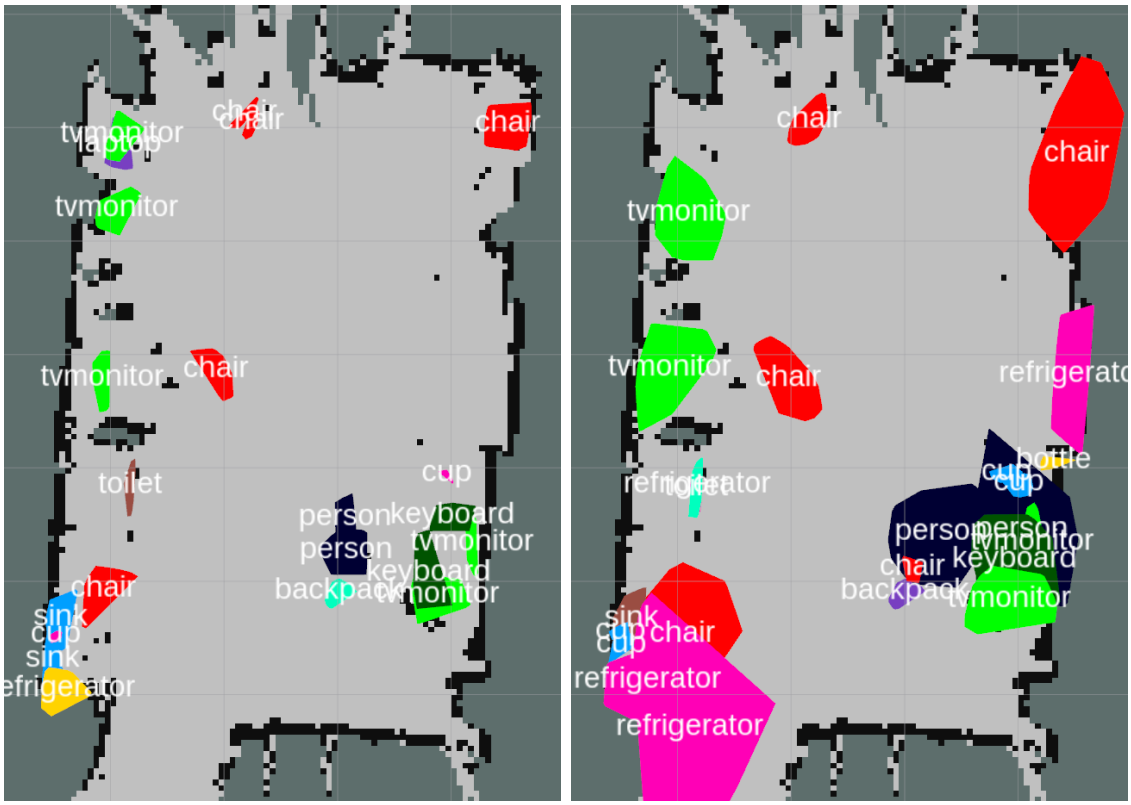
Figure 14 shows the two generated maps in the lab environment. Figure 14a shows the map for the best shape method and Figure 14b for the convex hull method. The computed metrics are displayed in Table 4.

Subjectively, Figure 14a looks cleaner and the shapes more accurate than in Figure 14b. This is confirmed by analysis: The average Jaccard index of ground truth objects with matching objects in the generated maps is 0.41 for the best shape



(a) $\theta_{th} = 3°$ (b) $\theta_{th} = 20°$

Figure 13: Monitor segments.



(a) Best shape (b) Convex hull

Figure 14: Resulting maps - Laboratory.

method and only 0.26 for the convex hull method, with a p-value of 0.023. Regarding the centroid distance, the convex hull method achieves a lower average deviation with 0.41 m compared to 0.51 m. However, the p-value of 0.23 is high, indicating that this difference could be caused due to random variation.

When looking at classes separately, the biggest difference in shape accuracy is observable for the monitors: The average Jaccard index is 0.46 for the best shape method compared to 0.15 for the convex hull method. Since there are only three

Table 4: Shape generation experiment results - Laboratory.

	Jaccard index			Distance (m)		
	Best shape	Convex hull	p	Best shape	Convex hull	p
all	0.41	0.26	0.023	0.51	0.41	0.23
tvmonitor	0.46	0.15	0.13	0.063	0.077	0.63
chair	0.41	0.31	0.23	0.71	0.48	0.42
cup	0.43	0.19	0.031	0.11	0.085	0.22

monitors in the ground truth, the significance of this result is limited ($p = 0.13$). When looking at the detected shapes, for the convex hull method the monitors are round rather than having a slim monitor-like shape. An explanation is that small misalignments can lead to a significant deformation for slim shapes compared to rounder shapes like for example chairs.

