Thomas Senfter, Bsc

Active Object Search in Unknown Open Environments

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Information and Computer Engineering

submitted to

Graz University of Technology

Supervisor

Assoc.Prof. Dipl.-Ing. Dr.techn. Gerald Steinbauer

Institute for Softwaretechnology
Head: Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Slany

Graz, May 2018

This document is set in Palatino, compiled with pdfLaTeX2e and `Biber`.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Abstract

Everyday human environments have an underlying structure. Knowledge about this structure is used by humans to carry out tasks efficiently. In this thesis a robotic system is presented which uses such commonsense knowledge to execute an intelligent active object search in unknown open indoor environments. The main challenges are the selection of useful commonsense knowledge, the design of a suitable representation of the environment, and the development of a search planner which utilizes the gathered information. The concept of rooms, and commonsense knowledge about the connection between the room type of an area and the objects usually located in that area are utilized in this thesis. Using the commonsense knowledge, the room type of an area allows to estimate object probabilities in this area and the room type detected after peeking into a room allows to estimate the object probabilities in the whole room. A mapping system was developed which creates a semantic map of the environment containing the room structure and information about seen objects and room types of areas of the environment. This information is extracted from images using convolutional neural networks. Based on this map, a probabilistic planner generates view poses for the robot to drive to which minimize the expected search time. Test runs of the developed system showed a more intelligent behavior, where likely areas were searched first, and an improved performance compared to a coverage-maximizing search system. The findings of this work can be used in similar tasks with the goal to create robotic systems being able to help humans with a wide variety of tasks.

Contents

List of Figures

1 Introduction

With technical progress in the field of robotics and related areas, autonomous robots are now able to execute many tasks in factories, even without modifications to the environment like painted lines. However, robots struggle to conquer everyday human environments. Domestic environments might seem very chaotic at the first glance, with a wide variety of different types of environments and huge differences in their appearance. However, those environments have an underlying structure which is used by humans all the time to carry out tasks efficiently. Indoor environments are usually segmented into rooms which are separated by walls and only connected by doors. A room consists of one or a small number of areas, where each area has specific purposes, e.g. a kitchen is used for baking and cooking. For each purpose specific objects are used, e.g. a pan for cooking which can be assumed to be somewhere around.

The goal of this thesis is to equip a robot with commonsense knowledge about indoor environments to execute a task efficiently. The investigated task is to search an object in an unknown open indoor environment. This is a relative simple task, but can benefit a lot from commonsense knowledge about indoor environments, e.g. knowledge about room types and which objects are usually located in rooms of a certain type. It is also an essential skill for a robot working in domestic environments. Therefore, research on this topic will be useful to enable future robotic systems. During the execution of this task the robot has to find the best next view poses. In this thesis the best next view pose is the pose that minimizes the expected search time. The robot has to decide which areas are promising to contain the searched object. It also has to find a balance between exploration and the search of already explored areas because the environment is unknown at the start of a search. Here the commonsense knowledge is important to make better estimations about unseen or only vaguely seen areas.

The system presented in this work uses a hierarchical approach. The environment is seen as a topology of rooms at a higher abstraction level and the details are handled for every room separately. Within a room convolutional neural networks (CNNs) for object detection and place classification are used to create semantic maps containing information about the positions where objects were seen and the room types of areas in the room. Using the results of the object detection, the place classification and the connection between the type of an area and the objects usually located in that area, object probabilities are estimated, even for not or only vaguely seen areas. To achieve this a new mapping system was developed, capable to create hybrid maps which combine the room topology and semantic maps of the rooms. The developed mapping system is also able to cope with the uncertain results of the sensors and the CNNs. The hybrid map is then used by the developed hierarchical search planner which first selects the next room to search and then plans the search within that room, to execute the search task efficiently.

Similar systems were already developed, e.g. by Aydemir et al. [1]. However, the generality of the used CNNs are a step towards the flexibility which will be necessary in future robots used for example in households. The proposed system was tested in common indoor environments, where the behavior was investigated and a comparison with a system using no commonsense knowledge was done.

1.1 Challenges

The development of an intelligent active object search system poses various challenges. The main questions to answer during the development were:

- **What commonsense knowledge can be used to execute the object search intelligently:** Not only useful types of commonsense knowledge have to be found, but also ways to obtain and represent the knowledge.
- **What information the robot should gather and what sensors are necessary for that:** This includes not only the question of how to detect objects, but also what data is necessary for the navigation and

the application of the commonsense knowledge. Also suitable sensors have to be found.
- **How the gathered information about the environment should be represented:** This involves the design of a semantic map representation which contains all the gathered information. This design has to take into account the types and quality of the incoming data and has also to be suitable for the planning of the search.
- **How the search planning, using this knowledge, should be done:** The questions are how to use the gathered knowledge and especially how to handle the complexity of the active object search problem.

To reduce the amount of work suitable existing software packages have to be found and if necessary adapted. Furthermore, as this is a robotic system, many unexpected problems occur during the developed of the active object search system which have to be solved.

1.2 Contribution

In this thesis an intelligent object search system is presented and evaluated which can efficiently find objects of various different types in indoor environments. A hybrid semantic map design was therefore developed which represents the room structure of the environment as a topological map and the details of each room as separated semantic metric maps. These room maps contain, in addition to occupancy information, also information about detected objects and the room types of areas in the room. The developed system is also able to estimate object probabilities in vaguely seen areas and in partially explored rooms, using the information about the room types and commonsense knowledge about the connection between room types and objects usually located there. Furthermore, a hierarchical search planner was developed which in a first step selects the optimal next room to search based on the estimated object probabilities and the expected search times in the rooms and in a second step plans the search within the chosen room, where likely object locations are searched first. This enables the robot to search in likely rooms and room areas first and to postpone the search in unlikely rooms.

1.3 Outline

The structure of this thesis is as follows. In Chapter 2 a formal representation of the problem is given. Chapter 3 covers related research on the topic of active object search and its subproblems and Chapter 4 contains information about basic components used in this work. Chapter 5 describes the concept of the proposed active object search system, while Chapter 6 focuses on the Implementation of the concept. In Chapter 7 the results of the evaluation are presented. Finally, Chapter 8 contains a summary of the thesis, a discussion of the results and possible future work on this topic.

2 Problem Formulation

This thesis describes an autonomous robot executing an active object search task in an intelligent way. Active object search is the problem of deriving an efficient policy to find an object in a large scale, unknown, 3D indoor environment ψ, with $\psi \subseteq \mathbb{R}^3$.

In this thesis an object is defined as $O := \langle x, y, z, o \rangle$, where the vector $[x, y, z]$ is the position of the object and o is the object class or object type with $o \in C_O$ and C_O being the set of detectable object classes.

An indoor environment can be segmented into a set of rooms $R := \{r_1, ..., r_K\}$, with K being the number of rooms and a room $r_i \subseteq \mathbb{R}^3$. Further holds $\psi = r_1 \cup r_2 \cup ... \cup r_K$ and $r_k \cap r_l = \emptyset \; \forall k \in K, l \in K, k \neq l$.

We define a grid map, which discretizes the world into equally sized cells, with $grid\_map := \{cell_c\}$. Further, for every object class $o \in C_O$ a three dimensional grid map P^o_ψ is introduced. Each cell in these maps contains the probability P^o_c of an object of type o being in the cell c. In contrast to an uninformed object search, where P^o_c is constant, these probabilities are not equal for all cells, enabling the possibility to find a more intelligent search policy.

Further, we introduce a three dimensional grid maps P^R_ψ, where each cell contains a probability distribution over the detectable room types P^R_c. Further, a ground plane is assumed and cells over the same area of the ground plane are assumed to have the same probability distribution. This grid map is used to estimate P^o_c, as P^o_c depends on P^R_c.

A search task comprises searching for an object of a given target object class $o \in C_O$ in the current indoor environment until it is found or the whole environment was searched. In this thesis no prior knowledge about ψ, like

occupied space or likely object locations, is available at the beginning of the search.

Assuming a ground plane the robot can execute two dimensional motion actions to goal pose $g := \langle x, y, \theta \rangle$ in the motion space ϕ, with $\phi \in \mathbb{R}^2$, x and y specifying the position and the angle θ specifying the orientation. The motion space ϕ is a projection of the search space ψ onto the xy-plane. The robot performs sensing actions periodically, where S is the sequence of all acquired sensor information. The system should execute the search task on the path $\mathbf{p}^* := \arg\min_{\mathbf{p_i}}(cost(\mathbf{p_i}))$ with $\mathbf{p_i}$ being the i^{th} possible path $\mathbf{p_i} := (g_i^1, g_i^2, ..., g_i^{end})$ and $cost(\mathbf{p_i})$ being the cost of the path $\mathbf{p_i}$. The cost function used in this thesis is the expected search time $E[t|\mathbf{p_i}] = \int_0^\infty t \cdot f_{o_i}(t|\mathbf{p_i})dt$ with $f_{o_i}(t)$ being the probability density function of stopping the search of object o_i at time t.

The probability density function $f_{o_i}(t|\mathbf{p_i})$ depends on the path $\mathbf{p_i}$ and the object probability map P_ψ^o. P_ψ^o has to be estimated based on the acquired sensor information S and commonsense knowledge K. The sensor information is defined as a set of laser scans and RGBD-images $S = \{I_{1:t}, l_{1:t}\}$. A laser scan consists of a set of range measurements $l_i = \{l_i^1, ..., l_i^N\}$ and an RGBD-image is defined as a matrix of pixels $I_j[x, y] = p_{x,y}$, where every pixel is a tuple of values for red, green, blue and depth $p_{x,y} = \langle r, g, b, d \rangle$. The commonsense knowledge K is a $|C_R|x|C_O|$ matrix, where $|C_R|$ is the number of detectable room types. A matrix element $K_{r,o}$ contains the probability $P(o|r)$ of an object of type o being in a room of type r.

3 Related Research

In this chapter some relevant literature related to the topic of active object search and also to the subproblems encountered in this thesis are presented. The first section covers what commonsense knowledge can be used in visual active object search. The second section focuses on the detection of objects, while the third section discusses semantic maps and their creation. The last section covers relevant literature about exploration and search planning.

3.1 Commonsense Knowledge Used in Active Object Search

In an active object search task the goal is to find an object as fast as possible by an active movement of the used sensors. Without further knowledge this results in a coverage search problem, as tackled for example by Dornhege et al. [2]. In this work the two subproblems of an active object search, the set cover problem [3] and the the traveling salesman problem [4], are solved separately. In a first step high-utility view poses covering the search region are generated by calculating the minimal partition of sample view poses covering the search region. The sequence of the high-utility view poses minimizing the travel cost is then calculated by solving the traveling salesman problem, which is feasible with the reduced number of view poses. Additional knowledge about the environment and the searched object can make the search much faster because it allows to reason where an object is more likely. One type of useful knowledge is the knowledge about object-object relations, which is for example used by Kollar and Roy [5]. For example, a remote is more likely near a sofa than near a sink. Kunze et al. [6] go beyond that and use stricter relation between objects, like a

keyboard is usually in front of a monitor. Also useful is knowledge about types of places and the correlation between place type and the objects in this place, which is for example used by Viswanathan et al. [7]. For example, a knife is more likely in a kitchen than in a bedroom. Important work on the topic of active object search was done in the CogX project by Aydemir, Sjöö and other coworkers [8][1][9]. In their work they presented a system which combines knowledge about object-object relations and place-object relations with reasoning about unexplored space. Based on the type of a room assumptions about neighboring rooms are made in this work, like a corridor has usually multiple neighboring rooms while an office has probably only one.

3.2 Object Detection

The detection of an object is an essential part of an active object search system. In this thesis the task is to find an object of a specified type instead of an object instance. Therefore a general object detector is needed. Visual object detection using convolutional neural networks (CNNs) [10] is the most promising method for general object detection at the moment [11]. There the detections are typically in the form of bounding boxes and large datasets exists for the training of object detection CNNs, like PASCAL VOC [12] and COCO [13]. One type of architecture used for visual object detection are region-based convolutional neural networks (R-CNNs) [11]. R-CNNs generate in a first step region proposals, which are then classified. However, R-CNNs are slow and even their faster versions Fast R-CNN [14] and Faster R-CNN [15] are computational expensive. There are also visual object detectors using only a single CNN, like SSD [16] and YOLO [17][18][19]. YOLO divides the image into a coarse grid. For each grid cell the probabilities of the detectable object types are predicted. Furthermore, for each cell multiple bounding boxes are predicted, where each bounding box has also a confidence score. The newest version of YOLO, YOLOv3, predicts those bounding boxes at three different scales. This is all done in a single forward pass of the CNN. The confidence of the bounding boxes are then combined with the object probabilities of the corresponding cells to get the final result.

If a point cloud was created together with the image, which is the case if an RGBD-camera was used, the location of the object in 3D space can be estimated. The point cloud can also be used to segment the bounding boxes into object and background. Schwarz et al. [20] used this approach for an exact detection of objects for grasping and Sünderhauf et al. [21] used this to insert objects into semantic maps.

3.3 Semantic Maps

The creation of a map and the localization within this map is an essential skill for a robot working in an unknown environment. In the literature this problem is called simultaneous localization and mapping (SLAM). One approach to solve SLAM is the use of a particle filter. A good example for this approach is GMapping [22]. GMapping implements a Rao-Blackwellized particle filter [23], where every particle has its own map and trajectory. Using a proposal distribution which is based on the odometry data and the incoming range measurements, the particles of the next generation are drawn. For every particle an importance weight is calculated and unlikely particles get replaced in the following resampling step, which is necessary due to the limited number of particles. As a particle itself has no uncertainty in its trajectory, the maps of the particles can be created using mapping with known pose [24]. Another method for SLAM is using a graph representation. Each node in the graph is a robot pose at a certain time, carrying the sensor information. The edges connect the nodes using some measurement observations like odometry. A famous framework for graph-based SLAM is g2o [25]. Another important work on SLAM is FastSLAM [26][27]. The representation of the trajectories is similar to GMapping as also a Rao-Blackwellized particle filter is used. In contrast to GMapping, FastSLAM works with landmarks, where each particle has a Kalman filter for each landmark to estimate the locations of the landmarks.

For intelligent active object search the map has to contain more information than just occupancy information for navigation. Segmentation of the environment into rooms is beneficial as it allows to use the topological structure of the environment. Kleiner et al. [28] achieve this by applying

watershed segmentation on the 2D grid map. Friedman et al. [29] create a Voronoi graph from the 2D grid map and the points on the Voronoi graph are inserted as nodes into a conditional random field. Using this technique they are not only able to segment the environment into rooms but can also label areas into hallway, doorway or room. Aydemir et al. [1] create nodes at equal intervals, where each node corresponds to a discrete place in the environment. Together with edges representing direct paths between the nodes an undirected graph is formed. By detecting doors these nodes are grouped into rooms. These nodes also contain shape, size and appearance properties of the surrounding area, which are used for room type classification. Also objects found in a room are attached to the grouped nodes of a room. For active object search it is beneficial to have a more detailed representation for object locations, which improves the generation of the best next view pose. Wada et al. [30] use a 3D grid map containing object probabilities to represent object locations.

3.4 Exploration and Search Planning

A widely used approach for exploration proposed by Yamauchi [31] is to drive to frontiers, which are areas on the boundary between accessible space and unexplored space. The planning of the search is much more complicated. In a first step reasoning about likely object locations has to be done. Aydemir et al. [1] use a chain graph model [32]. Kollar and Roy [5] explored two models, a Markov random field model and a Naive-Bayes model. The Naive-Bayes model simplifies the problem by making "naive" independence assumptions. Aydemir et al. [1] formulated the problem of finding the optimal search path as a partially observable Markov decision process (POMDP). In large environments the state space is far too large for the problem to be solved by a POMDP-planner. Therefore, they decided to use a hierarchical approach with a classical planner for the selection of the next room and a POMDP planner for the search within the room. The classical planner makes an *all outcome determinization*, which creates a distinct deterministic instance for every probabilistic outcome [33], to make the problem, which contains uncertainty, solvable. Using an objective function which combines the cost to execute the plan and a reward dependent on

the probability of finding the object, the next room is selected. Kollar and Roy [5] formulated the problem as a minimization of the expected travel distance. The expected travel distance is calculated using the likelihood of finding the object at a certain location. The best view pose is selected using a breadth-first search to a specified horizon.

4 Prerequisites

This chapter gives an overview of the basic components used in this work. It contains information about the Robot Operating System (ROS), a robotics framework used in this work, the `move_base` package used for navigation, and the GMapping library used for simultaneous localization and mapping (SLAM).

4.1 ROS Robot Operating System

The Robot Operating System (ROS) [34] is an open-source robotics framework. It provides a structured communication layer for interprocess communication. As TCP-IP sockets are used, the processes can also run on different machines, so outsourcing of computational heavy tasks to off-board computers and communication between multiple robots can easily be done with ROS. ROS also includes the `tf` transformation system. The `tf` system collects coordinate transformations and constructs a dynamic transformation tree. Using this transformation tree the `tf` system can provide transformations between all coordinate frames in the transformation tree. It is also able to handle changing transformations, like transformations between joints of a robot arm, and can interpolate between transformations at different times. Furthermore, ROS includes build tools and tools for package managing. This makes using third party packages, contributed by members of a big community, easy. These packages range from ROS-wrappers for device drivers and software libraries to complete implementations of algorithms. ROS supports multiple programming languages, the most important are C++ and Python.

A robotics system using ROS is typically split into multiple processes, called nodes, where each node executes an individual task. Different ways for the communication between the nodes are available in ROS. The most important ones are messages, which are predefined data-structures which can hold information described through the interface definition language (IDL). The messages can be published on and received from topics, which are named data channels. The topics are managed by a special node, the ROS master, which also keeps track of all the running nodes. As ROS uses peer-to-peer communication, the ROS master is only responsible for setting up connections between the nodes, but not for the data transmission. Before a node can publish on a topic, the topic has to be advertised and before a message of a certain topic can be received the node has to subscribe to the topic. A topic can be advertised by multiple nodes and also multiple nodes can subscribe to a topic. The most important property of messages is that the sender of a message has no knowledge about the receiving nodes and if the messages are processed by the receivers.

A second way of communication between nodes are services. Services are used for synchronous communication, where a request is sent to another node and a response is sent back after the request is processed. The structure of the request and the response are also described through the IDL. The `actionlib` package enables a third way of communication. Using messages the functionality of services is implemented, with the additional possibility for feedback during the execution. This is especially useful during long-lasting tasks.

4.2 `move_base`

The `move_base` package [35] is the most important part of the ROS navigation stack and is used for path planning, both for local planning and global planning, and their execution. The `move_base` package receives goal poses as input and calculates velocity commands for the robot to reach the goal. In Figure 4.1 an overview over the `move_base` package architecture is given.

The modular architecture is beneficial as each component can be chosen separately to fit the robot and the task. The planning problem is split into

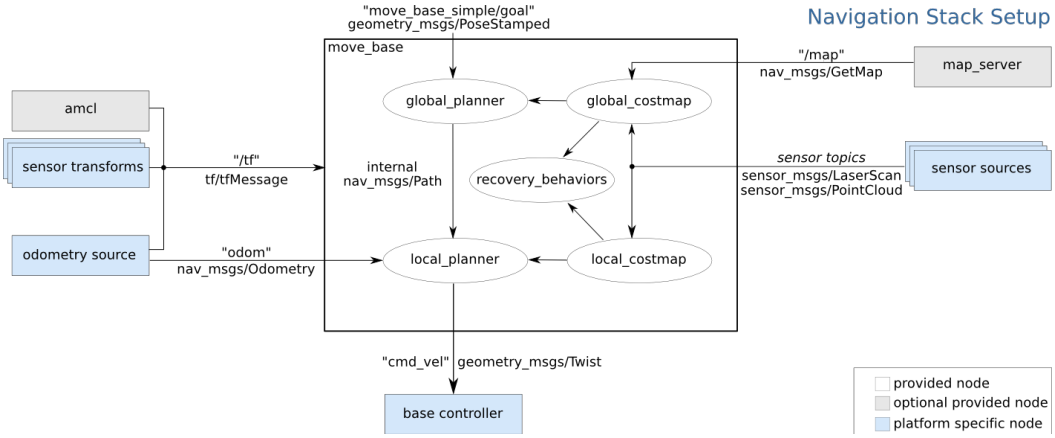


Figure 4.1: `move_base` architecture (source: `http://wiki.ros.org/move_base`)

two parts, the global planning and the local planning. The `global_planner` receives the goal pose and plans a path from the current robot location to the goal. For the generation of valid paths a costmap of the environment is necessary. A costmap is a grid map, where each cell contains information about how much the planner should avoid the cell. Cells containing obstacles have very high costs and the cost decreases with increasing distance to an obstacle. The costmap used by the `global_planner` is usually based on a map of the environment and additional information from sensors. The `local_planner` receives the path generated by the `global_planner` and generates velocity commands to follow this path. To avoid collisions the `local_planner` has a costmap of the local neighborhood which is generated from incoming sensor data. If either the `global_planner` fails to generate a valid path or the `local_planner` fails to generate valid velocity commands, recovery behaviors are executed. These involve the clearance of the costmaps and, if possible, in-place rotations. If the recovery behaviors do not solve the problem, the navigation is aborted.

4.3 GMapping

GMapping [22] is an open-source library implementing two dimensional simultaneous localization and mapping (SLAM). The sensor data used by

GMapping are range measurements from a light detection and ranging sensor (LIDAR) and odometry data.

SLAM is a difficult problem because for the creation of a map the pose of the robot is necessary and for accurate localization a map is necessary. GMapping implements a Rao-Blackwellized particle filter [23] to solve this chicken-and-egg problem. The SLAM problem is split into two parts, the estimation of the trajectory and the mapping given the trajectory. The probability distribution over the trajectories is represented by a particle filter. Each particle in the particle filter has a weight and its own map which is created by occupancy grid mapping with known pose.

The particle filter algorithm consists of three steps, sampling, importance weighting and resampling. In the sampling step the poses of the particles are updated by a movement step sampled from a proposal distribution. This proposal distribution is based on the odometry data and in more sophisticated systems further sensor data. In the importance weighting step a weight is assigned to each particle representing the likelihood that the particle represents the true trajectory. In the resampling step particles with low weight are replaced by particles with a higher weight. This step is needed as a continuous distribution is approximated by only a finite number of particles. After the resampling the maps of the particles are updated using the particle pose and the range measurements from the LIDAR.

The problem of this approach is the high computational cost when using a large number of particles, which is usually needed for a good representation of the distribution over the trajectories. Therefore, GMapping also takes the range measurements into account in the sampling step. This way unlikely particle poses can be ruled out immediately and a much lower particle number is sufficient. GMapping also uses a sophisticated resampling technique which only does resampling if necessary. This reduces the probability of discarding good particles, which is especially important for low particle numbers.

5 Concept

5.1 Overview

A number of fundamental design decision had to be made in the early stages of the development of this active object search system. The main questions were:

- what commonsense knowledge can be used to execute the object search intelligently
- what information the robot should gather and what sensors are needed for that
- how the gathered knowledge about the environment should be represented
- how the search planning, using this knowledge, should be done

Indoor environments can be segmented into rooms and room areas which serve specific purposes, e.g. a kitchen is used for cooking. Therefore, the objects located in an area correlate with the type of the area. This connection is used in this work to make the search intelligent and therefore more efficient. Furthermore, knowledge to recognize objects of the searched type and to classify areas into room types is needed. Neural networks, trained on thousands of images, are used, which represents this knowledge in their learned weights.

An RGBD-camera is used to gather the input images for the neural networks. The additional depth information is helpful to estimate the positions of the detected objects in space. This would be difficult otherwise. The depth measurements are also necessary for 3D obstacle detection and mapping. The camera is mounted on the robot on a fixed location to keep the setup simple. Therefore, the camera pose can only be change by the movement of

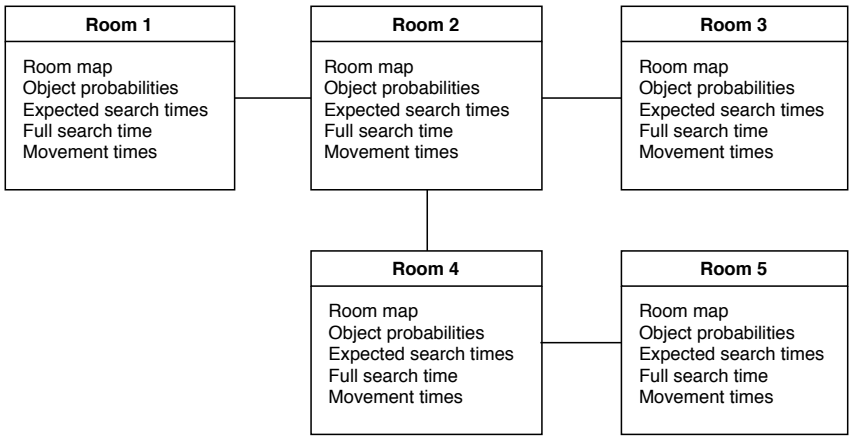


Figure 5.1: Example of a topological map with 5 rooms and 4 doorways

the robot in the plane. As the depth data from the RGBD-camera is not very accurate, a light detection and ranging sensor (LIDAR) is also mounted on the robot. Together with the odometry data from the robot it is used for 2D simultaneous localization and mapping (SLAM). Finally, the robot is equipped with a second RGBD-camera which is used for the detection of doorways. The robot setup is depicted in Figure 5.4.

A hybrid map is used to represent the environment. In this hybrid map each room has its own semantic metric map. These semantic maps contain, in addition to the standard occupancy information for navigation, information about object locations and the room type of areas within the room. Furthermore, locations of detected doorways are inserted into the map. The information in the semantic map and commonsense knowledge are combined to estimate likely object locations within the room which is then used to find objects faster. On the other hand a topological map is created with a node for every discovered room. These nodes contain references to the corresponding semantic room maps and also additional data generated from those maps. This additional data consists of the probability of a searched object being in a room, an estimation of the time needed to fully search a room, the expected time spent searching in a room, and an estimation for the time needed to leave or traverse a room. The nodes are linked by edges which correspond to the doorways connecting two rooms. Figure 5.1 shows an example of such a topological map.

The objective of the search planning is to minimize the expected search time. The complexity of the problem makes finding the optimal next view pose infeasible since it contains the traveling salesman problem [4] and the set cover problem [3] [2]. Therefore, a heuristic is needed. In this thesis the planning problem is split into two abstraction levels. The high-level planner selects the optimal next room to search based on the topological map and also generates high-level tasks, like move through a doorway, explore a room or search a room. Based on the pending high-level tasks the low-level planner generates the best next goal pose for the robot using a greedy strategy. Figure 5.2 shows a flowchart depicting the general planning procedure. This planner design fits perfectly to the hybrid map as the high-level planner does its planning only on the topological map, while the low-level planner operates only on the metric map of the current room.

Figure 5.3 gives an overview over the structure of this intelligent active object search system, showing the main components of the system and also the most important data exchanged by these components.

5.2 Hardware

The robot used in this master's thesis is a modified version of the Turtlebot 2 and is shown in Figure 5.4. The Yujin Kobuki base of the Turtlebot is a differential drive base with an odometry amended by a gyroscope. The maximum translational velocity of this base is 0.7 m/s and the maximum rotational velocity is 180 °/s, though the full speed is not used. The maximum payload is 5 kg, which is about the weight of the notebook, the sensors and the construction.

For accurate localization and 2D mapping a Hokuyo URG-04LX LIDAR is mounted on the base at a height of about 15 cm. The specified range is from 0.02 m to 4 m with an accuracy of ± 10 mm and the angular range is from -120° to 120° with a resolution of 0.36°. The LIDAR scans at a rate of 10 Hz.

The main camera for object detection, place classification and 3D mapping is an ASUS Xtion PRO LIVE RGBD-camera mounted at a height of about

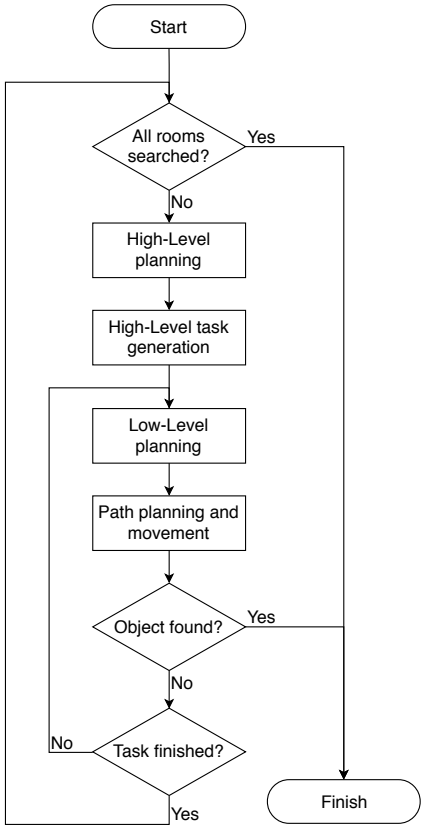


Figure 5.2: Flowchart of the planning procedure

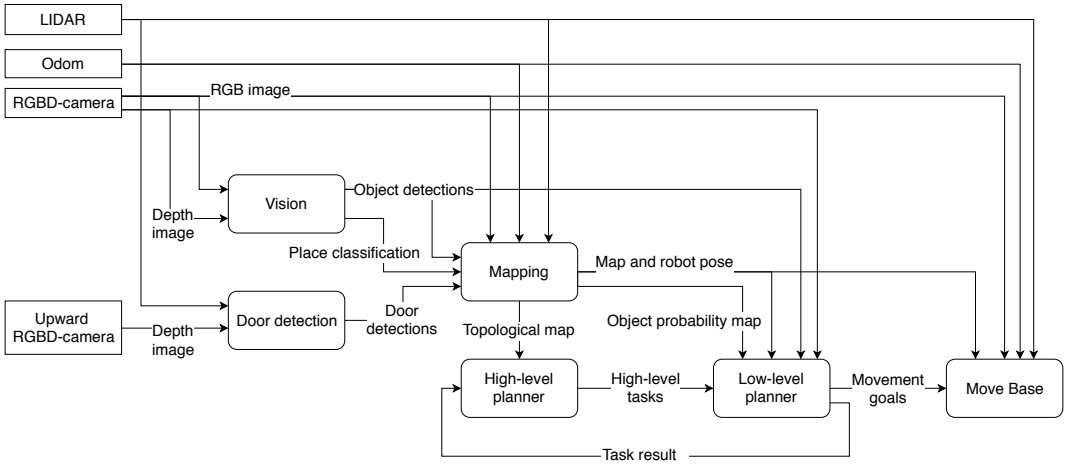


Figure 5.3: Overview over the full system structure

1m facing slightly downwards. The camera takes color images and depth images at a rate of 30 Hz with a resolution of 640x480. The field of view is 58° horizontally and 45° vertically. With the orientation of the camera on the robot the visible area is from -29° to 29° horizontally and from -30° to 15° vertically. The range of the depth measurements is specified with 0.8 m to 3.5 m, though experiments showed a minimum range of about 0.55 m. A second ASUS Xtion PRO LIVE camera, used for doorway detection, is mounted looking upwards.

An ASUS notebook is mounted on the robot running the complete software. It is equipped with a 7^{th} generation Intel i7 quadcore CPU working at 2.8 GHz, 16 GB DDR4 RAM and an NVidia Geforce GTX 1050 Ti Mobile GPU with 4 GB memory. Those components are up to date and quiet powerful for a notebook. However, high-end hardware, especially state-of-the-art desktop GPUs, are much more powerful. The lack in computational power was needed to be taken into account in the design of this active object search system.

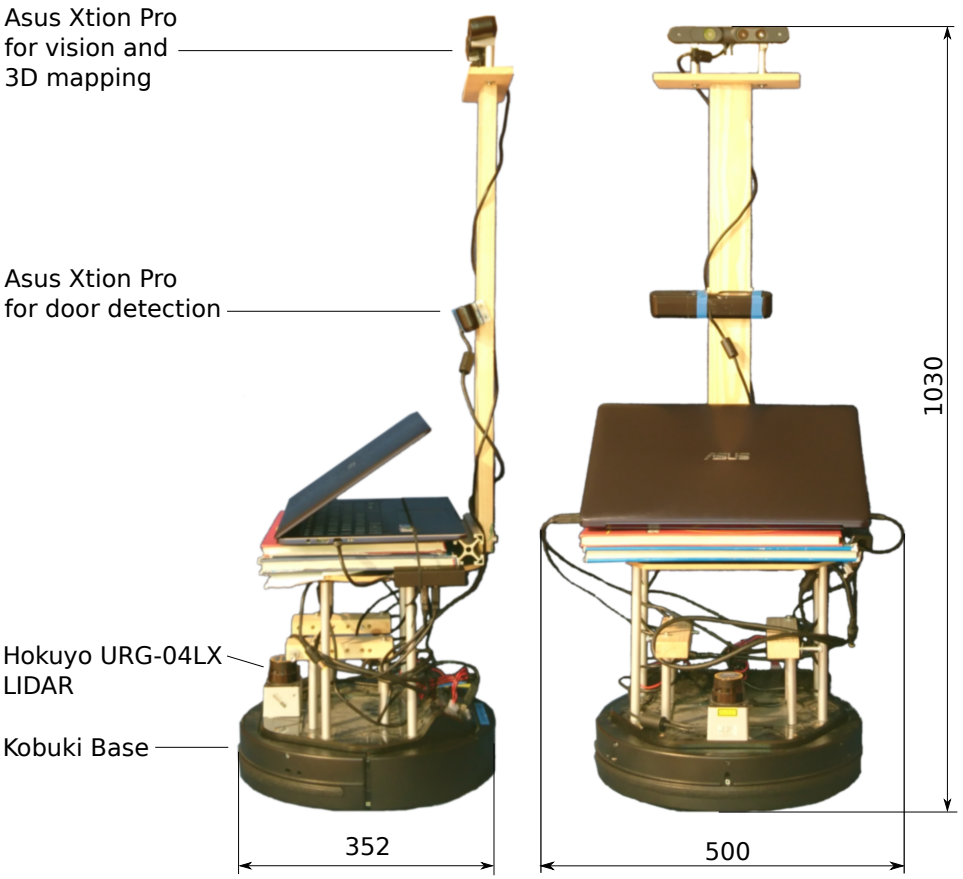


Figure 5.4: Robot setup in front and side view with dimensions and labels of most important parts

5.3 Commonsense Knowledge

An inevitable prerequisite for intelligent behavior is knowledge. A learning system is out of the scope of this thesis because of the huge amount of time it takes a robot to learn enough about indoor environments to make reasonable decisions. Therefore, the knowledge useful for the robot has to be chosen and prepared. In this work commonsense knowledge about indoor environments, rooms, and objects is important. Some knowledge, like the concept of a room, is implicitly used through the design of the system. Other knowledge, like the visual appearance of objects and room types is encoded in the weights of the CNNs which are trained on thousands of labeled images. Many already trained CNNs are available and ready to use.

Additionally the robot is equipped with knowledge about the correlation between the type of a room area and the objects located in this area. One application of this knowledge is the estimation of object probabilities in unexplored areas of a room. Based on the room types in already explored areas of a room the room type of the unexplored areas can be estimated and in further consequence also the object probabilities. Therefore, it is possible to make a reasonable estimate about the probability of the searched object being in a room after just peeking into the room. The second application is to find likely object locations based on the room type of an area. The room type of areas in the room can be determined quickly but many objects are only detected when taking a closer look. Knowing in what type of area the object we are searching for is more likely helps to find the object faster.

Only correlations between the detectable object classes and detectable room types are important. These are the classes the neural networks are trained for. With N_R room types and N_O object classes this results in a N_RxN_O matrix KB, where the element $KB_{r,o}$ represents the probability of an object of type o being in a room of type r.

The idea is to use the information from image datasets created for computer vision challenges. The dataset the object detector was trained on consists of thousands of images with annotations specifying the objects visible in the images. To get also the place categories of the images the place classification CNN proposed by Wang et al. [36] was run on all images of the dataset and

the probability distribution over all place classes was stored for every image. With this information the matrix elements were calculated with

$$KB_{r,o} = \frac{\sum_{i=1}^{|Images|} P_i(r)P_i(o)}{\sum_{i=1}^{|Images|} P_i(r)} \tag{5.1}$$

where $P_i(r)$ is the probability of the image i showing a place r, which is the result of the place classification CNN, and

$$P_i(o) = \begin{cases} 1 & if\ object\ in\ image \\ 0 & else \end{cases} \tag{5.2}$$

This approach is more promising than other approaches, like the estimation based on the number of search results in image search engines proposed by Hanheide et al. [37]. One reason is that the images in computer vision challenge datasets are intended to be representative for the world. In contrast some uncommon scenes are highly overrepresented on the Internet, like the duck in the bath tube. A second benefit is that the names of objects and room types are not used in the proposed method, but only the concepts learned by the CNNs. This avoids problems with ambiguous terms like mouse, which can describe an animal or an electronic device, and not clearly defined terms like most room types.

With this approach the meaning of the matrix KB is slightly different. The matrix element $KB_{r,o}$ is now the probability of an object of type o being in an image taken in a room of type r. So the size of the space usually seen in an image in the dataset has to be taken into consideration, as images covering more space are more likely to contain an object. On the other hand the size of the region of interest has also to be considered because smaller areas have lower object probabilities. This is described in more detail in Section 5.5.5.

5.4 Visual Data Extraction

This section describes the extraction of the information from the input images needed in later stages of the system. It covers the object detection,

the room type classification, and the doorway detection.

5.4.1 Object Detection

The object detection uses the images taken from the main RGBD-camera to find objects visible in the images and also estimates their positions in space. The most common way at the moment is to use a visual object detection. Visual features of objects are very descriptive, a lot of research was already done in this field and huge datasets containing labeled images are available for the training of the proposed object detectors. This resulted in the decision to use an object detector working on 2D images and then projecting the object detections into 3D space using the depth image.

Convolutional Neural Networks (CNNs) are the best performing systems for visual object detection at the moment. The quality of the dataset used to train a CNN is very important for the performance of a CNN and the object types the CNN can detect are dependent on the objects labeled in the dataset. In this thesis a CNN trained on the COCO dataset is used because it provides a sufficient number of object types without being too specific. In Table 5.1 a list of the object types in the COCO-dataset is shown. The dataset also contains enough images, so there is enough variation in the objects to achieve a good generality. The drawback is that objects are not labeled per pixel, but only bounding boxes are given. Therefore, the detected objects are also only represented by bounding boxes, which is neither an accurate representation nor ideal for further processing. As no suitable dataset with objects labeled per pixel is available, the object detection and the later stages of the system have to cope with this problem.

The result of such an object detection CNN is a set of bounding boxes. Each bounding box, specified by location in the image and size, has also probability values for each detectable object class attached. In Figure 5.5 a typical result of the object detection is shown. In this image only the most likely detections are displayed. As one can see the result is far from perfect, even with state-of-the-art object detection CNNs. Many objects are not detected or have a low confidence. Another problem visible in the image is the possibility of overlapping bounding boxes. These problems

Table 5.1: Detectable object classes grouped in categories, crossed out classes are ignored in this thesis

categories	classes
person	person
accessory	backpack, umbrella, handbag, tie, suitcase
animal	bird, cat, dog, ~~horse~~, ~~sheep~~, ~~cow~~, ~~elephant~~, ~~bear~~, ~~zebra~~, ~~giraffe~~
vehicle	bicycle, ~~car~~, ~~motorbike~~, ~~aeroplane~~, ~~bus~~, ~~train~~, ~~truck~~, ~~boat~~
outdoor objects	~~traffic light~~, ~~fire hydrant~~, ~~stop sign~~, ~~parking meter~~, ~~bench~~
sports	frisbee, skis, snowboard, sports ball, kite, baseball bat, baseball glove, skateboard, surfboard, tennis racket
kitchenware	bottle, wine glass, cup, fork, knife, spoon, bowl
food	banana, apple, sandwich, orange, broccoli, carrot, hot dog, pizza, donut, cake
furniture	chair, sofa, potted plant, bed, dining table, toilet
appliance	microwave, oven, toaster, sink, refrigerator
electronics	monitor, laptop, mouse, remote, keyboard, cell phone
indoor objects	book, clock, vase, scissors, teddy bear, hair drier, toothbrush

have to be taken into account in the design of the object mapping system, as the accumulation of information from multiple images is needed for more reliable results.