For chairs, the results are less conclusive. The chair on the top right corner of the room is recognized accurately with the best shape method, but on the other hand, the second chair below is not found. With the convex hull method, both chairs are combined to a larger single chair. The chair in front of the noteboard on the top left is recognized as two chairs for the best shape method, and it seems to have only captured the back of the chair as shape. The convex hull method has combined this to a single shape. The chair below the sink is more accurately shaped with the best shape method, as it is extended to the background for the convex hull method. Since this chair was only a stool, a lot of background is in the detection bounding box, which may have led to the selected segments sometimes consisting of the background.

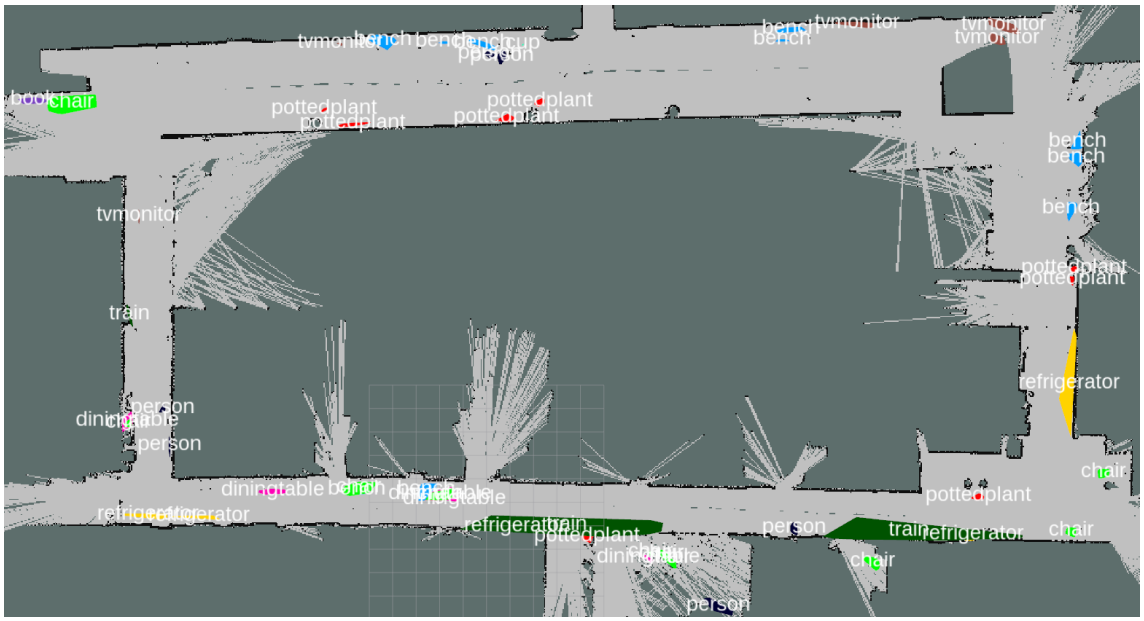
The mathematical analysis shows an average Jaccard index of 0.41 for the best shape method compared to 0.31 for the convex hull method for chairs. The p-value of 0.23 shows however that this is not a significant difference.

Regarding cups, it is first noticeable that with the convex hull method, both cups on the sink are recognized as separate objects and two of the three cups on the desk on the right, while with the best shape method, only one of the cups on each side is displayed. However, the shapes are more accurate with the best shape method, as the Jaccard index of 0.43 compared to 0.19 for the convex hull method. While the p-value of 0.031 shows significance, it as to be considered that the used algorithm does not prevent the same detected shape from being compared to multiple ground truths. Therefore, while 5 different ground truth shapes were used, only two separate detected shapes were compared to them in the case of the best shape method, which might reduce the significance of the result.