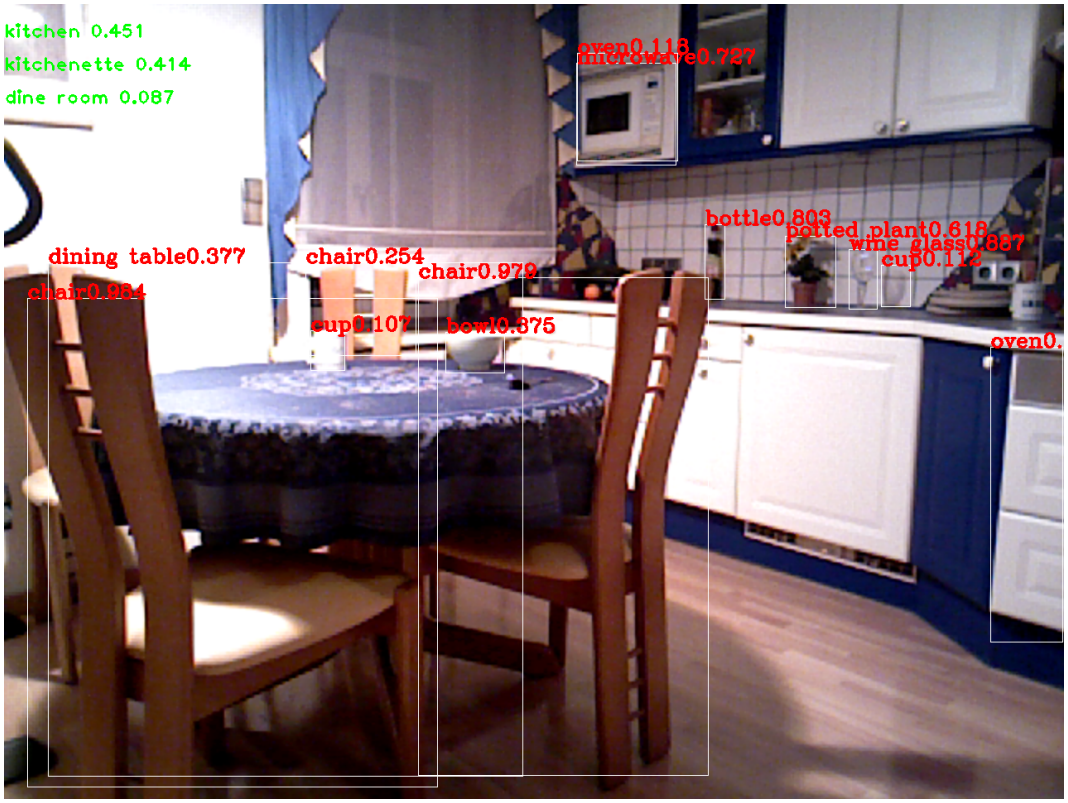
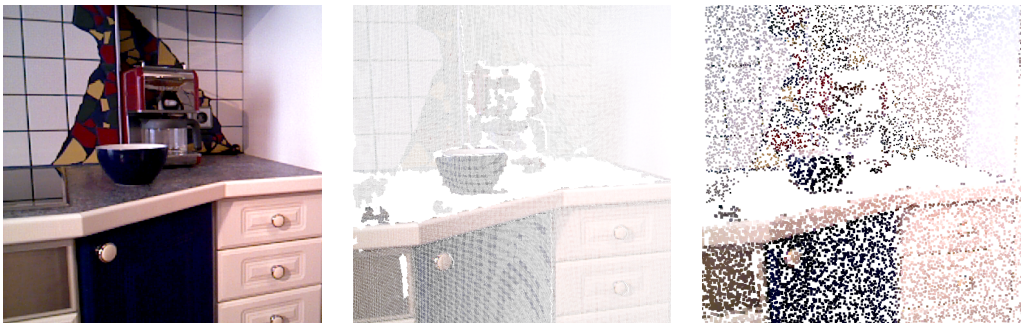


Figure 5.5: A typical result of the object detection and the room type classification; detections with a probability greater than 0.1 are shown with bounding box, most likely class and its probability; in the upper left corner of the image the top classifications of the room type classifier and their probabilities are shown

Alongside with the color image the RGBD-camera takes also an depth image which contains depth measurements for every pixel. Due to technical limitations the depth of some pixels cannot be measured. Combining the position of the pixel, the depth value and the intrinsic camera parameters the position of the pixel in space can be calculated. Projecting a bounding box into 3D space on the other hand is not meaningful as neighboring pixels in the image are not necessarily nearby in 3D. The approximate location of



(a) RGB-image

(b) point cloud containing all pixels with valid depth values

(c) sampled point cloud

Figure 5.6: Example image with corresponding point cloud and sampled point cloud; the points in the sampled could are depicted larger for better visibility

a detected object can be estimated using the median depth in the bounding box. However, the extend of an object is difficult to estimate. Therefore, a sample-based approach is used. A sufficient number of pixels with valid depth values is drawn randomly from the image. This is illustrated in Figure 5.6. Each of those samples holds an object probability for every object class. The problem is that bounding boxes also contain pixels not belonging to the object and pixels can be within multiple bounding boxes. A per-pixel segmentation of the detected objects is an option to solve this problem, but the segmentation is costly for a large number of detections and wrong segmentations are possible, especially with small or complex objects, overlapping bounding boxes and occlusion. Therefore, following a simple approach, the maximum object probability of all bounding boxes overlapping a pixel is used. The idea behind this is the assumption that all bounding boxes with lower probabilities are just accidentally overlapping. The unreliability of this approach is then taken into account in the inverse sensor model of the object mapping.

Ultimately, the result of the object detection is a set of 3D points, each one holding probabilities for every detectable object type. Furthermore, the result contains the object type and the estimated 3D location of very confident object detections.

Table 5.2: indoor room categories

art gallery	cafeteria	game room	music studio
art studio	candy store	gift shop	nursery
assembly line	classroom	home office	office
attic	closet	hospital room	pantry
auditorium	clothe store	hotel room	parlor
bakery	coffee shop	ice cream parlor	reception
ballroom	conference center	jail cell	restaurant
banquette hall	conference room	kindergarden	restaurant kitchen
bar	corridor	kitchen	shoe shop
basement	dinette	kitchenette	shower
beauty salon	dining room	laundromat	staircase
bedroom	dorm room	livingroom	supermarket
bookstore	engine room	lobby	television studio
bowling alley	food court	locker room	veranda
butcher shop	galley	martial arts gym	waiting room

5.4.2 Room Type Classification

In this thesis room type classification is done base on images. Visual features are very expressive for room type classification and also image datasets containing lots of labeled images exist. One of those is the MIT Places-205 dataset. It contains about 2.5 million images assigned to 205 place categories and CNNs trained on this dataset are publicly available. 60 of those categories are relevant as the other categories are outdoor place categories. These indoor place categories, which are room types and types of areas in rooms, are listed in Table 5.2. The result of a CNN trained on the Places-205 dataset is a probability distribution over the 205 place categories. The irrelevant outdoor categories are then ignored by setting their probabilities to zero. The remaining probabilities are normalized to 1.0, resulting in a probability distribution over the 60 indoor categories for the room type seen in the input image.

5.4.3 Doorway Detection

The detection of doorways is important for the segmentation of the environment into rooms. In this thesis doorways are detected based on the typical height and shape of the top part of the door frame. Doorways are usually 2 meters high and at this height few objects can be found. Therefore, the detection is very reliable using the depth images of an upward looking RGBD-camera. A passable doorway is detected if

- there is a rectangular surface at about 2 meters height
- this surface has a typical width and depth for a door frame
- the area beside the door is occupied, as there has to be a wall
- the area in front of and behind the door is free of obstacles, so the robot can drive through

Using the depth image of the upward looking RGBD-camera, an image of the projections onto the ground plane of surfaces at about 2 meters height is created. This image is filtered and rectangles in the filtered image are extracted. Rectangles that have no typical size for a door frame are then discarded and the others are possible doorways. The range measurements from the LIDAR are then used to check if the area within, in front and behind a possible doorway is free and if the area beside the doorway is occupied. If this is the case, a doorway is found. The position of the detected doorway is set to the center of the rectangle and the orientation is set parallel to the smaller axis of the rectangle. To be unambiguous the doorway has to face in positive x-direction. This results in detected doorways always facing out of the room. In Figure 5.7 this procedure is illustrated.

5.5 Mapping

This section describes the mapping concept used in this thesis. The mapping fuses the data gathered by the sensors and the results of the visual data extraction into a semantic map. In addition to the information about drivable space and obstacles needed for navigation, the semantic map also contains information about rooms, room types, and objects to enable an intelligent search planning.

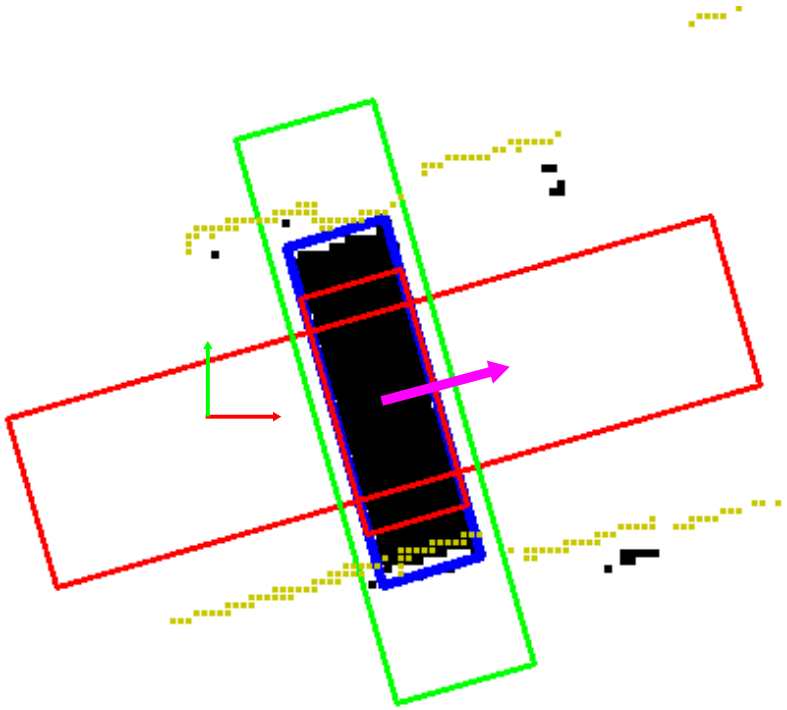


Figure 5.7: A successfully detected doorway marked by the purple arrow in the coordinate frame of the robot; black dots are the projections onto the ground plane of points at about 2 meters height detected by the RGBD-camera; the yellow points are range measurements from the LIDAR; the detected door frame is marked with the blue rectangle; the red rectangles are the area within the door frame and the area on both sides of the doorway and the green rectangle contains the area beside the doorway; those rectangles depend on the size of the doorway and size parameters, which are described in more detail in Section 6.3.

The first main idea in the mapping design used in this work is to use a hybrid map. The chosen hybrid map design combines two paradigms, grid-based and topological. Grid-based maps are accurate metric maps optimized for navigation. However, with increasing size grid-based maps are difficult to be kept consistent and the efficiency goes down, especially for movement planning. Topological maps on the other hand are more difficult to use for local navigation, but scale very well. A hybrid map can combine the benefits of both map types. Indoor environments are very suitable for such a hybrid map. On a higher level an indoor environment can be represented by a topology of rooms and every room can then be described separately in more detail using a grid-based map. Therefore, this approach was used in this thesis. Every discovered room is represented by its own grid-based metric map, not only containing occupancy information but also information about the types of areas in the room and object locations. Each room is also a node in the high-level topological map, capturing the room structure of the environment. Another benefit of this design is the possibility to split the search problem into two abstraction levels, which will be described later in this chapter.

The second main idea is to use a 2D simultaneous localization and mapping (SLAM) system based on a Rao-Blackwellized particle filter [23]. SLAM is a difficult problem as the quality of the map depends on the quality of the localization and vice versa. This leads to uncertainty in the map, the current pose, and also the path taken during mapping. This uncertainty has to be taken into account when information is added to the map. The easiest way to do this is probably using a SLAM systems based on a Rao-Blackwellized particle filter. In this approach the uncertainty about the path is represented by particles and their weights. The particle itself represents no uncertainty. For every particle a map is created with known pose. In 2D SLAM this is only a 2D occupancy map, but further information can also be mapped the same way and because the mapping is done with known pose, this additional information is consistent with the 2D map. The drawback of this approach is that every particle needs its own set of maps, making it only feasible for low particle numbers. To overcome this drawback sophisticated sampling techniques, like the one used in GMapping [22], can be used, which allow to create high-quality maps even with low particle numbers. 2D SLAM is preferable to 3D SLAM as tests using 3D SLAM systems showed

mediocre results, probably because of the inaccuracy and unreliability of the depth measurements from the RGBD-camera.

The map representation consists of a topological map with a node for every room and an edge for every doorway connecting two rooms. Each room has a set of maps, called semantic room map, where each map contains different information. These are:

- Two 2D occupancy grid maps: One is created by the SLAM system based on the range measurements of the LIDAR and used for localization. The second one is based on the first 2D map, but also represents the projections onto the ground of the obstacles detected by 3D mapping and is used for navigation, exploration and in-room search planning.
- A 3D occupancy grid map: This map is created using the RGBD-camera. It is used in the creation of the 2D occupancy grid map for navigation and also in the calculation of likely object locations.
- A Room-Type-Map: This is a 2D grid map, where every grid cell contains a probability distribution over all detectable room types. It is created using the results of the room type classification.
- An Object-Map for every detectable object class: These are 61 3D grid maps containing the probability of an object of the corresponding type being seen in a grid cell given the cell is occupied. These maps are created based on the results of the object detection.
- An Object-Probability-Map: This is a 3D grid map containing the probability of the currently searched object being in a grid cell. This map is only created for the searched object type. It is created from the information in the other maps and the commonsense knowledge which is described in more detail in Section 5.5.5.

Furthermore, the semantic room map contains estimated locations of detected doorways. In further stages of the system the maps of the currently best particle are used.

In the following subsections the individual parts of the mapping system are presented in more detail. The generation of the high-level attributes of the room, like the total object probability and the expected search time, is described at the end of this section.

5.5.1 3D Mapping and Creation of the 2D Map for Navigation

3D mapping is done using the OctoMap library [38] which does 3D mapping with known pose using an Octree representation for the map. Each node in the Octree represents a cube in space and contains a probability value for being occupied. These probabilities are later used in the creation of the Object-Probability-Map. As some obstacles, like tables, are not sufficiently detectable by the LIDAR, the information in the 3D map is also necessary for safe navigation. To avoid the more complex path planning in 3D this additional information about obstacles is inserted into a second 2D map which is then used for navigation. Therefore, all nodes in the Octomap with an occupancy probability higher than a threshold and a z-coordinate smaller than the robots height are projected into the 2D map. This is illustrated in Figure 5.8.

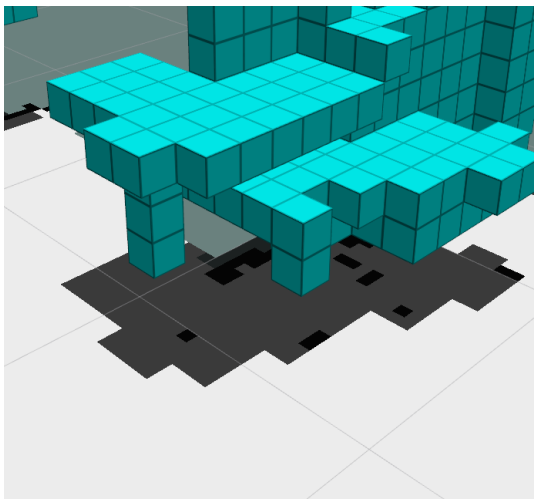


Figure 5.8: A snippet of a room showing a table and chairs, which are insufficiently detected by the LIDAR: 3D map (teal) and 2D map are shown; the light area is free, the black cells are obstacles in the 2D map and the dark gray cells are set occupied by the down-projection procedure

5.5.2 Object-Map Creation

This section covers the creation of the Object-Map which contains information about the positions of the objects seen by the robot. The object detection can detect lots of objects in an image, especially in cluttered indoor environments. Those detections are often not reliable and the location and especially the extent of detected objects is inaccurate. Moreover, an object tracking approach is not very promising. Therefore, a grid map representation containing object probabilities was chosen. This approach can handle uncertainty much better and the number of detected objects has no impact on the performance.

For every detectable object class an independent 3D grid map is created. Every cell in this grid map holds the probability of containing an object of the corresponding type given it is occupied. Mapping this conditional probability is beneficial, as the result of the object detection are sample points which lie on object surfaces and therefore in occupied cells, assuming the depth measurements are accurate. The unconditional probability can be calculated using the 3D occupancy map. With this approach the costly ray-tracing which is necessary to find the cells to be marked as free, has only to be done in the 3D mapping.

Some assumptions were made to make this approach feasible. The independence of the grid cells and a static map are assumed. Furthermore, the assumption is made that the process follows a hidden Markov model. These are assumptions frequently made for similar problems, like occupancy grid mapping [24]. Another assumption is the independence of different object types. This assumption implies that multiple objects can be located in a grid cell. This is reasonable if the resolution of the grid is not very high. This is the case in our application.

Using those assumptions and Bayes' theorem the formula for the update of

a cell is derived:

$$
\begin{aligned}
P(O_c^o|I_{1:t}, \mathbf{x}_{1:t}, occ_c) = \\
\frac{P(I_t|O_c^o, I_{1:t-1}, \mathbf{x}_{1:t}, occ_c)P(O_c^o|I_{1:t-1}, \mathbf{x}_{1:t}, occ_c)}{P(I_t|I_{1:t-1}, \mathbf{x}_{1:t}, occ_c)} = \\
\frac{P(I_t|O_c^o, \mathbf{x}_t, occ_c)P(O_c^o|I_{1:t-1}, \mathbf{x}_{1:t-1}, occ_c)}{P(I_t|I_{1:t-1}, \mathbf{x}_{1:t}, occ_c)} = \\
\frac{P(O_c^o|I_t, \mathbf{x}_t, occ_c)P(I_t|\mathbf{x}_t, occ_c)P(O_c^o|I_{1:t-1}, \mathbf{x}_{1:t-1}, occ_c)}{P(O_c^o|occ_c)P(I_t|I_{1:t-1}, \mathbf{x}_{1:t}, occ_c)} = \\
\frac{1}{Z}\frac{P(O_c^o|I_t, \mathbf{x}_t, occ_c)P(O_c^o|I_{1:t-1}, \mathbf{x}_{1:t-1}, occ_c)}{P(O_c^o|occ_c)} \qquad (5.3)
\end{aligned}
$$

$P(O_c^o|I_{1:t}, \mathbf{x}_{1:t}, occ_c)$ is the probability of cell c containing an object of type o given the RGBD-images $I_{1:t}$, the poses these images were taken at $\mathbf{x}_{1:t}$ and being occupied. $P(O_c^o|I_t, \mathbf{x}_t, occ_c)$ is the inverse sensor model, $P(O_c^o|I_{1:t-1}, \mathbf{x}_{1:t-1}, occ_c)$ is the recursive term and $P(O_c^o|occ_c)$ is the prior. Z is a normalization term, which can also be calculated by:

$$
\begin{aligned}
Z = \frac{P(O_c^o|I_t, \mathbf{x}_t, occ_c)P(O_c^o|I_{1:t-1}, \mathbf{x}_{1:t-1}, occ_c)}{P(O_c^o|occ_c)} + \\
\frac{P(\neg O_c^o|I_t, \mathbf{x}_t, occ_c)P(\neg O_c^o|I_{1:t-1}, \mathbf{x}_{1:t-1}, occ_c)}{P(\neg O_{occ,c}^{OT}|occ_c)} \qquad (5.4)
\end{aligned}
$$

as the sum of $P(O_c^o|I_{1:t}, \mathbf{x}_{1:t}, occ_c)$ and $P(\neg O_c^o|I_{1:t}, \mathbf{x}_{1:t}, occ_c)$ has to be 1.0.

In this problem the prior cannot be set to 0.5 as an occupied cell usually does not contain an object of a certain object type. Furthermore, a low prior probability is necessary for many very uncertain object detections, resulting in a high object probability, which is an intended behavior. The value of the prior had to be estimated empirically.

Finding a suitable inverse sensor model is a difficult task. It has to cope with the uncertain results of the object detection and also the possibility that other objects are within the cell has to be taken into account.

The chosen inverse sensor model is

$$P(O_c^o|I_t, \mathbf{x}_t, occ_c) = \begin{cases} \max_{s \in \mathbf{S_t^c}} V_{hit} P(O_s^o|I_t) + V_{miss}(1 - P(O_s^o|I_t)) & |\mathbf{S_t^c}| > 0 \\ P(O_c^o|occ_c) & |\mathbf{S_t^c}| = 0 \end{cases} \tag{5.5}$$

with $P(O_s^o|I_t)$ being the probability for object type o of sample s given the image I_t. $\mathbf{S_t^c}$ is the set of samples which are the result of the object detection on image I_t, falling into cell c. V_{hit} and V_{miss} are scalar parameters modeling the probabilities for false positives and false negatives.

In words, the inverse sensor model is the highest probability for an object type of all samples falling into a cell with the integration application of uncertainty. The assumption behind this approach is that the sample with the highest probability is on the object, the other samples are on other objects in the cell.

The problem of pixels in a bounding box not belonging to the object is not explicitly handled by this approach. Suitable values for the parameters V_{hit} and V_{miss} and the integration of information from images taken at different poses should cope with this problem.

The final probability of an object of type o being seen in a cell can be calculated with:

$$P(O_c^o|I_t, \mathbf{x}_t) = P(O_c^o|I_t, \mathbf{x}_t, occ_c) P(occ_c|I_t, \mathbf{x}_t) \tag{5.6}$$

$P(occ_c|I_t, \mathbf{x}_t)$ is the probability of the cell being occupied, which is stored in the 3D map. If the cell is not occupied no object can be within the cell.

5.5.3 Room-Type-Map Creation

Assigning a single room type to a whole room is not sufficient for many rooms. On the one hand it is quiet common that different types of rooms, like a kitchen and a living room, are only partially or not at all separated by walls. On the other hand more accurate classes can often be found for areas in a room, like a pantry-like area in a kitchen. Therefore, a Room-Type-Map is created for every room, which is a 2D grid map where each grid cell has its own probability distribution over the detectable room types.

The mapping procedure is similar to the creation of the Object-Map and also assumes independent grid cells. Another assumption is that a cell is entirely of one type. The formula of the probability update is:

$$P(R_c|I_{1:t}, \mathbf{x}_{1:t}) = \frac{P(R_c|I_t, \mathbf{x}_t)P(I_t|\mathbf{x}_t)P(R_c|I_{1:t-1}, \mathbf{x}_{1:t-1})}{P(R_c)P(I_t|I_{1:t-1}, \mathbf{x}_{1:t})} = \frac{1}{Z}\frac{P(R_c|I_t, \mathbf{x}_t)P(R_c|I_{1:t-1}, \mathbf{x}_{1:t-1})}{P(R_c)} \quad (5.7)$$

$P(R_c|I_{1:t}, \mathbf{x}_{1:t})$ is the probability distribution over the detectable room types of cell c given the RGBD-images $I_{1:t}$ and the poses these images were taken at $\mathbf{x}_{1:t}$. $P(R_c|I_t, \mathbf{x}_t)$ is the inverse sensor model, $P(R_c|I_{1:t-1}, \mathbf{x}_{1:t-1})$ is the recursive term and $P(R_c)$ is the prior. Presuming no prior knowledge the prior $P(R_c)$ is set to $\frac{1}{|\mathbf{R}|}$, with $|\mathbf{R}|$ being the number of detectable room types. Z is a normalization term which can also be calculated with

$$Z = \sum_{R_c} \frac{P(R_c|I_t, \mathbf{x}_t)P(R_c|I_{1:t-1}, \mathbf{x}_{1:t-1})}{P(R_c)} \quad (5.8)$$

as the probabilities have to sum to 1.0.

The inverse sensor model $P(R_c|I_t, \mathbf{x}_t)$ uses the result of the room type classification for the image I_t, $P(R_{I_t}|I_t)$. Also the visibility of the cells has to be taken into account. A simple approach is used, assuming all cells with a cell center within the view cone of the camera and within a certain range, are visible and receive the same update. This is illustrated in Figure 5.9. A more complex mode is not used for two reasons. One reason is that no reasonable estimation can be made how much a visible area contributed to the classification result. The other one is that most of the cells wrongly updated using the simple model lie behind walls. The area behind a wall is usually outside of the room and therefore ignored anyway in later stages of the system. Most other obstacles, like tables, only partially block the view, so the area behind those obstacles is assumed to be sufficiently visible.

The inverse sensor model also models the uncertainty of the classification result and the possibility of an image showing areas with different room

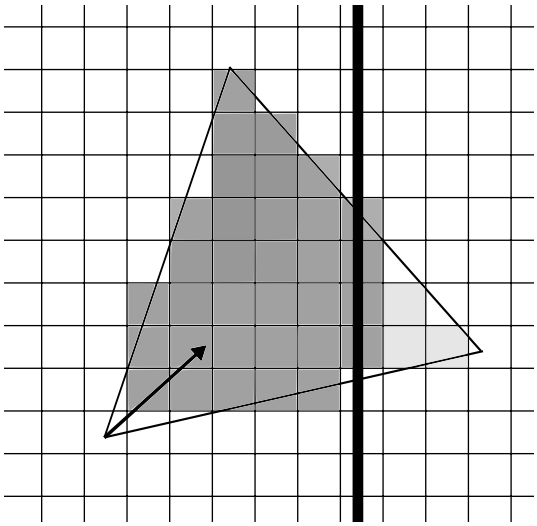


Figure 5.9: The cells updated in the Room-Type-Map using the image taken at the robot position marked with the arrow, light gray cells are later ignored for being behind a wall and outside the room

types. The connection between classification result and room type probability of the cell based on the classified image is given by the following formula:

$$P(R_c|I_t, \mathbf{x}_t) = \begin{cases} \sum_{R_{I_t}} P(R_{I_t}|I_t)\ P(R_c|R_{I_t}) & if\ cell\ c\ visible \\ P(R_c) & else \end{cases} \tag{5.9}$$

$P(R_c|R_{I_t})$ is the probability of the room type of a cell c given the room type mainly shown in the image R_{I_t}. This conditional probability is important because one probability distribution is calculated for the whole image. The combinations form a $|\mathbf{R}|$x$|\mathbf{R}|$ matrix. The diagonal elements, where $R_c = R_{I_t}$, are set to a parameter V_{eq}. Instead of presuming no prior knowledge and making all other values equal, prior knowledge is gathered to get better mapping results. The off-diagonal elements are weighted based on how related the two room types are. How much two room types are related is estimated using Divisi2[1], a library for extraction of information from commonsense databases, and the ConceptNet4 commonsense knowledge base.

[1] https://github.com/commonsense/divisi2

5.5.4 Doorway Mapping

The doorway mapping inserts and updates the poses of doorways in the semantic room map, using the result of the doorway detection. The result of the doorway detection is a set of doorway proposal poses in the coordinate frame of the robot. These proposal poses are transformed into the map frames of the particles and a data association problem has to be solved. If no already found doorway is within a threshold distance to a proposal pose, a new doorway is added into the map. Otherwise the best fitting doorway, selected based on distance and orientation, is updated. This is shown in Figure 5.10. A running average filter is used to estimate the true doorway pose. Due to the orientation constraint of doorway detections, doorways are always facing out of the room.

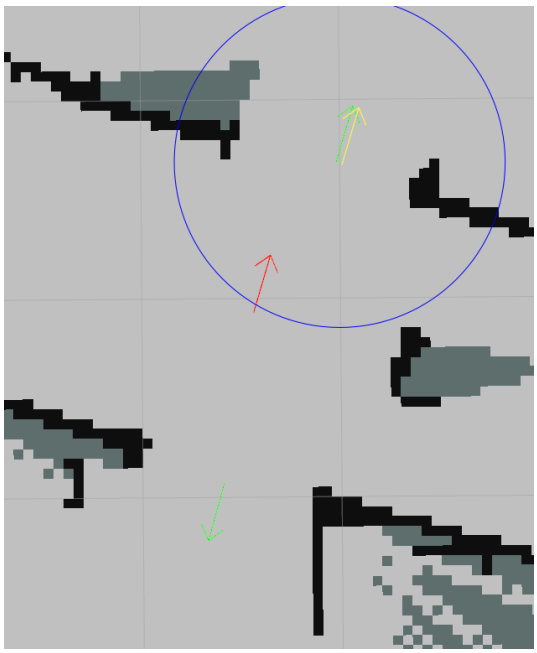


Figure 5.10: Doorway mapping example: The red arrow is the current robot pose, the green arrows are already found doorways, the yellow arrow is the result of the door detection which is used to update the pose of the nearby doorway and the blue circle marks the area which had to be free of doorways for a doorway to be added

5.5.5 Object-Probability-Map Estimation

This subsection describes how all the information stored in the semantic room map is put together to get a 3D grid map containing the probabilities of an object of the searched type being in a grid cell. In a first step the room types in areas which were seen by the LIDAR, but not by the RGBD-camera, are estimated. Then the object probabilities based on the Room-Type-Map and the commonsense knowledge are calculated and in a last step these are fused with the information in the Object-Map to obtain the Object-Probability-Map. The Object-Probability-Map contains no information not already present in another map. Therefore, this map is only created when requested and only for the searched object type.

Estimation of room types in areas seen by the LIDAR, but not by the camera

The LIDAR has a field of view of 240°, while the RGBD-camera has a horizontal field of view of only 58°. Therefore, it is possible that areas were seen by the LIDAR, but not by the RGBD-camera. An example is shown in Figure 5.11. Those cells are considered as possible object locations and are therefore of interest. Cells neither seen by the camera nor the LIDAR are ignored, as they are most likely outside the room. The room type distributions in the areas only seen by the LIDAR are estimated based on the average room type distribution $P_{avg}(R|I_{1:t}, \mathbf{x}_{1:t})$ in the room and the room type distribution of the neighborhood. The average room type distribution is calculated with:

$$P_{avg}(R|I_{1:t}, \mathbf{x}_{1:t}) = \frac{1}{|\mathbf{C_{seen}}|} \sum_{c \in \mathbf{C_{seen}}} P(R_c|I_{1:t}, \mathbf{x}_{1:t}) \tag{5.10}$$

$\mathbf{C_{seen}}$ is the set of cells in the Room-Type-Map which were at least once updated. The room type distributions in cells only seen by the LIDAR are estimated using a two dimensional Gaussian kernel $G(\Delta x, \Delta y)$, as can be

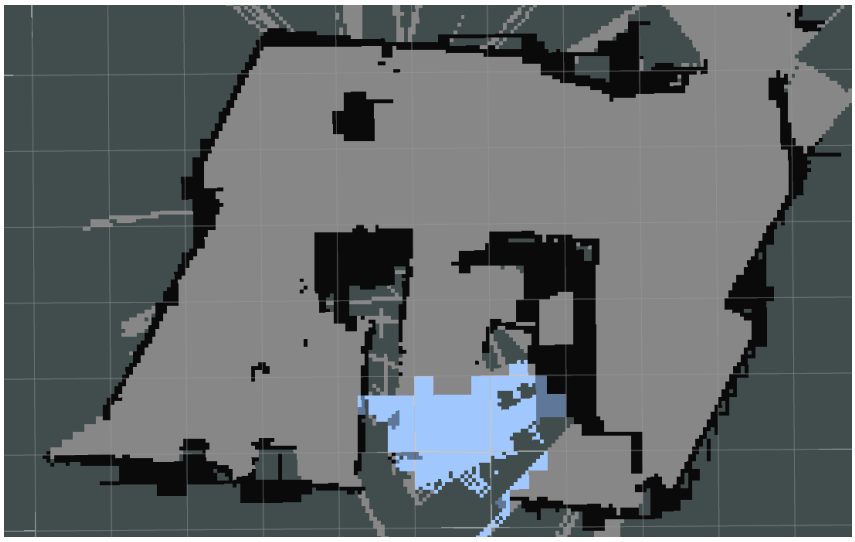


Figure 5.11: An explored room; blueish areas were seen by the LIDAR, but not by the camera; dark blue cells are occupied, light blue cells are free

seen in the following formula:

$$P(R_c|I_{1:t}, \mathbf{x}_{1:t}) = \sum_{c_2} \begin{cases} G(\Delta x, \Delta y) P(R_{c_2}|I_{1:t}, \mathbf{x}_{1:t}) & if\ c_2\ was\ seen\ by\ camera \\ G(\Delta x, \Delta y) P_{avg}(R|I_{1:t}, \mathbf{x}_{1:t}) & else \end{cases} \tag{5.11}$$

Object probabilities based on the Room-Type-Map

As described in Section 5.3 there exists a correlation between the room type of an area and the objects located in this area. This correlation is captured in the matrix KB which contains the probability of an object o being in an image showing room type r. In a first step another matrix $KB^*_{r,o}$ has to be calculated, which contains the probability of an object of type o being in a grid cell of room type r. Assuming an object does not occupy multiple cells and N_v cells being visible in an typical image of the dataset, the following formula is derived:

$$KB^*_{r,o} = 1 - (1 - KB_{r,o})^{\frac{1}{N_v}} \tag{5.12}$$

The probability of an object of the searched object type being in a cell based on the probability distribution of the room type of the cell $P(O^o_{R,c}|I_{1:t}, \mathbf{x}_{1:t})$ can be calculated with:

$$P(O^o_{R,c}|I_{1:t}, \mathbf{x}_{1:t}) = \sum_{R_c} P(R_c|I_{1:t}, \mathbf{x}_{1:t}) P(O^o_{R,c}|R_c) = \sum_{R_c} P(R_c|I_{1:t}, \mathbf{x}_{1:t}) KB^*_{R_c,o} \tag{5.13}$$

This calculation uses the assumption that the room type is independent of the z-coordinate of a cell, so the 2D Room-Type-Map is sufficient.

Object probabilities based on all information

In a last step the information in the Object-Map is combined with the object probabilities estimated based on the Room-Type-Map. This is done using the following formula:

$$P(Obj^o_c|I_{1:t}, \mathbf{x}_{1:t}) = \frac{1}{Z} P(O^o_{R,c}|I_{1:t}, \mathbf{x}_{1:t}) P(O^o_c|I_{1:t}, \mathbf{x}_{1:t}) \tag{5.14}$$

Z is a normalization term, as $P(Obj^o_c|I_{1:t}, \mathbf{x}_{1:t}) + P(\neg Obj^o_c|I_{1:t}, \mathbf{x}_{1:t}) = 1$ and can be calculated with:

$$Z = P(O^o_{R,c}|I_{1:t}, \mathbf{x}_{1:t}) P(O^o_c|I_{1:t}, \mathbf{x}_{1:t}) + P(\neg O^o_{R,c}|I_{1:t}, \mathbf{x}_{1:t}) P(\neg O^o_c|I_{1:t}, \mathbf{x}_{1:t}) \tag{5.15}$$

Some post-processing is done on the Object-Probability-Map. The probabilities of cells assumed to be outside the room are set to zero. These cells are in areas never seen by the RGBD-camera or the LIDAR, or in areas behind doorways. Probabilities in free space are set to zero because if there would be an object, the robot would not be able to drive there and also probabilities of cells near the ground are set to zero because the ground is ignored in the 3D map. For every other cell a minimum probability is set which is half the value of the room type based probability.

5.5.6 Topological Mapping and Room Transitioning

The topological map consists of nodes for every room which contain the semantic room maps and condensed information for the high-level planning based on the semantic room maps. It also contains edges, where each edge corresponds to a doorway and holds information about which rooms are connected and the pose of the connecting doorway in both rooms. The topological mapping is responsible for the creation of this topological structure, including the initialization of new room maps, the localization in the topological map and the handling of room transitions.

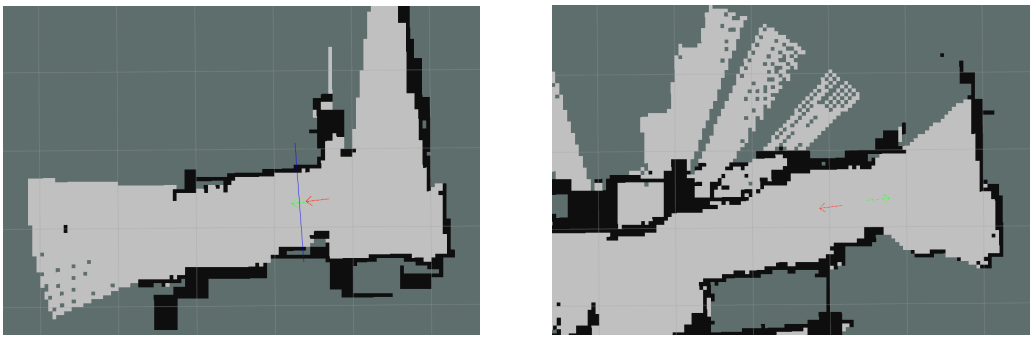
When a new doorway is found a new node is inserted into the graph with an edge connecting the node of the current room and the new node. An empty semantic room map is assigned to the new node. In this map the newly detected doorway is inserted to have a valid connection between the two rooms with corresponding doorway poses in both room maps. In this work environments are assumed to have no loops. Therefore, the room behind a newly found doorway cannot be already discovered. Without this assumption complicated loop closure would be necessary.

When a doorway is passed the semantic room map of the new room has to be activate and the old one deactivated. A doorway is passed if the robot is more than a threshold distance through the doorway. The threshold is marked with a blue line in the left image in Figure 5.12. It is important to achieve a hysteresis and avoid unstable behavior at the doorway. In a first step the mapping in the old room is stopped. Then the mapping in the new room is set up. If the room was never visited before, a new particle filter is created with the specified number of particles and empty maps for all particles. Otherwise the best particle, including its maps, is duplicated to have the specified number of particles for the mapping. In a next step the poses of the new room's particles are set to the pose of the best particle of the old room. This is sufficient as within a doorway the localization is very accurate. The pose has to be transformed into the new room's coordinate frame with

$$\mathbf{P_{new}} = P_{D_{new}} T_{flip} P_{D_{old}}^{-1} \mathbf{P_{old}} \tag{5.16}$$

$\mathbf{P_{old}}$ is the particle pose in the old room's coordinate frame and $\mathbf{P_{new}}$ is the particle pose in the new room's coordinate frame. $P_{D_{old}}$ and $P_{D_{new}}$ are the

poses of the doorway in the old and new room and T_{flip} is a 180° rotation matrix. The rotation is necessary as the doorway poses have an opposite orientation. After that all particles except the best one in the old rooms particle filter are discarded, as those particles are not meaningful when the robot returns to the room. In the last step the mapping in the new room is started. Figure 5.12 shows the map before and after a doorway passing. The maps contain also space in the other room. This is necessary for safe navigation through the doorway. However, space behind doorways is ignored in the Object-Probability-Map and in the search planning.



(a) Room before doorway passing (b) Room after doorway passing

Figure 5.12: Doorway passing; red arrow is the robot pose, green arrow is the doorway pose and the blue line marks where the map switch is triggered

5.5.7 High-Level Room Information Generation

The information in the semantic room maps has to be condensed into a compact form containing only the information necessary for the high-level planning. These high-level attributes of a room are

- the total probability of an object of the searched object type being in the room
- the time needed to completely search the room
- the expected value of the time the robot searches in the room until an object of searched type is found or the room is completely searched
- the time to traverse the room for all doorway pairs

- the time to leave the room for all doorways

The time necessary for exploration is ignored, as the exploration is difficult to model. On the one hand it takes time, but on the other hand some objects might be found during the process and information about other rooms and likely object locations is gathered. Therefore, ignoring the time for exploration seems to be a reasonable idea. Another used idea concerning exploration is setting the high-level attributes of unvisited rooms to very optimistic values. This is done to encourage the robot to peek into rooms. Peeking into a room is a desirable behavior as it takes relatively little time and gives the robot much more information about the environment.