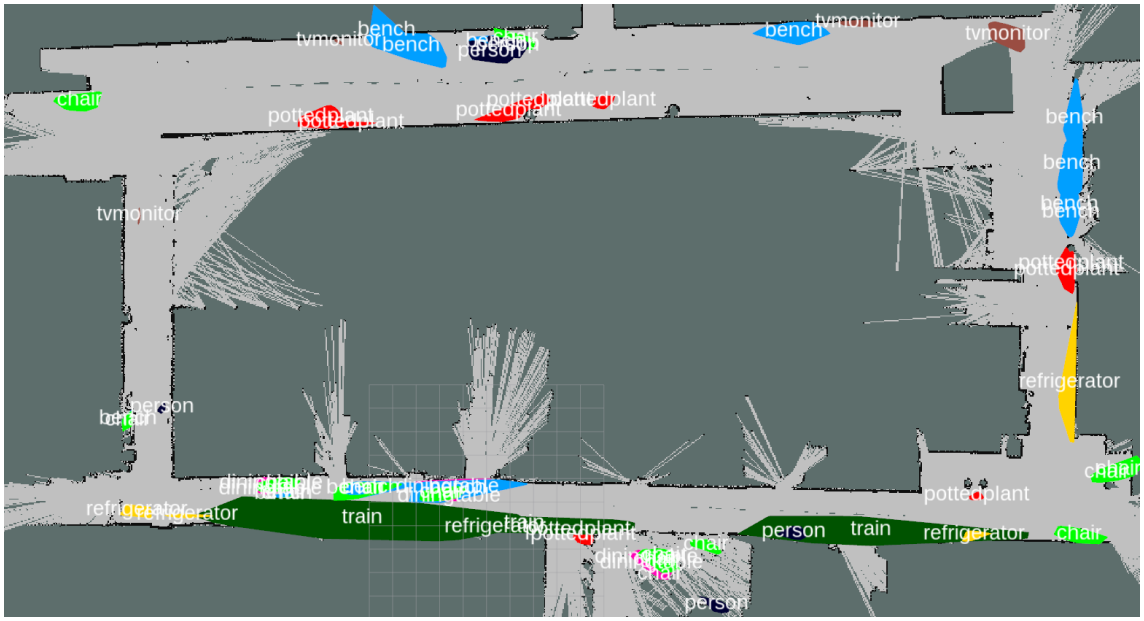
Floor

Figure 15 shows two generated maps in the floor environment. The computed metrics are show in Table 5. Both maps contain large false detections of trains and refrigerators. This could be avoided by training the object detection network for indoor environments instead of using the pre-trained model for the COCO dataset.

Again, the generated map where the best shape was used (Figure 15a) looks



(a) Best shape



(b) Convex hull

Figure 15: Resulting maps - Floor.

cleaner, but the shapes are not clearly more accurate anymore. The average Jaccard index computes to 0.36 compared to 0.35 for the convex hull method (Figure 15b). The average centroid deviation is again lower for the convex hull method (0.63 m vs 0.86 m). This time, the difference is significant with a p-value of 0.024.

When looking at classes separately, it is noticeable that the benches are mostly small spots in the best shape map, while the bench shapes for the convex hull method resemble the actual bench shapes more. Looking at the Jaccard index, this impression

Table 5: Shape generation experiment results - Floor.

	Jaccard index			Distance (m)		
	Best shape	Convex hull	p	Best shape	Convex hull	p
all	0.36	0.35	0.50	0.86	0.63	0.024
bench	0.27	0.44	0.047	0.46	0.34	0.34
pottedplant	0.32	0.43	0.19	1.77	1.49	0.11

is confirmed with an index of 0.44 for the convex hull method compared to 0.27 for the best shape method. The p-value of 0.047 shows that this can be considered significant evidence. An explanation could be that the segmentation algorithm tends to divide benches in multiple segments, and therefore, a combination of shapes is required to capture the whole bench area. No significant difference can be found in the centroid deviation for benches.

For potted plants, both the Jaccard index (0.43 vs 0.32) and the average centroid deviation (1.49 m vs 1.77 m) is slightly better for the convex hull method, but with no significant p-values. Note that the average centroid deviation is unusually high because one of the potted plants (the one in the top right corner) was not detected at all, leading to the next shape being more than 7 m away for this plant. If this plant is excluded from the analysis, the deviation shrinks to 0.45 m and 0.49 m respectively.

Conclusion

It is not clear which method produces the better map. The best shape method seems to produce better shapes for some objects (e.g. monitors), while the convex hull method works better on others (e.g. benches). For a lot of objects, the results are inconclusive. More data would need to be collected in order to determine which method is better. The convex hull method often seems to produce less deviation in centroid distance. However, this was only significant when comparing all objects in the floor environment. Again, more data would be necessary to confirm this hypothesis.