Total object probability in a room r $\mathbf{P_r(o)}$

A searched object being in a room is the complementary event of no cell within the room containing the searched object and therefore can be calculated with:

$$P_r(o) = P(Obj^o|I_{1:t}, \mathbf{x}_{1:t}) = 1 - \Big(\prod_{c \in \mathbf{C_r}} (1 - P(Obj_c^o|I_{1:t}, \mathbf{x}_{1:t})) \Big) \tag{5.17}$$

Three types of cells within a room $\mathbf{C_r}$ have to be distinguished. There are the explored cells $\mathbf{C_{r,ex}}$, whose object probabilities are known. In not fully explored rooms also not explored cells, lying in unexplored areas of the room, have to be considered. The number of unexplored cells is estimated with:

$$N_{unex} = \begin{cases} 0 & if\ room\ explored \\ max(N_{room} - |\mathbf{C_{r,ex}}|, N_{min}) & else \end{cases} \tag{5.18}$$

N_{room} is a general estimate of the size of an unexplored room and N_{min} is a minimal number of unexplored cells in a not explored room. The object probability in those cells is estimated based on the average room type probability distribution $P(R_{avg})$ which is calculated according to Equation

5.10. For the calculation of $P(Obj^o_{unex})$ the correlation matrix between room type and object KB^* is used:

$$P(Obj^o_{unex}) = \sum_{R_{avg}} P(R_{avg}) P(Obj^o_{unex}|R_{avg}) = \sum_{R_{avg}} P(R_{avg}) KB^*_{R_{avg},Obj^o_{unex}} \tag{5.19}$$

Cells of the third type are technically not within the room. Those are cells in undiscovered adjacent rooms. To avoid the addition of uncertain rooms into the topological map those are considered to be part of the discovered adjacent room during the probability calculation. The expected number of those cells N_{undis} is estimated with:

$$N_{undis} = \begin{cases} 0 & if\ room\ explored \\ max\Big(\sum_{R_{avg}} P(R_{avg}) N_{adj}(R_{avg}) - N_{r,dis}, 0\Big) N_r & else \end{cases} \tag{5.20}$$

$N_{adj}(R_{avg})$ is the average number of adjacent rooms for a room of type R_{avg}. Based on results presented in [39] $N_{adj}(corridor)$ is set to 6 and $N_{adj}(R)$ is 1.2 for all other room types. The number of already discovered adjacent rooms $N_{r,dis}$ has to be subtracted and obviously the number of undiscovered adjacent rooms cannot be smaller than zero. The probability of the object being in one of those cells $P(Obj^o_{undis})$ is calculated assuming an equally distributed room type probability distribution in the undiscovered room.

Putting all together the following formula is obtained:

$$P_r(o) = 1 - \Big(\prod_{c \in \mathbf{C_{r,ex}}} (1 - P(Obj^o_c | I_{1:t}, \mathbf{x}_{1:t})) \Big) \\ (1 - P(Obj^o_{unex}))^{N_{unex}} (1 - P(Obj^o_{undis}))^{N_{undis}} \tag{5.21}$$

Full search time in room r $\mathbf{T_r}$

The time needed to search a room is difficult to estimate because it not only depends on the room but also on what the search planner is doing and how well the navigation works. In this work a minimalistic model is used because a lot of test data would be needed for a more sophisticated

model which is not possible to generate in a reasonable time. The number of occupied cells in the 2D map $|\mathbf{C_{occ2d}}|$ has a high correlation with the search time. A high number means many possible object locations and also many obstacles, which slow down the robot movement. Therefore, the time needed to completely search a room is the number of occupied cells in the 2D map within the room times a parameter T_c found in test runs. As in the calculation of the total object probability, unexplored areas in the room have to be taken into account. This is done using a general estimate for the search time in an unexplored room T_{room} and a minimum time needed for searching in unexplored areas of the room T_{min}. This results in

$$T_r = \begin{cases} |\mathbf{C_{occ2d}}|T_c & if\ room\ explored \\ max(T_{room}, |\mathbf{C_{occ2d}}|T_c + T_{min}) & else \end{cases} \tag{5.22}$$

Expected search time in room r $\mathbf{E_r(o)}$

The expected time spent searching in a room also takes into consideration that a searched object might be found and the search can be stopped earlier. The expected value is calculated with

$$E_r(o) = \int_0^\infty f_r^o(t) \cdot t \cdot dt \tag{5.23}$$

The search ends if the object is found or all areas in the room are covered. Therefore, the probability distribution of ending the search over time $f_r^o(t)$ is

$$f_r^o(t) = (1 - P_r(o))\delta(T_r) + f_{find,r}^o(t) \tag{5.24}$$

with $f_{find,r}^o(t)$ being the probability distribution over time for finding the object which has to fulfill

$$\int_0^\infty f_{find,r}^o(t) \cdot dt = P_r(o) \tag{5.25}$$

and

$$f_{find,r}^o(t) = 0 \ \ \forall\ t < 0, t \geq T_r \tag{5.26}$$

The estimation of $f^{o}_{find,r}(t)$ is done by discretization of time. To calculate the probability of finding the object in a time step $[T_{i-1}, T_i)$ the object probabilities of all cells in the room are sorted and assigned to the time steps in decreasing order. The probability of finding the object in a certain time step $[T_{i-1}, T_i)$, P_i, is then calculated with

$$P_i = \prod_{j=0}^{i-1}(1 - P_j) \cdot (1 - \prod_{c \in \mathbf{C_i}} (1 - P(Obj^{o}_{c} | I_{1:t}, \mathbf{x}_{1:t})) \tag{5.27}$$

P_0 is set to 0. The estimated search time is then

$$E_r(o) = \sum_i P_i T_i + (1 - P_r(o))T_r \tag{5.28}$$

This model was chosen as it takes into account the distribution of object probabilities in the room and the decreasing probability of finding an object due to the objective of the search planner. While the time discretization results in too high expected search times, the ordering is too optimistic and results in too low times. These two effects should balance each other out.

Times for movement

The time spent moving between the rooms is also considered in the high-level planning. The time necessary to drive from a starting room to a target room is split into the time necessary to leave the starting room and the times necessary to traverse rooms on the way. The time it takes the robot to drive from a pose A to a pose B is estimated by the distance between A and B multiplied with an average velocity. So for every pair of doorways in the room the estimated movement time is calculated based on their distance. The time to leave a room depends on the location of the robot which can be anywhere in the room after exploration or search. Therefore, an assumption is made that the robot is in the middle of the room which is the center of mass of all free cells. For every doorway the time necessary to leave the room is calculated based on the distance from the center of the room to the doorway.

5.6 High-Level Planner

This section covers the high-level planner which operates on the topological map and selects the room the robot should search next. Based on the selected target room it also generates high-level tasks which are then planned and executed by the low-level planner. The expected search time was chosen as the objective function to be minimized by the planner. Other possible objectives for the planner would be maximizing the probability of finding the object in limited time or minimizing the time until a certain confidence about the existence of a searched object is reached. These tackle slightly different problems, but the results should be quiet similar in most scenarios. The high-level planner is executed when a search task is started and also every time a high-level tasks finishes.

5.6.1 High-Level Planning

The objective of the high-level planner is to find the optimal search path $\mathbf{p}^* = (r_1, r_2, ..., r_{|\mathbf{R}|})$ with r_i being the i^{th} room to search and $|\mathbf{R}|$ being the number of not searched rooms given a starting room r_0. A valid search path contains every not already searched room exactly once. In this work the optimal path is the valid path with the minimal expected search time.

$$\mathbf{p}^* = \underset{\mathbf{p} \in valid paths}{\arg\min} \; E(t|\mathbf{p}, r_0) \tag{5.29}$$

The expected search time given a staring room r_0 and a search path $\mathbf{p}$ can be calculated with

$$E(t|\mathbf{p}, r_0) = \sum_{i=1}^{|\mathbf{p}|} \Big(\prod_{j=1}^{i-1} (1 - P_{r_j}(o)) \Big) (M_{r_{i-1}, r_i} + E_{r_i}(o)) \tag{5.30}$$

The definition $\prod_{j \in \emptyset} = 1$ is used. This calculation is illustrated in an example calculation in Figure 5.13. Searching rooms with high object probabilities first results in a lower expected search time as it is more likely that other rooms do not have to be searched. However, the probability of finding the

object is the same for all paths. The matrix M contains estimations of the movement times, where M_{r_i,r_j} is the time needed to drive from room r_i to room r_j. The matrix elements are calculated using the movement times described at the end of Section 5.5.7 and the topological map. The expected search timers $E_{r_i}(o)$ and the total object probabilities $P_{r_j}(o)$ are also described in Section 5.5.7.

The possible paths form a search tree, as illustrated in Figure 5.13. The tree representation shows possibilities for optimizations in the calculation like the reuse of intermediate results and pruning.

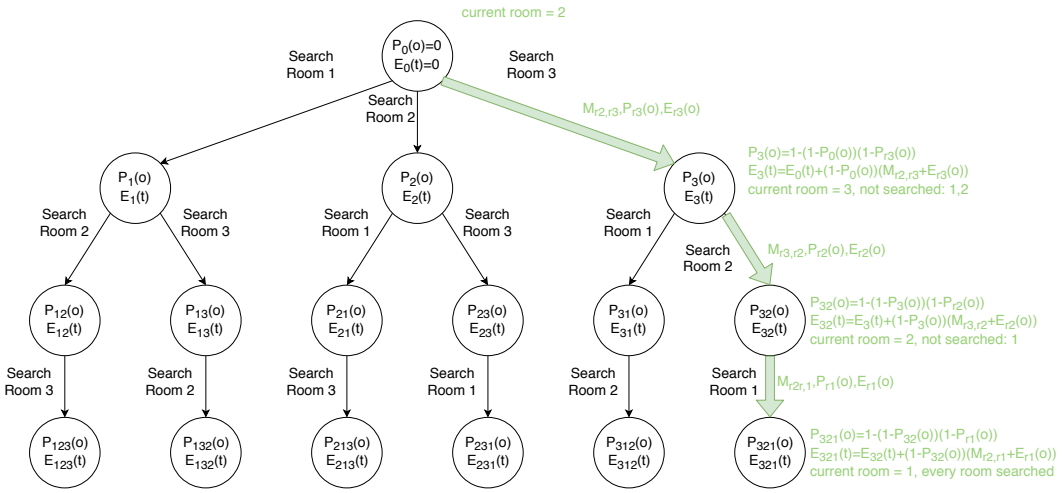


Figure 5.13: Search tree of all possible search paths: The edges are search actions and the nodes are states which also hold the probability of the object already been found and the expected search time so far; in green an example calculation is shown

5.6.2 Generation of next High-Level task

The search path contains only the order, in which the rooms are searched. To execute the search path tasks have to be generated, which are then planned and executed by the low-level planner. These high-level tasks are:

- **Move through doorway:** A doorway is specified through which the robot should drive to enter another room.

- **Peek:** The robot should turn left and right to quickly gather some information about the room.
- **Explore room:** The robot should explore the current room to obtain knowledge about likely object locations and find all doorways in the room.
- **Search room:** The robot should search for the target object in the current room

The high-level planner keeps track of whether a room was visited, explored or searched. Using this information the generation of the high-level tasks is done as shown in Figure 5.14. The path to the room to search, containing the rooms on the way, is retrieved out of the topological map. As replanning is done after every high-level task only the generation of the next high-level task is necessary.

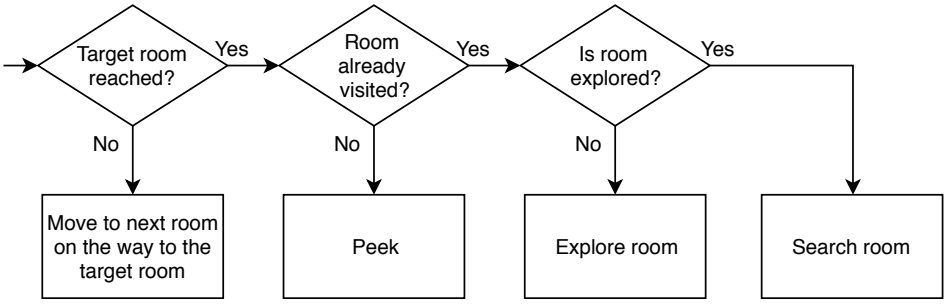


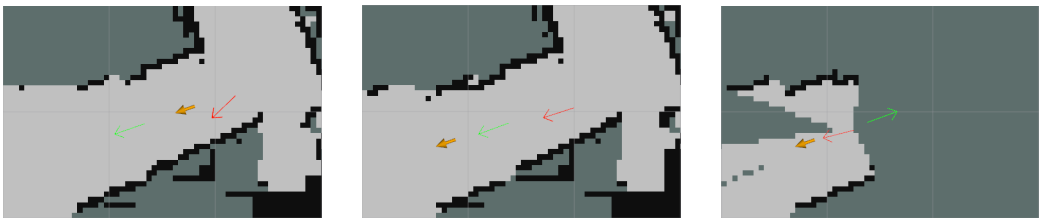
Figure 5.14: Flow diagram for high-level task generation

5.7 Low-Level Planner

Based on the currently pending high-level task the low-level planner generates goal poses for the robot. All high-level tasks can be executed within a single room, with the exception of the move through doorway task. For the other tasks the low-level planner can ignore other rooms to make the planning easier. The low-level planner also decides if the searched object was found, using the results of the object detection. An overview on how those tasks are planned and executed is given in this section.

5.7.1 Move Through Doorway Task

With this task the robot is initiated to drive through a specified doorway into the next room. It is the only high-level task where the robot can drive into another room. The execution of this task is illustrated in Figure 5.15. In a first step the robot is sent to a position in front of the doorway facing towards the doorway. In this process the door pose becomes more accurate and possible obstacles behind the doorway are detected. In a second step the robot is sent to a goal behind the doorway to drive into the next room. During the execution the low-level planner keeps track of the doorway pose and changes the goal pose accordingly. This is especially important when the map switch occurs, as the doorway pose might be very different in the new room's map coordinate frame.



(a) align in front of doorway (b) set goal behind doorway (c) finish after map switch

Figure 5.15: The three steps of the move through doorway task: the robot pose is in red, the current goal pose is in orange, and the door pose is in green

5.7.2 Peek Task

The purpose of this task is to quickly get a first impression of a room never visited before. This is achieved by turning the robot left and right, so the area seen by the robot is increased. If the object detection detects a searched object with a sufficiently high confidence during this task, the object is assumed to be found and the task is stopped.

5.7.3 Explore Room Task

The goals of the explore room task is to find all doorways in the room, to detect likely object locations and also to find easy to spot target objects. 2D exploration is done to achieve these goals. After finishing 2D exploration all doorways are guaranteed to be found, as unexplored space would be behind an undiscovered doorway. During the exploration most of the room is seen with the camera, though most areas only in a few images. Some areas might not be seen at all by the camera because the exploration is finished when all accessible cells were seen by the LIDAR. Due to the smaller field of view of the camera, some cells might only be seen by the LIDAR and not by the camera and also some cells in inaccessible areas might not be seen.

A frontier-based approach is used for the exploration, similar to the approach proposed by Yamauchi [31]. In this thesis a frontier is defined as an accessible cell in the occupancy grid map adjacent to at least one unexplored cell, as shown in Figure 5.16. Based on an objective function a frontier is selected and the robot is sent to this frontier. On the way to the frontier some area beyond the frontier is explored. In this work the objective function is the distance between the robot and the frontier, where nearer frontiers are preferred.

Only the current room should be explored. To prevent the robot from driving out of the room during exploration all detected doorways are blocked by virtual obstacles, as shown in Figure 5.17.

If the object detection detects the searched object with a sufficiently high confidence, the object is assumed to be found. The estimated location in space is reported and the search is stopped. A decision based on multiple images, like during the search task, is not done during the exploration. Another reason to stop the exploration early is the detection of a new doorway, so the high-level planner can be executed on the updated topological map.

Figure 5.16: A map with accessible cells (purple and yellow cells) and frontiers (yellow cells)

5.7.4 Search Room Task

The goal of the search room task is to find an object of a given type within the current room or to search the whole room if no searched object is in the room. The low-level planner has to find the best next view pose which minimizes the expected search time, based on the information gathered. Also the decisions have to be made if an object was found and if the room was fully searched.

In this work a greedy approach is used to find the best next view pose. The best view pose $\mathbf{p}^*$ is selected by:

$$\mathbf{p}^* = \arg\max_{\mathbf{p}} \frac{P(o|\mathbf{p}) + B(\mathbf{p})}{T(\mathbf{p}_0, \mathbf{p})} \tag{5.31}$$

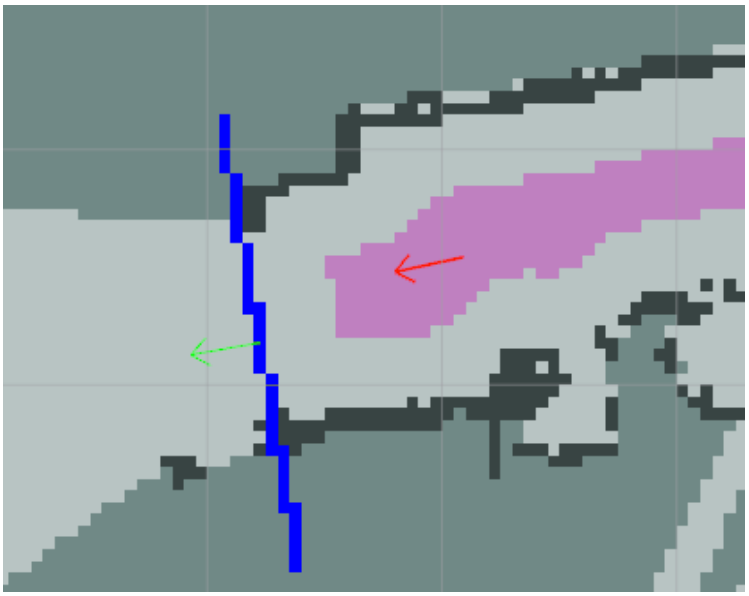


Figure 5.17: Doorway (green arrow) is blocked by a virtual obstacle (blue) during exploration

$P(o|\mathbf{p})$ is the probability of finding an object of type o when the robot is at pose $\mathbf{p}$. $B(\mathbf{p})$ is an additional term to promote coverage and is also necessary for the termination criteria, as described later. $T(\mathbf{p}_0, \mathbf{p})$ is the time to drive from the current pose $\mathbf{p}_0$ to the pose $\mathbf{p}$. Assuming movement on the shortest path between two poses, $T(\mathbf{p}_0, \mathbf{p})$ is estimated with:

$$T(\mathbf{p}_0, \mathbf{p}) = \frac{|\measuredangle(\mathbf{p}_0, \overrightarrow{\mathbf{p}_0\mathbf{p}})| + |\measuredangle(\overrightarrow{\mathbf{p}_0\mathbf{p}}, \mathbf{p})|}{\omega} + \frac{|\overrightarrow{\mathbf{p}_0\mathbf{p}}|}{v} + T_{const} \tag{5.32}$$

ω and v are the average rotational and translational velocities and T_{const} is a constant time added to account for planning time and other delays.

Estimating the probability $P(o|\mathbf{p})$ is much more difficult. It depends on what is visible when taking an image at pose $\mathbf{p}$, the object probabilities in the visible space and how likely a searched object located in the visible space is detected. Therefore, $P(o|\mathbf{p})$ is calculated with:

$$P(o|\mathbf{p}) = 1 - \prod_{c \in cells\ in\ room} (1 - P(Obj_c^o|I_{1:t}, \mathbf{x}_{1:t})P(vis_c|\mathbf{p})P(det_c|\mathbf{p})) \tag{5.33}$$

Determining visible space is computational expensive and also not really possible in messy environments without a high quality 3D map, therefore a heuristic is used. The used heuristic reduces the problem from 3D to 2D and the limited vertical field of view is ignored. The basic assumption is that an object can always be seen from the direction of the nearest cell accessible by the robot. This is true if objects are side-by-side and not behind each other. The yellow arrows in Figure 5.18 show these directions, which are referred to as best directions. The probability of an object being visible depends on the angle between the best view direction and the direction from robot to object. This is modeled with

$$P(vis_c|\mathbf{p}) = \begin{cases} e^{-\frac{\measuredangle(\mathbf{p},\vec{\mathbf{p}^c})^2}{2\sigma^2_{view}}} & if\ in\ field\ of\ view \\ 0 & else \end{cases} \tag{5.34}$$

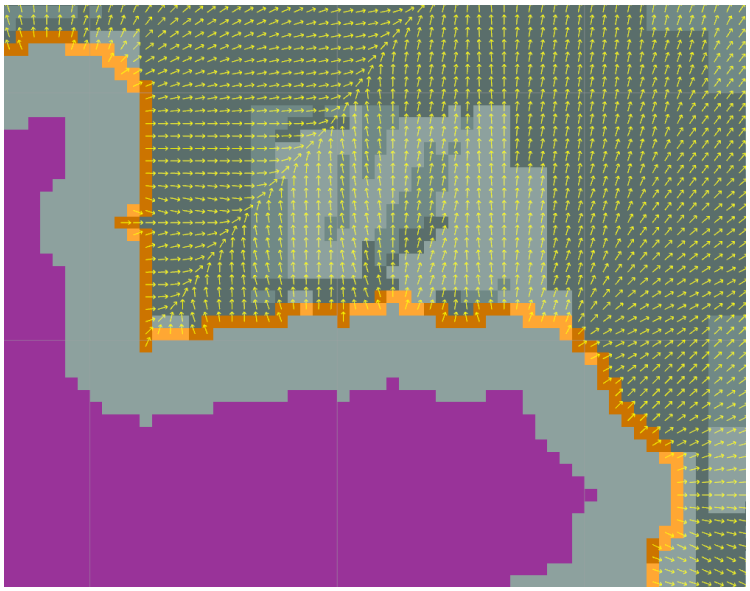


Figure 5.18: Directions from the nearest accessible cell for every cell outside of the border (yellow arrows), accessible area (purple), border of free space (orange) and 2D grid map in the background

This model of visibility is very pessimistic as many objects are also visible from other view angles. To reduce the number of unnecessary views, cells

seen more often than a threshold value are assumed to be sufficiently viewed and ignored in later probability calculations.

The model used to estimate the probability of detecting the object depending on its location relative to the robot is shown in Figure 5.19. Near cells are weighted higher as small objects are difficult to detect from far away. Large objects on the other hand occupy multiple cells and therefore view poses covering the whole object are implicitly preferred. At the edge of the field of view successful object detections are less likely, which is also taken into account.

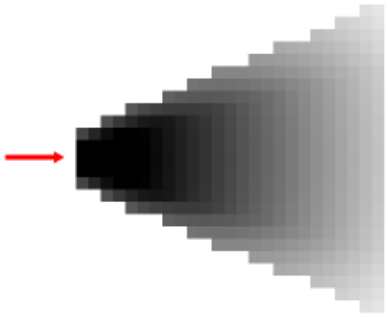


Figure 5.19: Probabilities for detecting the object depending on the location relative to the robot (red arrow); the probability of cells is 1 for black cells and 0 for white cells

To find the best next view pose sample poses are generated at regular x and y distances with multiple different orientations. This discretization is necessary because only a limited number of poses can be evaluated. A sample pose is valid if it is accessible by the robot and was not visited before. From the valid sample poses the best one is chosen based on Equation 5.31.

Two approaches are used to decide if an object was found. The first approach uses only the current image. The object is assumed to be found if a searched object was detected by the CNN with a sufficiently high confidence. In this case the location of the found object is estimated based on the position in the image and the depth values of the detection. The second approach combines the information from multiple images and is done like the creation of the Object-Map in Section 5.5.2. Thus, also a 3D map is created. Here the

performance is not as much a concern for two reasons. Firstly, the accuracy of the localization should be accurate enough after exploration, so this has not to be done for multiple particles. Secondly, only the searched object is of interest. Therefore, more detection samples can be used, leading to more reliable results. If the probability of a searched object being in a cell is higher than a threshold, the object is assumed to be found. If an object is found the position of the object is signaled and the search is stopped.

In case that no object can be found the planner needs to decide if the room was fully searched. It is impossible for the system to decide which cells in a room can possibly be seen, at least based on the generated 3D map. Therefore, another heuristic is used to define a termination criteria. Above the assumption was made that a cell is seen from the nearest accessible cell. In reverse this means the room is fully searched if the robot has looked outward at every accessible cell. Assuming that only the direction is important and not the distance, the robot has only to look outward over the border of the accessible area. A set of cells representing this border is created and the direction from the nearest accessible cell to the border is calculated for every cell in this set. This is shown in Figure 5.20. In addition every cell in the set gets a counter. When the object detection is executed on an image, all cells of the border in the field of view of the camera with an orientation similar to the direction of the camera are increased by one. This is illustrated in Figure 5.21. Using the same line of thought the term $B(\mathbf{p})$ is calculated with:

$$B(\mathbf{p}) = \begin{cases} V_B |\mathbf{C}_{\mathbf{B}_{\mathbf{interest}}}| & if\ |\mathbf{C}_{\mathbf{B}_{\mathbf{interest}}}| > 0 \\ -\infty & else \end{cases} \tag{5.35}$$

where $|\mathbf{C}_{\mathbf{B}_{\mathbf{interest}}}|$ is the number of border cells, whose counter would be increased in a view from pose $\mathbf{p}$ and is lower than a threshold parameter. V_B is a parameter weighting coverage versus finding probability. Ultimately the termination criteria is defined with $P(o|\mathbf{p}^*) = 0 \vee B(\mathbf{p}^*) \leq 0$.

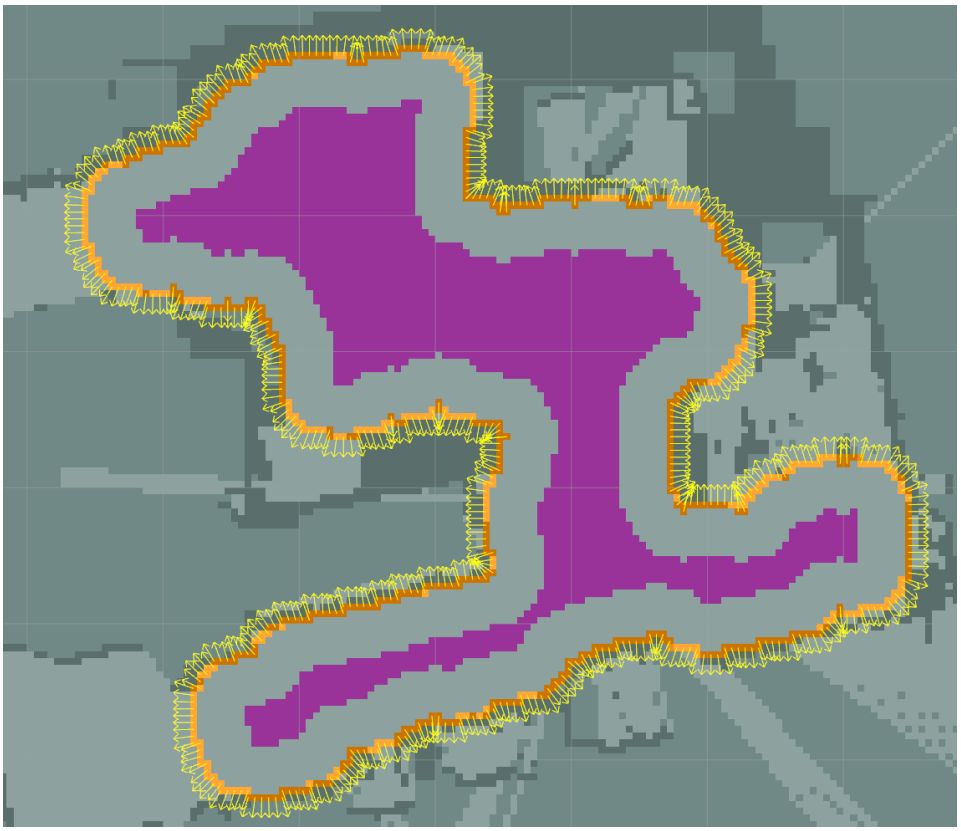


Figure 5.20: Accessible area (purple), border of free space (orange), outward directions of the border cells (yellow arrows) and 2D grid map in background

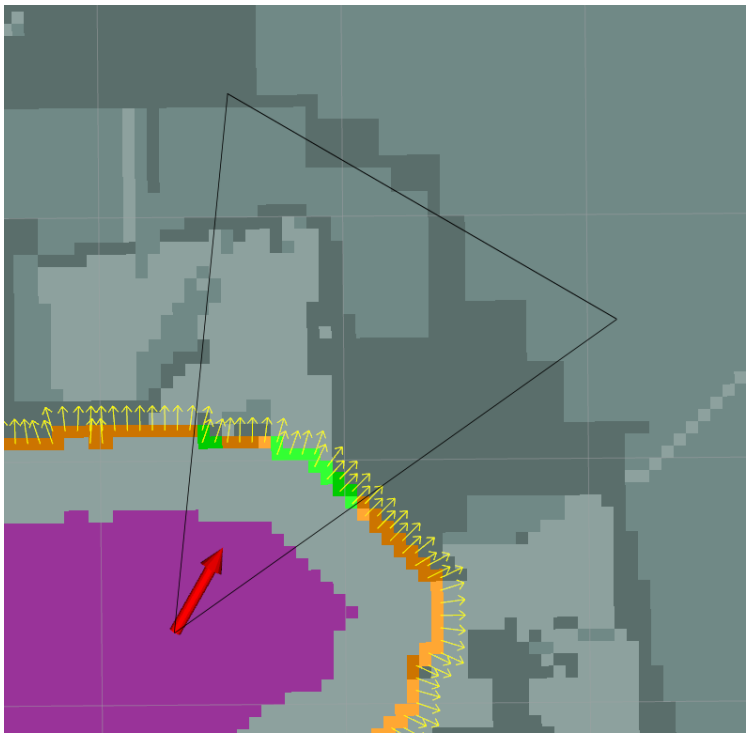


Figure 5.21: Update of border cell counter: Only cells within the field of view (black triangle) and having a similar orientation as the camera (red arrow) are updated (green cells); other cells on the border are not updated (orange cells)

6 Implementation

In this chapter details of the implementation of the concept presented in Chapter 5 are described. The software runs on a Linux operation system and uses ROS Kinetic as middle-ware. The usage of ROS allows easy communication between different processes, also called nodes, and the easy integration of existing software packages. Existing packages are used for the communication with the robot and the sensors. The source code of some packages was used as a starting point for the implementation. The in ROS integrated tf library, which keeps track of the coordinate frames and allows nodes to obtain the most recent transformation between two coordinate frames and also transformations between coordinate frames at past times, is also useful. Other used libraries are OpenCV[1], PCL[2], the mapping libraries Octomap[3] and GMapping[4], and the deep learning libraries Caffe[5] and Darknet[6]. OpenCV is an open-source image processing library. In this work it was not only used for image processing, but also in the internal representation of grid maps and for operations executed on those maps. PCL (Point Cloud Library) was used for operations on point clouds, like coordinate transformations or filtering. Except for some debug programs written in Python and the message specifications, the whole implementation was written in C++.

Figure 6.1 shows the architecture of the implementation, with its nodes and the communication channels between the nodes.

[1] https://opencv.org/
[2] http://pointclouds.org/
[3] https://octomap.github.io/
[4] https://www.openslam.org/GMapping.html
[5] http://caffe.berkeleyvision.org/
[6] https://pjreddie.com/darknet/

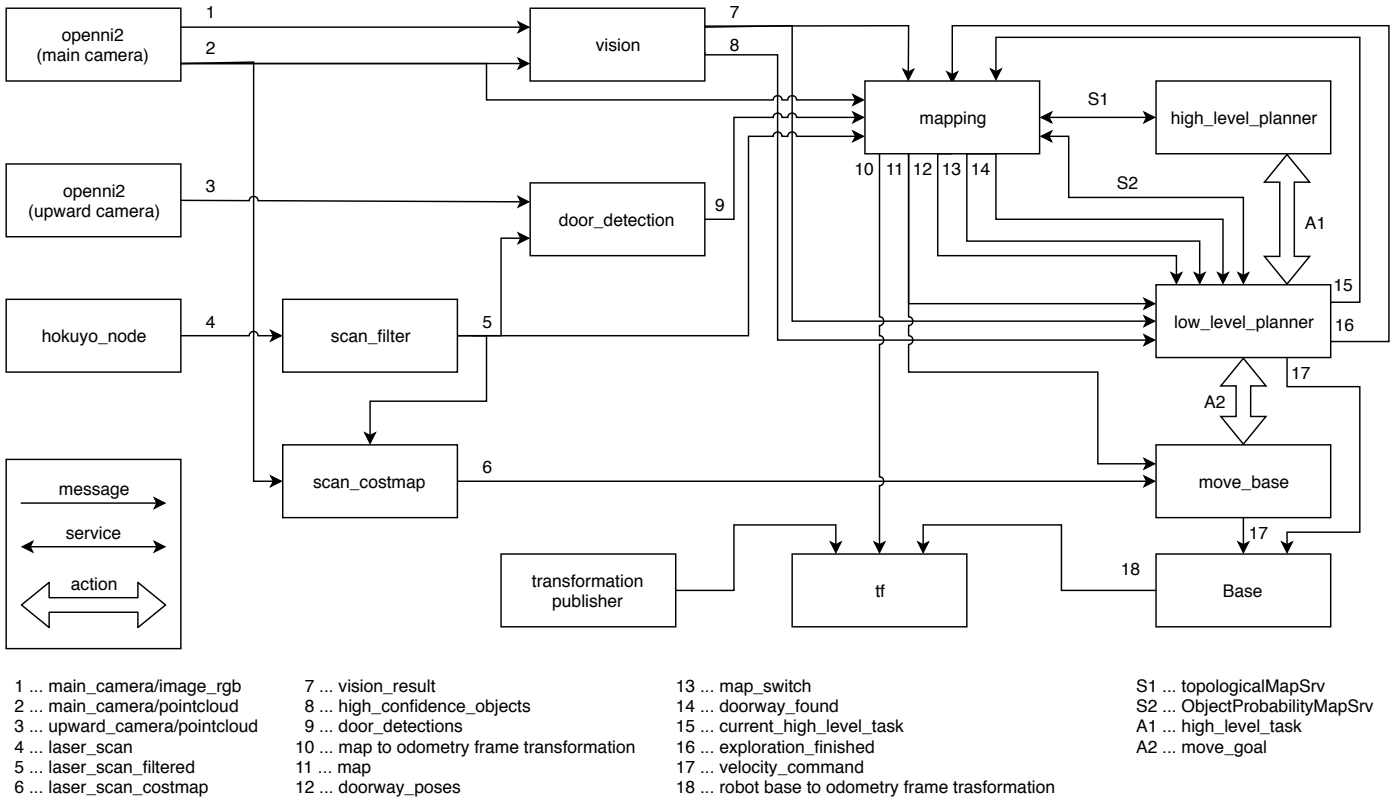


Figure 6.1: Overview over the nodes in the system and the messages, services, and actions for inter-node communication; message connections are drawn from publishing node to the subscribed nodes; nodes connected with the tf package publish updated coordinate transformations

The following sections contain an in-depth description of the implementation of the nodes and the communication between the nodes.

6.1 Hardware Interface

The ASUS Xtion Pro RGBD-cameras communicate with the rest of the system via the openni2-package. The nodes of this package connects to the camera and transforms the raw data from the camera into various ROS-messages. The RGB-image and the depth-registered point cloud of the main camera are used by the vision node. The depth-registration assigns a 3D point to every pixel in the RGB-image. For pixels where this is not possible, the point is set to (NaN,NaN,NaN). The point cloud from the upward looking camera is sent to the door_detection node. To reduce the computational cost, the frame rates of the cameras are reduced to 5 frames per second, as more images cannot be processed anyway.

The hokuyo_node provides access to the LIDAR. It publishes a laser scan message, which contains an array of range measurements in polar coordinates and the information about the angle range and the resolution of the laser scan.

The transformation publisher publishes the transformations of the robot, so the transformations from all the components of the robot into the robot base coordinate frame are available. For this implementation important are the transformations between the two cameras and the robot base and between the LIDAR and robot base.

The interface to the Turtlebot takes velocity commands, which contain a translational and a rotational velocity. These commands are then executed by the robot. The interface also provides the data from the odometry as a transformation between the robot and the odometry coordinate frame.

6.2 Preprocessing

The images and the point clouds can be used without preprocessing. The laser scans on the other hand have to be modified for further use. The scan_filter node prepares the laser scans for the mapping. Therefore, range measurements on the far left and far right of the field of view are discarded. This is necessary as some parts of the robot are visible to the LIDAR, as shown in Figure 6.2. Further, invalid range measurements need to be handled. There are two causes for invalid range measurements. Firstly, the laser beam was not sufficiently reflected by an obstacle. In this case the range measurement should be ignored. Secondly, no obstacle is within the maximum range of the LIDAR. In this case the range measurement should be set to the maximum range, so the mapping can use those beams to clear space. Due to the short range of the used LIDAR this happens often. There is no way to distinguish between these two cases and always assuming one case is not sufficient. Discarding all invalid measurements causes problems in open areas and corridors, as no or not enough information is sent to the mapping. Setting all range measurements to maximum range can result in occupied cells being wrongly classified as free. Therefore only invalid measurements of every fifth scan are set to the maximum range, which tests showed to be a reasonable trade-off.

The navigation is done in 2D, but it is also necessary to consider the obstacles seen by the RGBD-camera. Therefore the laser scans sent to the local planner are modified to contain also those obstacles. This is done by intersecting the beams sent by the LIDAR with the projected point cloud of the RGBD-camera. The range measurement of a beam is then the minimum of the range measurement from the LIDAR and the distance to the first intersection with the projected point cloud. This is show in Figure 6.2.

6.3 Doorway Detection Node

The input data for the doorway detection node are the point clouds from the upward looking RGBD-camera and the range measurements from the LIDAR. The most recent range measurements from the LIDAR are stored for

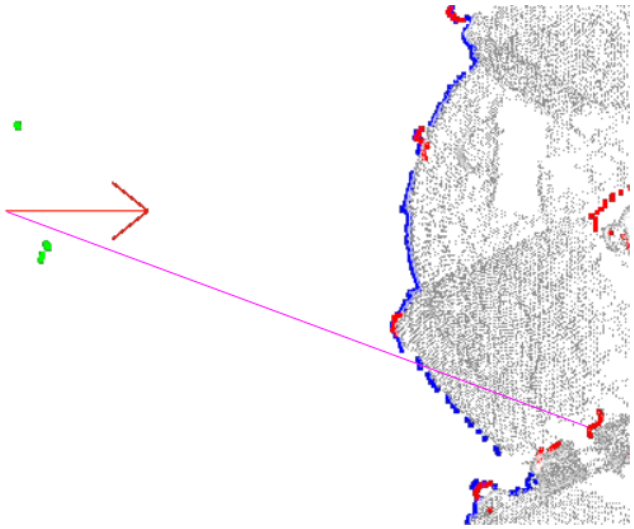


Figure 6.2: Preprocessing of the laser scan: Discarded range measurements (green); result of the scan_filter node (red); projected point cloud of the RGBD-camera (black); example beam (purple); range measurements changed by the scan_costmap node (blue)

further processing. When a new point cloud arrives and the robot has moved considerably since the last execution of the doorway detection, the doorway detection is started. The movement check is done because of performance reasons and to avoid redundant information in the mapping. The output is a set of doorway detections represented as an array of poses in the robot coordinate frame.

The first step is to transform the point cloud from the coordinate frame of the camera into the coordinate frame of the robot using the known static transformation. This transformation was determined manually. First, the z-coordinate, the pitch angle, and the roll angle were determined by comparing the point cloud of the camera with a ceiling of known height. The x-coordinate, y-coordinate, and yaw angle were determined by comparing the point cloud with the laser scan in front of a room corner. In a second step two subsets of points are extracted from the point cloud, one subset containing only points with a height between 1.95 m and 2.05 m and one subset containing only points lower than 1.95 m. Both subsets of points are projected onto the ground and converted into 2D images represented

as OpenCV matrices, as shown in Listing 6.1. The `resolution` and the `origin` specify the conversion between the robot coordinates and the image coordinates. The resulting images are shown in Figure 6.3 (a) and (b). From the left image it can be seen that not only points of the upper part of the door frame are at about 2 m height, but also points on the walls beside the doorway. The purpose of the second image, which also contains those walls, is to eliminate the undesired pixels from the walls. Therefore the second image is subtracted from the first one. Applying the morphological operations closing, to fill possible holes, and opening, to remove stray pixels, results in the filtered image shown in Figure 6.3 (c).