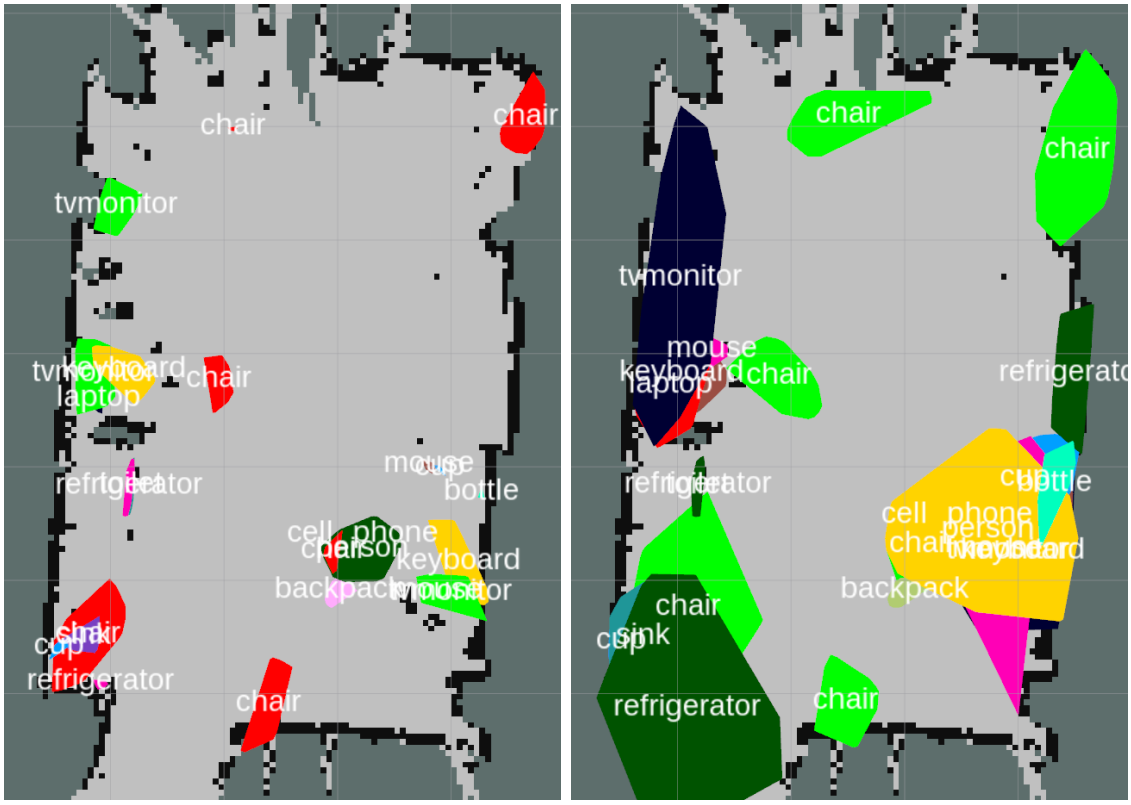
5.2.4 Object search

The goal of this test was to compare two methods of selecting which shapes to group as evidence. In the previous test, the Jaccard index between a new shape and existing objects was calculated. If it exceeded a set threshold (0.2 for the maps in Section 5.2.3), the shape is added as evidence to the object. In the following comparison, this will be called “overlap method”.

The second method was to find all objects with a set radius around the centroid of the new shape. Since intersections with circles are difficult to calculate within the used boost geometry library, an axis aligned bounding box was used. The “radius” (half of the side length) of this box was set to 0.5. This method will be called “radius search method”.

Laboratory

Figure 16 shows the maps generated with the radius search method. Table 6 shows the results in combination with the best shape method, Table 7 in combination with the convex hull method. With the best shape method (Figure 16a), the map looks similar to the one generated with the overlap method (Figure 14a). However, an analysis shows that the Jaccard index is worse (0.29 compared to 0.41), which is significant with a p-value of 0.0008.



(a) Best shape (b) Convex hull

Figure 16: Radius search maps - Laboratory.

Table 6: Object search experiment result - Laboratory, Best shape method.

	Jaccard index			Distance (m)		
	Overlap	Radius	p	Overlap	Radius	p
all	0.41	0.29	0.00080	0.51	0.26	0.62
tvmonitor	0.46	0.18	0.13	0.063	0.18	0.25
chair	0.41	0.29	0.0039	0.71	0.40	0.53
cup	0.43	0.27	0.031	0.11	0.14	0.31

Looking at the chairs, the chair at the bottom next to the door got recognized additionally to the chairs recognized with the overlap method. A reason could be

Table 7: Object search experiment result - Laboratory, Convex hull method.