Listing 6.1: Projection of points into corresponding image

```
for(point in pointCloud){
  if(point.z>=1.95 and point.z<=2.05)
    at2mImage(point.y*resolution+origin.y,point.x*resolution+origin.x) = 255
  if(point.z<1.95)
    under2mImage(point.y*resolution+origin.y,point.x*resolution+origin.x) = 255
}
```

(a) Projection onto the ground of points at about 2 m

(b) Projection onto the ground of points lower than 1.95 m

(c) Filtered image

Figure 6.3: Preprocessing of the doorway detection

The filtered image is searched for contours and around every contour found a minimum area rectangle is fitted. Both is done using OpenCV functions.

This is depicted in Figure 6.4, where only one contour was found. All rectangles with $d < 0.05\,m \wedge d > 0.5\,m \wedge w < 0.6\,m$ are ruled out. The width and depth can be converted from pixels to meters using the same resolution as in the conversion into an image.

To execute the other checks described in Section 5.4.3, the latest laser scan from the LIDAR has to be transformed into the correct coordinate frame. Laser scan and point cloud are usually not acquired at the same time, so robot movement in the time between the measurements has also to be considered. This is done using the tf library, which allows to specify not only the coordinate frames when accessing a transformation but also the time of interest. Three areas are defined in Figure 6.4. For every area the number of points from the transformed laser scan within the area are counted. If the sum of points in the red areas is less than 5 and the number of points in the green area is greater than 30, a doorway is found. The former is necessary for a doorway to be not blocked and the parameter was chosen to be 5 to allow some outliers in the laser scan. The latter is necessary as there have to be walls beside the doorway and the parameter was found in tests. If the rectangle was classified as doorway, the pose of the doorway is estimated as described in Section 5.4.3 and added to the result array. This is done for all fitted rectangles fulfilling the size constrains, so the result is an array containing all detected doorways.

6.4 Vision Node

The vision node operates on the RGB-image and the depth-registered point cloud from the main RGBD-camera. The output are two messages. The `VisionResult` message, sent to the mapping node and the low_level_planner node, contains the results of the room type classification and the object detection and is described in Listing 6.2. The `HighConfidenceObjects` message, sent only to the low_level_planner, contains the types and locations of object detections with very high confidence and is described in Listing 6.3.

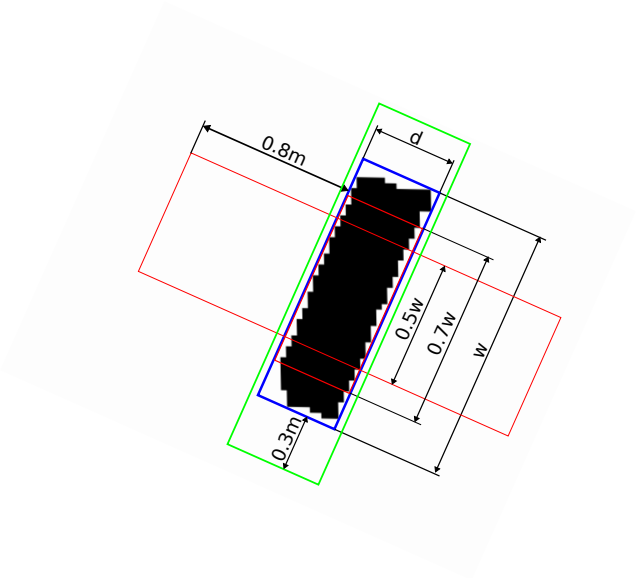


Figure 6.4: The minimum area rectangle (blue) around the filtered doorway contour (black) and specifications of the areas for the doorway check; the number of points from the laser scan has to be less than a threshold in the red rectangles and greater than another threshold in the green rectangle for a doorway being detected

Listing 6.2: `vision/VisionResult`

```
Header header
DetectionSample[] samples
   geometry_msgs/Point position
   float32[] probabilities
float32[] room_type_probabilities
```

Listing 6.3: `vision/HighConfidenceObjects`

```
Header header
geometry_msgs/Point[] positions
int16[] object_types
```

The implementation of this node uses two threads. The first thread waits for input images and point clouds and decides if an images should be processed. The second thread runs the CNNs on the images delivered by

the first thread, creates the output messages and publishes these messages. The multi-threaded approach is used to avoid waiting for a new image and to be able to receive images all the time.

The decision to process an image is made based on the movement of the robot. If the robot turns too fast the images are blurry and therefore discarded. A rotational velocity of 0.5 radians per second was found to be the maximum speed where the object detection produces still reasonable results. If the robot does not move no new information is gathered and no processing is necessary. This also avoids using the same information multiple times in the mapping. The movement of the robot is obtained from the odometry data, which is accessible through the `tf` library. The robot has to move at least 0.3 meters or turn 0.1 radians until a new image is processed. If an image should be processed, the image and the current point cloud are stored in a buffer for the second thread. If an old image is still in the buffer, the old image is discarded.

The images are processed in three steps, starting with the execution of the place classification CNN and followed by the execution of the object detection CNN. In the last step the output messages are created using the results of the CNNs. In the following these steps are described in more detail.

6.4.1 Place Classification

In this thesis a CNN using the VGGNet-16 model and trained on the Places205 dataset is used. The training was done by Wang et al. as described in [36]. This CNN was chosen as it achieves state-of-the-art performance with an accuracy of about 60% on the Place205 testset. Another advantage is that the model of the CNN and the trained weights are publicly available and ready to use with the Caffe deep learning library. Caffe [40] is an open-source deep-learning framework written in C++. Using CUDA, a framework for executing code on the GPU, the CNN can be executed on the GPU, which is essential for running large neural networks with reasonable frame rates. On the used hardware a forward pass takes about 25 ms.

An example code for setting up a CNN and the execution of the classification on an image is distributed with Caffe. This code was modified to use the chosen CNN and adapted to fit into the vision node. The result of the classification is an array of 205 probability values, one for each place category.

6.4.2 Object Detection

For object detection the YOLOv3 [19] object detection CNN is used. This CNN was chosen because according to the related publication it has the best trade-off between speed and accuracy. State-of-the-art object detection systems cannot be used because the used GPU is not fast enough to achieve reasonable frame rates. With YOLOv3 a forward pass still takes about 150 ms. YOLOv3 is available running on a second deep learning framework, which is Darknet [41]. Darknet is open-source, written in C and also uses CUDA for GPU support. This causes problems when linking the vision node because Caffe links CUDA with C++-linkage and Darknet with C-linkage. To resolve this problem Darknet was ported to C++, which involved mainly the renaming of variables with names forbidden in C++.

The code of an example program using YOLOv3 is shipped with Darknet. The necessary post-processing of the result of the CNN is also included in the example code. First the bounding boxes are transformed into image coordinates. Then a non-maximum-suppression is applied. In decreasing order, based on their probability, the overlap with all already processed bounding boxes is calculated and if it is greater than a threshold, the bounding box is discarded. The example code was adapted and integrated into the vision node code. The main adaptation was to change the input size to 640x480, which is the resolution of the RGBD-camera.

The output is an array of object detections. Each object consists of a bounding box, given by the location of its center and its size in pixels, and an array of probabilities, one probability for every object class.

6.4.3 Output Generation

The `VisionResult` message should only contain the probabilities of the relevant indoor place categories, also called room types. Using an array containing the information about what place categories are relevant, all irrelevant place categories are removed from the result of the place classification. The remaining probabilities are normalized to one and the resulting array is inserted into the `VisionResult` message.

The handling of the object detections is more complex. In a first step all probabilities of irrelevant types, which are stated in Table 5.5, are removed from the probability arrays of the object detections. Furthermore, all very unlikely detections, which are detections with no class having a probability greater than 0.00001, are also discarded because their impact is negligible. Then samples are drawn randomly from the valid points in the point cloud. The point cloud created by the openni2-package is organized like the corresponding image with rows and columns. Not for all pixels in the image depth measurements are available. Therefore, some points in the point cloud are marked as invalid and have to be avoided in the sampling procedure. However, the structure of the point cloud allows to find the pixels in the image corresponding to the samples. For every sample and every relevant object type the maximum probability of all overlapping object detections is assigned. The samples, consisting of a 3D point and a probability for every object type, are then inserted into the `VisionResult` message.

To generate the `HighConfidenceObjects` message, an iteration over all object detections is done. If the probability of the most likely object class of a detection is higher than 0.9, the detection is assumed to be correct. This value was found in tests to result in almost no false positives and is low enough for reliable detections having a higher probability. In this case the median z-coordinate, which is in the direction the camera is facing, of all points in the bounding box is calculated. This is a robust estimation of the distance to the object. The direction to the object is calculated using the center of the bounding box and the intrinsic parameters of the camera, which is then used together with the estimated distance to calculate the object location. This object location and the ID of the found object class are then inserted into the `HighConfidenceObjects` message.

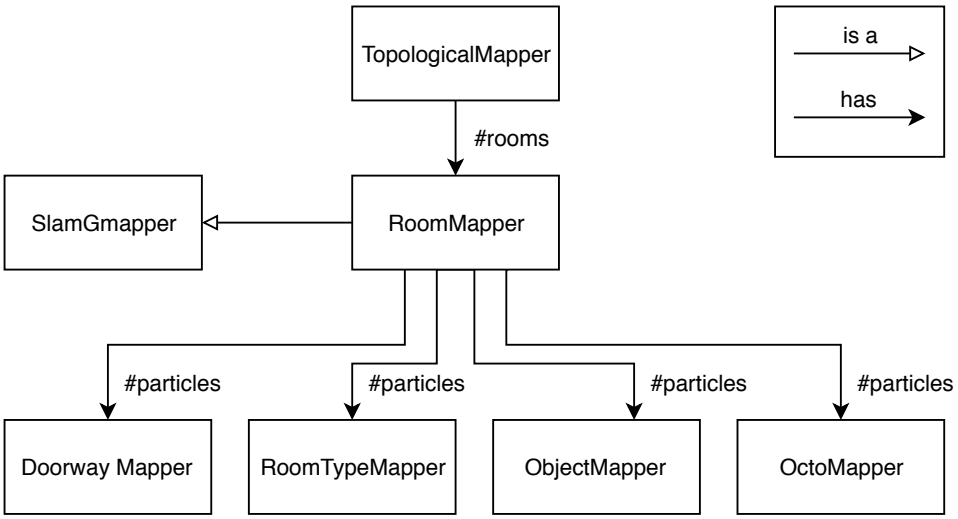


Figure 6.5: An overview over structure of the mapping node

Finally both generated messages are published.

6.5 Mapping Node

As shown in Figure 6.1 the mapping node has a central position in the system. It is also the largest node in terms of written code. This section is split into two parts: the class structure and the execution of the node.

6.5.1 Class Stucture

In Figure 6.5 the general structure of the mapping node is shown. The `TopologicalMapper` class is the main class in this structure, holding an instance of the `RoomMapper` class for every room. The `RoomMapper` class is derived from the `SlamGmapper` class, which does the 2D SLAM using GMapping. The `RoomMapper` class holds a separate mapping class for every type of map and every particle. This design was chosen to reduce the required adaptation to the existing code of GMapping and to have a more modular structure.

In Listing 6.4 the most important member variables and methods of the `TopologicalMapper` class are shown.

Listing 6.4: `TopologicalMapper` class

```
//members
RoomMapper[] room_mapper
int current_mapper
bool[] room_explored
topologicalMapSrv::Response topology
bool[] room_changed
int current_task

//methods
run()

pointcloudCallback(point_cloud)
laserCallback(laser_scan)
doorwayCallback(doorway_detections)
visionCallback(vision_result)
currentTaskCallback(current_task)
exploredCallback(explored_room)

topologicalMapServiceCallback()
objectMapServiceCallback()
```

Beside an array of `RoomMapper` it also contains variables to keep track of the current room and the rooms already explored. The latter is important in the generation of the high-level attributes in the `topologicalMapSrv` service. The variable `topology` stores the last `topologicalMapSrv` service response and `room_changed` keeps track of which rooms' maps were updated since the last execution of the `topologicalMapSrv` service. Together they are used to avoid unnecessary calculations on unchanged room maps. Also the current task is stored to decide if the doorways have to be blocked by virtual obstacles.

The `TopologicalMapper` class contains the `run` function, which is started during initialization and contains the main processing loop. The class also contains the callback functions for all messages and services, which are processed by the node. More details on the messages and services is given in Section 6.5.2.

The code in the ROS-node for GMapping [7] was the starting point for the implementation of the `SlamGmapper` class, whose most important member variables and methods are shown in Listing 6.5.

Listing 6.5: `SlamGmapper` class

```
//members
ParticleFilter particle_filter
OccupancyGrid map
Transform map_to_odometry_transform

//methods
processLaser(laser_scan)
```

This class is the interface to the GMapping library used for 2D SLAM. It contains an instance of the `ParticleFilter` class, which is derived from GMapping's particle filter class and has additional functionality for map switching and access to the particles implemented. The `SlamGmapper` class also stores the currently most likely map and the transformation between the map and the odometry coordinate frames. While the transformation is only updated when a laser scan is processed, the robot pose can still change based on the odometry data. The `processLaser` function is called to update the localization and the map.

The `RoomMapper` class is the main class for the mapping within a room and an overview over the most important members and methods is given in Listing 6.6.

[7] `https://github.com/ros-perception/slam_gmapping`

Listing 6.6: RoomMapper class

```
//members
OctoMapper[] octo_mapper
ObjectMapper[] object_mapper
RoomTypeMapper[] roomtype_mapper
DoorwayMapper[] doorway_mapper

//methods
activate(robot_pose, doorway)
deactivate()

processCloud(point_cloud)
processDoorway(doorway_detections)
processVision(vision_result)
```

The class inherits the 2D SLAM functionality from the SlamGmapper class and also contains the mappers for all the other information. It has functions for activation and deactivation, and also to update the different map types.

The OctoMapper class is based on the code of the octomap_server ROS-node[8]. It is the interface to the OctoMap library. In addition to the processCloud function, which is called to update the 3D map, it has a projectDown function, which projects the obstacles found in the 3D map into the 2D map.

Listing 6.7: OctoMapper class

```
//members
Octree octree

//methods
processCloud(point_cloud)
projectDown(map2D)
```

The RoomTypeMapper class contains 60 instances of the RoomTypeMap class, one for each room type. In Listing 6.8 the most important member variables

[8] https://github.com/OctoMap/octomap_mapping

and methods of the RoomTypeMapper class and the RoomTypeMap class are shown.

Listing 6.8: RoomTypeMapper class and nested RoomTypeMap class

```
//members
RoomTypeMap[] room_type_maps
   cv::Mat probability_map
   cv::Mat seen_count_map
   float resolution
   cv::Point origin

//methods
updateMaps(vision_result, particle_pose)
```

The RoomTypeMap class is implemented using OpenCV's cv::Mat to represent the two dimensional grid map. The probability_map contains the probabilities of the given room type for all cells in the map. The seen_count_map stores the number of updates of a cell. This information is important when estimating the object probability in unseen areas. Additionally, the resolution and the origin are stored, so transforming world coordinates to map coordinates and vice versa is possible. The updateMaps function is called to update the map.

The ObjectMapper class contains 61 instances of the ObjectMap class, one for each object type. In Listing 6.9 the most important member variables and methods of the ObjectMapper class and the ObjectMap class are shown.

Listing 6.9: `ObjectMapper` class and nested `ObjectMap` class

```
//members
ObjectMap object_maps
   cv::Mat[] probability_map
   cv::Mat[] seen_count_map
   float resolution
   cv::Point origin

   updateMap(vision_result)

//methods
updateMaps(vision_result, particle_pose)
```

The `ObjectMap` class is implemented using an array of OpenCV's `cv::Mat`. All cells at the same height are stored in one `cv::Mat` and those `cv::Mats` are stacked on top of each other to form the 3D grid. The `probability_map`, the `seen_count_map`, the resolution and the origin are similar to those in the `RoomTypeMap` class. The `updateMaps` function is called to update the maps. The `ObjectMap` class also has its own `updateMap` function because the maps for every object type can be updated independently.

The `DoorwayMapper` class holds an array containing all doorways found in this room. Besides the pose of the doorway the `Doorway` class contains an ID and the information about its corresponding doorway in the other room. This design to store connections between rooms is easy to maintain, though the access to the topological structure is more complicated. However, the topological structure is not needed frequently, only on map switches and `topologicalMapSrv` service requests. The `pose_array` member stores the last poses for the running average filter, which is used to calculate the `pose` of the doorway. The `processDoorway` function is called to update the doorway poses or insert a new doorway.

Listing 6.10: DoorwayMapper class and nested Doorway class

```
//members
Doorway[] doorways
   Pose pose
   Pose[] pose_array
   int id
   int counterpart_id
   int this_room
   int other_room

//methods
processDoorway(doorway_detections)
```

6.5.2 Execution

This node is implemented using multiple threads to distribute the computational load and to avoid that computational expensive tasks block the rest of the execution.

After the initialization the main thread calls the `run` function of the `TopologicalMapper`, which executes the main processing loop. The main processing loop is illustrated in Figure 6.6.

First, the correct callbacks for all pending messages are executed. Most of the code gets called this way. After all messages are processed, the 2D map for navigation is created and published. The last step in the loop is to check if a doorway was passed and, if necessary, execute the room map switch.

Message processing

Six types of message can be received, which are all first processed by the `TopologicalMapper`. The `current_high_level_task` messages and the `exploration_finished` messages are processed directly in the `TopologicalMapper`. The `current_high_level_task` messages are used to update the `current_task` variable, which is used to decide if the doorways

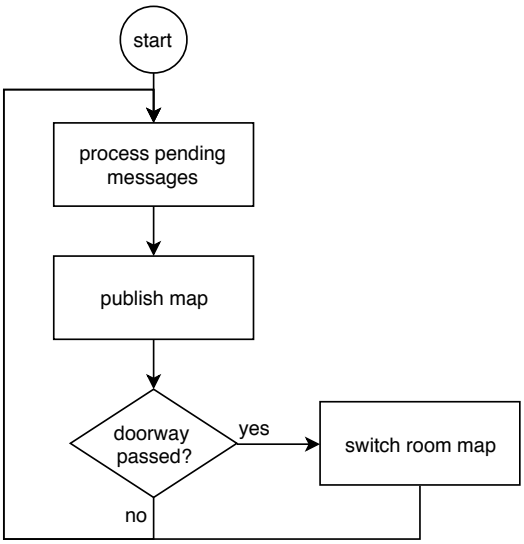


Figure 6.6: The main processing loop of the mapping node

have to be blocked by virtual obstacles. The `exploration_finished` messages are used to update the `room_explored` variable of the corresponding room, which is important in the generation of the high-level attributes in the `topologicalMapSrv` service. The other messages are forwarded to the `RoomMapper` of the current room.

The processing of the `laser_scan_filtered` message involves calling the `processLaser` function of the `RoomMapper` of the current room, which sends the laser scan to GMapping for localization and mapping. After the processing of a laser scan GMapping might do a resampling of the particles to discard unlikely particles. The discarded particles are replaced by more likely particles. When particles in the particle filter are replaced by GMapping also the corresponding mappers in the `RoomMapper` have to be replaced. The extended `ParticleFilter` class allows access to the result of the resampling. Using this result the correct replacement of the mappers is done. The transformation between the map coordinate frame of the best particle and the odometry frame is done in a separate thread to ensure regular publishing.

The 3D mapping is done in a separate thread because the 3D mapping takes

significantly time. Using a separate thread distributes the computational load on multiple cores of the processor and the other mapping is not slowed down. Therefore, the `processCloud` function of the `RoomMapper` of the current room only stores the `main_camera/pointcloud` message in a buffer. The thread for 3D mapping idles until a new message is in the buffer and executes the update of the 3D map. First all operations independent of the particle poses are executed. These are downsampling, as processing all points in the point cloud takes too long, the transformation of the point cloud from the camera coordinate frame into the robot coordinate frame, and splitting the point cloud into two point clouds, one for points near the ground and one for points at heights considered in the 3D map. Obstacles near the ground are not mapped, so the points near the ground have to be handled differently. For every particle these point clouds are then transformed into the map coordinate frames of the particle and the `processCloud` function of the corresponding `OctoMapper` is called. Using the point clouds and ray-tracing a list of cells containing points and a list of visible cells containing no points are created. The point cloud with points near the ground is only used for the latter. This way no obstacles are mapped near the ground. Then the 3D map is updated using the two lists. The ray-tracing and the map update are done using functions of the OctoMap library.