	Jaccard index			Distance (m)		
	Overlap	Radius	p	Overlap	Radius	p
all	0.26	0.20	0.017	0.41	0.30	0.67
tvmonitor	0.15	0.023	0.13	0.077	0.56	0.13
chair	0.31	0.38	0.53	0.48	0.36	0.16
cup	0.19	0.024	0.031	0.085	0.10	0.094

that more evidence is combined when using the radius search, and therefore enough evidence could be collected. However, the average Jaccard index with the ground truth for chairs shows that the shapes are less accurate (0.29 compared to 0.41, with a p-value of 0.0039).

The monitors also seem to have a worse shape accuracy, with an average Jaccard index of 0.18 compared to 0.46. However, as mentioned in Section 5.2.3, there are only three monitors in the ground truth, which is no enough for significant evidence.

For cups, the average Jaccard index is only 0.27 compared to 0.43, with a p-value of 0.031. As mentioned in Section 5.2.3, due to only two cups being detected, the significance of this result is also limited.

For the map that uses the convex hull method for shape combination (Figure 16b), it is noticeable that this produces even bigger shapes in combination with the radius search method than with the overlap method. The Jaccard index is only 0.20 compared to 0.26, with a p-value of 0.017.

For monitors, it can be noticed that the two monitors on the left got combined to one large monitor, which produces a way too large monitor shape. Therefore, the average Jaccard index is only 0.023 for the radius search method, compared to 0.15 for the overlap method.

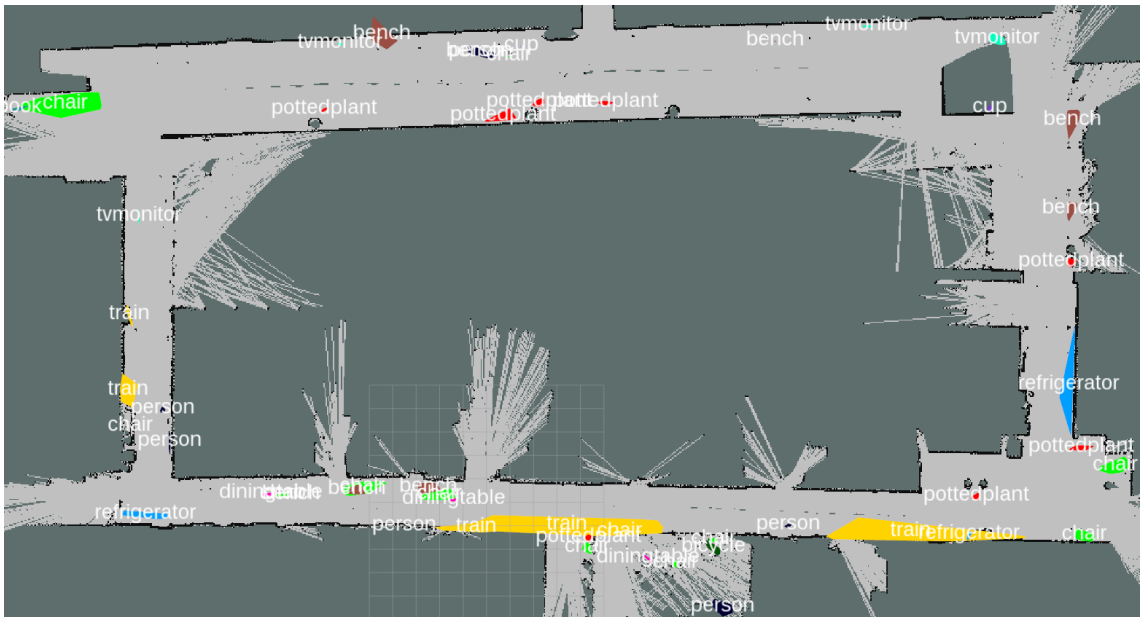
For chairs, the result is less conclusive. The Jaccard index is 0.38 compared to 0.31 for the overlap method. With a p-value of 0.53, the difference is not significant. As in Figure 14a, the additional chair at the bottom was also recognized here.

The cup shapes are less accurate for the radius search method, as multiple cups get combined to one long shape. Unlike in Figure 14b, where two separate cups were detected both on the sink and on the desk, only one cup on each side is recognized here. The average Jaccard index is only 0.024 compared to 0.19, with a p-value of 0.031.

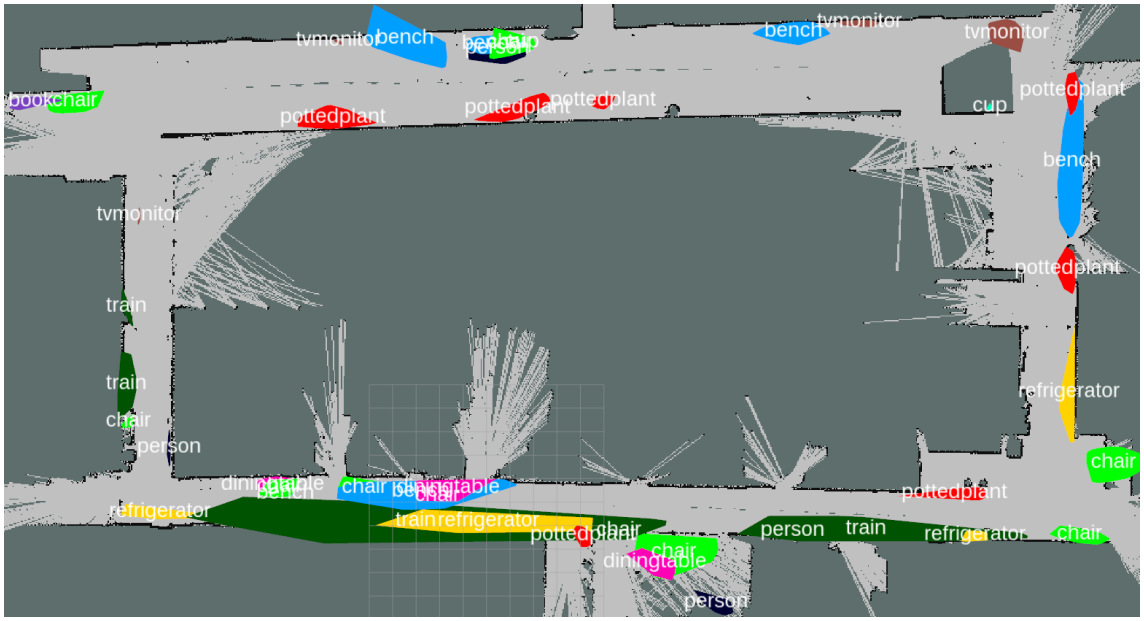
Floor

Figure 17 shows the maps generated in the floor environment with the radius search method. In Figure 17a, the best shape method was used, and in Figure 17b, the convex hull method. Table 8 shows the results in combination with the best shape method, Table 9 in combination with the convex hull method.

The results in this environment are less conclusive than in the laboratory environment. For the best shape method, the average Jaccard index of the radius search map with the ground truth is similar to the map from the overlap method (0.33



(a) Best shape



(b) Convex hull

Figure 17: Radius search maps - Floor.

compared to 0.36, with a p-value of 0.20).

When combined with the convex hull method, the average Jaccard index with radius search is only 0.25 compared to 0.35 with the overlap method. This result is significant, with a p-value of 0.00014.

When looking at the benches, it is noticeable that the three benches on the right were combined to a single bench. Comparing the shape quality shows a lower average Jaccard index of 0.31 compared to 0.44 with the overlap method. This result is

Table 8: Object search experiment result - Floor, Best shape method.

	Jaccard index			Distance (m)		
	Overlap	Radius	p	Overlap	Radius	p
all	0.36	0.33	0.20	0.86	0.82	0.79
bench	0.27	0.22	0.31	0.46	0.69	0.22
pottedplant	0.32	0.45	0.078	1.77	1.44	0.031

Table 9: Object search experiment result - Floor, Convex hull method.

	Jaccard index			Distance (m)		
	Overlap	Radius	p	Overlap	Radius	p
all	0.35	0.25	0.00014	0.63	0.57	0.88
bench	0.44	0.31	0.047	0.34	0.92	0.34
pottedplant	0.43	0.23	0.016	1.49	0.59	0.72

significant, as the p-value is 0.047.

For the potted plants, the average Jaccard index is 0.23 compared to 0.36 for the Jaccard index method. With a p-value of 0.016, this is also significant.

Conclusion

The result of the comparison is not completely conclusive, but overall, the overlap method shows advantages over the radius search method. While sometimes more evidence can be combined with the radius search method, leading to additional objects being recognized, it can also cause different object to be combined to a single large object. This especially happens when combined with the convex hull method for shape generation, as the larger shapes are more likely to be combined.

5.2.5 Discussion of experiment results

The experiments proved that meaningful semantic maps can be generated with the proposed algorithm. While the shapes and positioning of the objects is not completely accurate and the maps contain some false detection, most detectable objects could successfully be recognized and placed on the map.

Regarding the different methods for generating the shapes and combining the collected evidence, the experiments indicate that the overlap method works better for combining evidence, especially if neighboring objects of the same type should be differentiated. For the shape generation, the quality of the shapes for the different methods seems to depend on the object type. More experiments would be necessary to determine which method works better on average.

An additional consideration is that the best methods depend on the quality of the initial shape hypotheses. If for example the segmentation algorithm would be improved to produce more accurate initial shapes that most likely contain the whole object, the quality of the resulting maps could improve significantly.

6 Conclusion

In this thesis, a framework for managing multi-layered maps as well as a mapping algorithm for generating semantic maps were developed.

The hypermaps framework offers an occupancy and a semantic layer. It allows performing spatial and thematic queries on all layers. The maps can be saved to and loaded from a single file, but also be loaded from multiple separate maps using a configuration file. Both the occupancy and the semantic layers can be displayed separately in rviz, or the metadata topic can be utilized to automatically detect and display all layers.

The framework fulfills the set requirements. It is also extensible so that in the future, more layers to display other kinds of maps can be added.

The semantic mapping process has proven to be able to generate maps with reasonable hypotheses for object placements and shapes. While the shapes are not always accurate, they can still be used for planning purposes. A possible application that does not require the exact shape is finding the nearest object of a specified type, e.g. a cup or chair. For these types of applications, the performance of the mapping algorithm is sufficient.

In future work, the mapping process could be improved in multiple stages. For the object detection, a pre-trained network was used in this work, which, while providing generally good detections, also generated some false object classifications (for example, the floor wall was sometimes recognized as a train). Here, training the network specifically for the environment it is used in (e.g. the office environment) could significantly reduce false detections.

Furthermore, the segmentation step was not always able to capture complete objects as individual segments. One way of improving this would be to use a more intelligent segmentation algorithm. Another possibility would be utilize an object detection algorithm that directly works with point clouds.

An additional error factor was the localization on the map. The current method is not a SLAM method, it depends on the occupancy map for estimating the robot position. While a decent localization is provided most of the times, some inaccuracies occurred. A possible improvement of the mapping algorithm could take these inaccuracies into account and relocate previously recognized shapes if the position gets corrected.

Aside from improving the performance of the semantic mapping, future work could also focus on finding application scenarios for the hypermaps framework. A possible application would be an automated semantic mapping process that utilizes the occupancy layer to systematically explore the known map while building the semantic layer.

References

[1] M.-J. Kraak and R. Van Driel, "Principles of hypermaps," *Computers & Geosciences*, vol. 23, no. 4, pp. 457–464, May 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0098300497000101

[2] P. Corke, "Navigation," in *Robotics, Vision and Control: Fundamental Algorithms in MATLAB®*, ser. Springer Tracts in Advanced Robotics, P. Corke, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 87–106. [Online]. Available: https://doi.org/10.1007/978-3-642-20144-8_5

[3] R. Siegwart, I. R. Nourbakhsh, D. Scaramuzza, and R. C. Arkin, *Introduction to Autonomous Mobile Robots*. Cambridge, UNITED STATES: MIT Press, 2011. [Online]. Available: http://ebookcentral.proquest.com/lib/aalto-ebooks/detail.action?docID=3339191

[4] L. Zhang and B. K. Ghosh, "Line segment based map building and localization using 2d laser rangefinder," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 3, Apr. 2000, pp. 2538–2543 vol.3.

[5] G. Foux, M. Heymann, and A. Bruckstein, "Two-dimensional robot navigation among unknown stationary polygonal obstacles," *IEEE Transactions on Robotics and Automation*, vol. 9, no. 1, pp. 96–102, Feb. 1993.

[6] M. Missura, D. D. Lee, and M. Bennewitz, "Minimal Construct: Efficient Shortest Path Finding for Mobile Robots in Polygonal Maps," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2018, pp. 7918–7923.

[7] N. Sleumer and N. Tschichold-Gürmann, "Exact cell decomposition of arrangements used for path planning in robotics," *Technical report/ETH Zürich, Department of Computer Science*, vol. 329, 1999.

[8] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on Robotics*, vol. 23, no. 1, p. 34, 2007.

[9] S. Thrun and A. Bücken, "Integrating grid-based and topological maps for mobile robot navigation," in *Proceedings of the National Conference on Artificial Intelligence*, 1996, pp. 944–951.

[10] O. Booij, B. Terwijn, Z. Zivkovic, and B. Krose, "Navigation using an appearance based topological map," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, Apr. 2007, pp. 3927–3932.

[11] I. Kostavelis and A. Gasteratos, "Semantic mapping for mobile robotics tasks: A survey," *Robotics and Autonomous Systems*, vol. 66, pp. 86–103, Apr.

2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889014003030

[12] G. Costante, T. A. Ciarfuglia, P. Valigi, and E. Ricci, "A transfer learning approach for multi-cue semantic place recognition," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov. 2013, pp. 2122–2129.

[13] D. Meger, P.-E. Forssén, K. Lai, S. Helmer, S. McCann, T. Southey, M. Baumann, J. J. Little, and D. G. Lowe, "Curious George: An attentive semantic robot," *Robotics and Autonomous Systems*, vol. 56, no. 6, pp. 503–511, Jun. 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889008000316

[14] A. Nüchter and J. Hertzberg, "Towards semantic maps for mobile robots," *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 915–926, Nov. 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889008001127

[15] S. Sengupta, P. Sturgess, L. Ladický, and P. H. S. Torr, "Automatic dense visual semantic mapping from street-level imagery," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2012, pp. 857–862.

[16] O. M. Mozos, H. Mizutani, R. Kurazume, and T. Hasegawa, "Categorization of Indoor Places Using the Kinect Sensor," *Sensors*, vol. 12, no. 5, pp. 6695–6711, May 2012. [Online]. Available: https://www.mdpi.com/1424-8220/12/5/6695

[17] A. J. Trevor, S. Gedikli, R. B. Rusu, and H. I. Christensen, "Efficient organized point cloud segmentation with connected components," *Semantic Perception Mapping and Exploration (SPME)*, 2013.

[18] A. Pronobis and P. Jensfelt, "Large-scale semantic mapping and reasoning with heterogeneous modalities," in *2012 IEEE International Conference on Robotics and Automation*, May 2012, pp. 3515–3522.

[19] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2014, pp. 580–587.

[20] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot MultiBox Detector," in *Computer Vision – ECCV 2016*, ser. Lecture Notes in Computer Science, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Springer International Publishing, 2016, pp. 21–37.

[21] 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*, 2016, pp. 779–788.

[22] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[23] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[24] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[25] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, "YOLACT: Real-time Instance Segmentation," *arXiv:1904.02689 [cs]*, Apr. 2019, arXiv: 1904.02689. [Online]. Available: http://arxiv.org/abs/1904.02689

[26] M. Simon, S. Milz, K. Amende, and H.-M. Gross, "Complex-YOLO: An Euler-Region-Proposal for Real-Time 3d Object Detection on Point Clouds," in *European Conference on Computer Vision*. Springer, 2018, pp. 197–209.

[27] W. Ali, S. Abdelkarim, M. Zidan, M. Zahran, and A. El Sallab, "Yolo3d: End-to-end real-time 3d oriented object bounding box detection from lidar point cloud," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 0–0.

[28] "Region growing segmentation - Point Cloud Library (PCL)." [Online]. Available: http://pointclouds.org/documentation/tutorials/region_growing_segmentation.php

[29] N. Mamoulis, "Spatial Data Management," *Synthesis Lectures on Data Management*, vol. 3, no. 6, pp. 1–149, Nov. 2011. [Online]. Available: http://www.morganclaypool.com/doi/abs/10.2200/S00394ED1V01Y201111DTM021

[30] J. Nievergelt, H. Hinterberger, and K. C. Sevcik, "The grid file: An adaptable, symmetric multikey file structure," *ACM Transactions on Database Systems (TODS)*, vol. 9, no. 1, pp. 38–71, 1984.

[31] R. A. Finkel and J. L. Bentley, "Quad trees - A data structure for retrieval on composite keys," *Acta informatica*, vol. 4, no. 1, pp. 1–9, 1974.

[32] A. Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching," in *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '84. New York, NY, USA: ACM, 1984, pp. 47–57, event-place: Boston, Massachusetts. [Online]. Available: http://doi.acm.org/10.1145/602259.602266

[33] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," in *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '90. New York, NY, USA: ACM, 1990, pp. 322–331. [Online]. Available: http://doi.acm.org/10.1145/93597.98741

[34] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3. Kobe, Japan, 2009, p. 5.

[35] C. Glasbey, G. v. d. Heijden, V. F. K. Toh, and A. Gray, "Colour displays for categorical images," *Color Research & Application*, vol. 32, no. 4, pp. 304–309, 2007. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/col.20327

[36] R. Real and J. M. Vargas, "The probabilistic basis of Jaccard's index of similarity," *Systematic biology*, vol. 45, no. 3, pp. 380–385, 1996.

[37] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[38] M. Bjelonic, "YOLO ROS: Real-time object detection for ROS," https://github.com/leggedrobotics/darknet_ros, 2016–2018.

[39] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese, "Gibson Env: real-world perception for embodied agents," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9068–9079.