To achieve good results the transformation from the robot coordinate frame into the coordinate frame of the particles has to be accurate, especially the rotation. Particle poses are only known at times a processed laser scan was recorded, but for accurate mapping the particle pose at the time the point cloud was recorded is necessary. Therefore the particle pose at that time is estimated using the available particle pose nearest in time and the odometry data to estimate the movement of the robot in between.

In the callback for the `vision_result` messages the `processVision` function of the `RoomMapper` of the current room is called. For every particle the particle pose is calculated as described above and the `updateMaps` function of the corresponding `RoomTypeMapper` is called. There the map update is done according to Section 5.5.3. Also for every particle the `updateMaps` function of the corresponding `ObjectMapper` is called. In this function the `updateMap` functions of all `ObjectMap` instances are called. There the map update is done according to Section 5.5.2.

The doorway poses in the door_detections message are transformed into the map frames of all particles and the processDoorway functions of the corresponding DoorwayMapper instances are called. According to Section 5.5.4, the decision is made if it is a new doorway or another detection of an already found doorway. If it is a new doorway, a new instance of Doorway is added to the array of doorways. All elements in the pose_array and the pose variable of the new Doorway instance are set to the pose of the detected door, a new ID is generated for the doorway and the values of counterpart_id and other_room are set to -1. -1 signals that the doorway was never passed. If the detection is from an already found doorway, the oldest element in the pose_array is replaced by the pose of the doorway detection. Then the average of the poses in the pose_array is assigned to the pose variable. To calculate the average orientation the corresponding points on the unit circle are calculated for all orientations of the poses in the pose_array. The angle of the average of these points is used to calculate the average orientation.

Map creation and publishing

The creation of the map used for navigation and low-level planning is done as described in Section 5.5.6. As a second thread is working on the 3D map, locking is important to avoid race conditions, which can cause wrong results and program crashes. Additionally, if current_task is not zero, which corresponds to the move-through-doorway high-level task, the doorways are blocked. This is done by setting cells on a short line perpendicular to the doorway poses to occupied. This created map is then published.

Map switch

The last step in the main loop is the check if a doorway was passed. Therefore a test is executed on all doorways in the DoorwayMapper of the best particle. For this test the robot pose is transformed into the coordinate frame of the doorway. The first condition is that the x-coordinate of the transformed robot pose is greater than a threshold of 0.2 m, meaning the robot has

driven beyond the doorway. The second condition is that the distance to the doorway is smaller than second threshold of 1 m, which ensures the robot has really driven through the doorway. If both conditions are true, the currently checked doorway is passed and the map switch is triggered.

In Figure 6.7 the steps of the map switch are depicted. Firstly, the `deactivate` function of the `RoomMapper` of the current room is called. This function discards all particles except the best in the particle filter. Also the corresponding mappers of the discard particles are removed. Then the thread for publishing the map-to-odometry-transformation is paused.

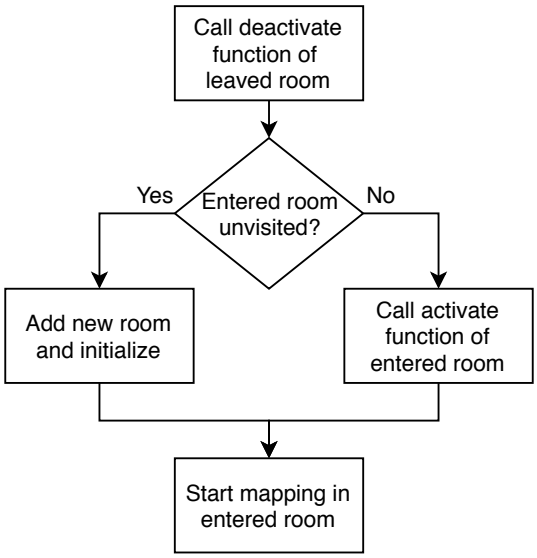


Figure 6.7: Steps when switching maps

If the `other_room` variable of the passed doorway is negative, a never visited room is entered and a new `RoomMapper` is created. The particle poses in the particle filter of the new `RoomMapper` are set to the pose of the best particle in the old room and also the map-to-odometry-transformation is copied to the new mapper, so the map coordinate frames of both rooms are initially the same. The pose of the passed doorway is flipped, so it is facing outwards in the new room, and added to the `DoorwayMappers` of the new `RoomMapper`. Then the new `RoomMapper` is added to the `room_mapper` array of the `TopologicalMapper`. The doorway in the old `RoomMapper` is updated,

so it contains the ID of the new room in the `other_room` variable and the ID of the passed doorway in the new room in the `counterpart_id` variable.

If the `other_room` variable of the doorway is not negative, it specifies the room to switch to. Using the ID in the `counterpart_id` variable of the doorway the corresponding doorway in the other room is found. As described in Section 5.5.6 the robot pose is transformed into the coordinate frame of the other room. With the transformed pose as parameter the `activate` function of the other `RoomMapper` is called. There the pose of the particle in the other room is set to this transformed pose and also the map-to-odometry-transformation is set accordingly. After that the particle and the mappers of the other rooms `RoomMapper` are duplicated, so the correct number of particles is available.

In both cases the next step is to set the `current_room` variable to the entered room and resume the thread for publishing the map-to-odometry-transformation in the entered room's `RoomMapper`. A message containing the transformation between the leaved and the entered room is then published as the last step.

Processing of Service Requests

The services requests are processed in a separate thread because otherwise the processing of the service requests would block the mapping for too long. This is especially true for the `TopologicalMapSrv` service, which can take more than one second.

In Listing 6.11 the request and response specifications of the `TopologicalMapSrv` service are shown. The request specifies the ID of the object of interest. Important to note is that references to the doorways in a room are stored in the rooms as well as the rooms connected by a doorway in the doorways. Also noteworthy is that the times to traverse a room are stored in the doorways. Therefore every doorway stores the times to travel to any doorway if the doorways are in the same room or -1 otherwise.

Listing 6.11: mapping/TopologicalMapSrv

```
# request
int16 object_id
---
# response
int16 current_room
RoomMsg[] rooms
   int16 id
   float32 object_probability
   int16[] doorways # doorways in the room
   float32[] to_doorway_times
   float32 search_time
   float32 expected_search_time
DoorwayMsg[] doorways
   int16 room1
   int16 room2
   geometry_msgs/Pose door1_pose # doorway pose in room 1
   geometry_msgs/Pose door2_pose # doorway pose in room 2
   float32[] traverse_times # distances to other doorways
```

When processing a service the first step is to copy the interesting data from the mapping structure. This way the mapping can proceed during the processing of the service request without creating race conditions. In a second step all maps of a room have to be brought into a common structure. Therefore, all types of maps are converted into an `ObjectMap` class. Furthermore, resampling is done to have the same resolution in all maps and the sizes of the maps are made equal by adding additional space to the maps if necessary. Now the calculation of the Object-Probability-Map is done for every room as described in Section 5.5.5. The response message is created according to Section 5.5.7, using the Object-Probability-Map and the other information stored in the `RoomMapper` class. If the result of a previous `TopologicalMapSrv` request exists, the calculations have only to be done for rooms marked as changed.

In Listing 6.12 the request and response specifications of the `ObjectProbabilityMapSrv` service are shown. The request specifies the room of interest and the searched object. The processing is done similar to the processing of the `TopologicalMapSrv` service, except only one room is

of interest and the response message can be created after the calculation of the Object-Probability-Map.

Listing 6.12: mapping/ObjectProbabilityMapSrv

```
int16 room_id
int16 object_id
---
ObjectMapMsg map
   int16 width
   int16 height
   int16 z_steps
   int16 origin_x # position of the origin in the map
   int16 origin_y
   float32 resolution
   float32[] data
```

6.6 High-Level-Planner Node

The high_level_planner node does the high-level control of the system, which also includes receiving search commands from the user. In Figure 6.8 an overview of the execution of a search is given. After an description of the data structures used in the implementation the individual steps are described in more detail.

6.6.1 Data Structures

The `TopologicalMap` class stores the information of the topological map in a form which is easier to use by the planner. For every room the object probabilities, the times to completely search the room, and the expected search times are stored. Furthermore, for every pair of rooms the time to travel from room A to room B, the rooms on the way from A to B, and the poses of the doorways on the way are stored.

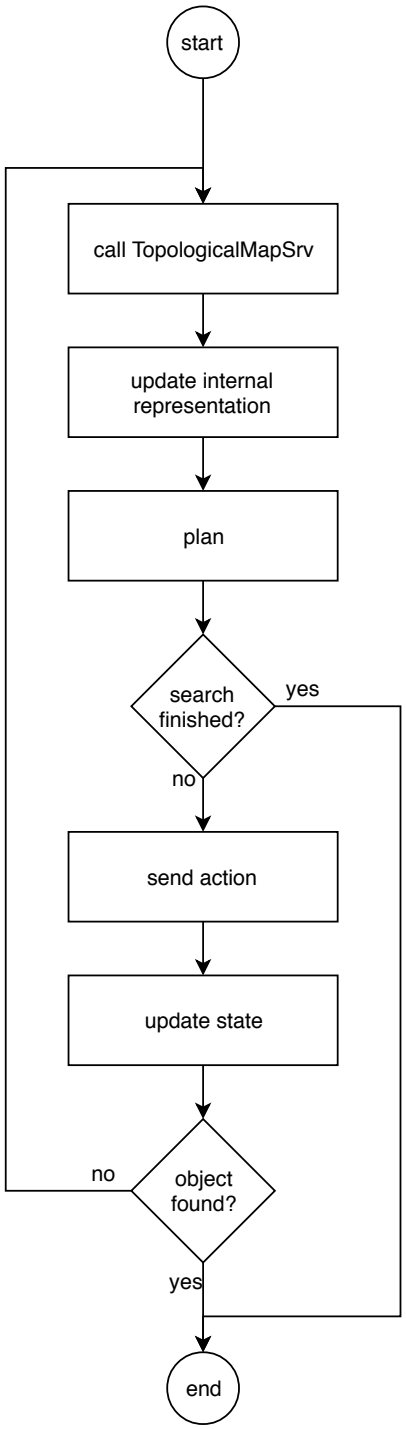


Figure 6.8: High-level planner execution overview

Listing 6.13: TopologicalMap

```
float[] object_probabilities
float[] search_times
float[] expected_search_times
float[][] travel_times
int[][][] travel_paths
Pose[][][] travel_waypoints
```

The `Action` class specifies the high-level tasks. Its variables contain the information needed for the execution of the task. The `type` variable specifies what high-level task to execute, the `target_object` and `target_room` variables contain the searched object and the current room. In case of the move-through-doorway task the next room is stored in the `target_room` variable and the `pose` variable contains the doorway pose.

Listing 6.14: Action

```
Type type
int target_object
int target_room
Pose pose
```

The `State` class represents the current state of the search, storing the current room and which rooms are already visited, explored or searched. The `getPossibleSearchActions` function returns a list of `Actions` with `type=SEARCH`, one `Action` for every not searched room.

Listing 6.15: State

```
int current_room
int[] searched
int[] explored
int[] visited

getPossibleSearchActions()
```

The `Plan` class contains the actions to be executed in the plan and also the `final_state` of the search after the correct execution of all actions in the path. This state is important for incomplete plans, which occur during the

planning, to know, which rooms are not searched in this path. Additionally, the probability of finding the object, the estimated time to fully execute the plan and the expected time of the search using this plan are stored in the `Plan` class. It has an `addAction` function, which adds an action to the plan and also updates the other member variables. The update of the `expected_search_time` is done according to Equation 5.30.

Listing 6.16: Plan

```
Action[] actions
State final_state
float expected_search_time
float full_search_time
float object_probability

addAction(Action, TopologicalMap)
```

6.6.2 Execution

The object search starts when the high-level planer gets the input of the object, which should be searched. If this is the first search in a session, there is only one room, which is set to not explored and not searched. Otherwise, all rooms are set to not searched, but the rest of the state is kept from previous searches because the mapping is not reseted in this case.

Update of the topological map

At the start of the search or after the execution of a high-level task a `TopologicalMapSrv` request is sent to the mapping node. Using the response of the service call, the internal representation is updated. This involves updating the data in the `TopologicalMap` instance and adding new rooms to the state if new rooms were discovered. The values in `object_probabilities`, `search_times` and `expected_search_times` can be copied from the response. To calculate the `travel_times`, `travel_paths` and `travel_waypoints` a graph is created, where each doorway is a node and doorways belonging to the same room are connected by an edge. The weight

of the edge, which is the time needed to drive from one doorway to the other, is given in the service response. Using the Floyd–Warshall algorithm the shortest paths and the length of the shortest paths are calculated for every pair of doorways. The `travel_waypoints` are the poses of the doorways on the shortest path, the `travel_path` contains the rooms between the doorways on the shortest path with the addition of the final room, and the `travel_times` are calculated by adding the correct `to_doorway_times` from the service response to the length of the shortest path. Those calculations are done for every pair of doorways, which is in further consequence also for every pair of different rooms. Pairs of the same room have `travel_times` of zero and empty paths.

Planning

The planning is done in three steps. In a first step a plan is generated using a greedy strategy. The second step is to execute a depth-first search on the tree of possible plans described in Section 5.6.1. The result of the greedy planning is used as a first estimate to enable effective pruning, which can reduce the planning time dramatically. From the result of the full planning, which only contains search actions, the next high-level task is generated.

The pseudo-code of the greedy planning is shown in Listing 6.17. First, an empty plan is created with the `final_state` being set to the current state. If the list of actions returned by the `getPossibleSearchActions` function is not empty, speed values for all actions in the list are calculated. The search speed value is the probability of finding the object in the corresponding room divided by the expected search time, which also includes the time to travel to the room to search. The search action with the highest search speed is added to the plan, using the `addAction` function described in Section 6.6.1. This is repeated until search actions for all not searched rooms are added to the plan, resulting in the greedy plan.

Listing 6.17: Greedy planning

```
Plan greedyPlan(TopologicalMap map, State state){
  Plan plan(state);
  Action[] possible_actions = plan.final_state.getPossibleSearchActions();
  while(!possible_actions.empty()){
    Action best_action;
    float max_speed = 0.0;
    for(action in possible_actions){
      float time = map.travel_times[plan.final_state.current_room][action.target_room]
                   + map.search_times[action.target_room];
      float speed = map.object_probabilities[action.target_room] / time;
      if(speed > max_speed){
        max_speed = speed;
        best_action = action;
      }
    }
    plan.addAction(best_action, map);
    possible_actions = plan.final_state.getPossibleSearchActions();
  }
  return plan;
}
```

The depth-first search of the search tree depicted in Figure 5.13 is implemented with a recursive function, which is shown in Listing 6.18. To start the depth-first search the `plan` function is called with the topological map, an empty plan, which is initialized with the current search state, and the expected search time of the greedy plan. If the list of actions returned by the `getPossibleSearchActions` function is empty, a leaf node is reached and the `old_plan` is returned. Otherwise new plans are created, which contain an additional search action. These plans are then evaluated further or discarded, if the expected search time is already higher than the `cutoff_time`. The best of those plans is then returned. If a complete plan, which is a plan where all rooms are searched in the `final_state`, is found, which has a lower expected search time than the `cutoff_time`, the `cutoff_time` is updated.

Listing 6.18: plan function

```
Plan plan(TopologicalMap map, Plan old_plan, float& cutoff_time){
  Action[] possible_actions = old_plan.final_state.getPossibleSearchActions();
  if(possible_actions.empty()){
    return old_plan;
  }

  SearchPlan best_plan;
  best_plan.expected_search_time = Infinity;
  for(action : possible_actions){
    SearchPlan new_plan = old_plan;
    new_plan.addAction(action, map);
    if(new_plan.expected_search_time >= cutoff_time){
      continue;
    }

    new_plan = plan(map, new_plan, cutoff_time);
    if(new_plan.expected_search_time < best_plan.expected_search_time){
      best_plan = new_plan;
      if(best_plan.expected_search_time < cutoff_time)
        cutoff_time = best_plan.expected_search_time;
    }
  }
  return best_plan;
}
```

The generated plan only contains the order in which the rooms are searched. If the plan is empty, the search is finished and no object was found. Otherwise, the generation of the high-level task is done according to Section 5.6.2, as shown in Listing 6.19.

Listing 6.19: generateNextTask function

```
Action generateNextTask(Plan plan, TopologicalMap map, State state){
  if(plan.actions.front.target_room != state.current_room)
    return Action(MOVE_TO, searched_object,
      map.travel_paths[state.current_room][plan.actions.front.target_room].front,
      map.travel_waypoints[state.current_room][plans.actions.front.target_room].front);
  else if(state.visited[state.current_room] == false)
    return Action(PEEK, searched_object, state.current_room, Pose());
  else if(state.explored[state.current_room] == false)
    return Action(EXPLORE, searched_object, state.current_room, Pose());
  else
    return Action(SEARCH, searched_object, state.current_room, Pose());
}
```

Executing the next high-level task

The generated next high-level task is sent to the low-level-planner in form of a `HighLevelTask` action, which is shown in Listing 6.20.

Listing 6.20: HighLevelTask

```
  # goal
  int16 type
  geometry_msgs/Pose pose
  int16 target_room
  int16 target_obj
  ---
  # result
  int16 result_number
  ---
  # feedback
```

The low-level-planner tries to execute the high-level tasks and sends back a result. Based on the result the state is updated. Possible results are:

- `SUCCEEDED`: The high-level task was successfully executed and nothing special happened. If a move-through-doorway task was executed, the entered room is set to visited. If an explore-room task was executed,

the current room is set to explored. If a search-room task was executed, the current room is set to searched.
- `ABORTED`: The high-level task could not be executed. In this case the state is not changed.
- `OBJECT_FOUND`: The high-level task was stopped because the object was found. In this case the state is not changed, but the search is finished.
- `NEW_DOORWAY_FOUND`: The high-level task was stopped because a new doorway was found. This result is only possible when executing an explore-room task. The state will be changed after receiving the new topological map.

6.7 Low-Level-Planner Node

The high_level_planner node sends `HighLevelTask` actions to the low_level_planner node, which tries to execute them and sends the result of the execution back to the high_level_planner. The execution of the robot movements is mainly done via the move_base package. The move_base takes goal poses and plans the robot controls to get to this goal poses, considering the map and sensor inputs for obstacle avoidance. If the goal is reached the move_base returns `SUCCEEDED` and if the goal could not be reached `ABORTED` is returned. Only during the peek task velocity commands are directly sent to the Turtlebot interface.

The node runs a loop, which is depicted in Figure 6.9. Which messages are processed depends on the currently active task, but the latest `map` message is always stored to have the most recent map available. Next, code dependent on the task, the received messages, and the result of the move_base is executed. This code involves the start of the task execution, the generation and sending of goal poses or velocity commands, and the sending of the results back to the high_level_planner. At the end of the loop the current action is published to inform the mapping node about the high-level task in execution. In the following sections the implementation of the individual tasks is described in more detail.

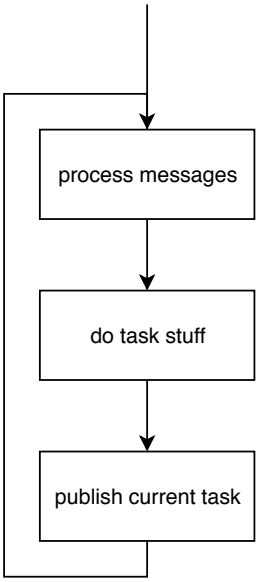


Figure 6.9: Overview over the execution of the low_level_planner node

6.7.1 Move Through Doorway Task

The steps of the execution of this task are already described in Section 5.7.1. The current state of the execution is stored in a state variable, which can either be `STARTED`, `FIRST_GOAL_REACHED`, `OTHER_ROOM_REACHED` or `FINISHED`. Depending on the state the goal pose is calculated and sent to the move_base. In Figure 6.10 the calculation of the goal poses is depicted. The obstacles in the map are inflated to get the area the robot can drive to. Depending on the state the goal is 0.5 m in front of the doorway (state = `STARTED`), 0.5 m behind the doorway (state = `FIRST_GOAL_REACHED`) or again 0.5 m in front of the doorway, but with flipped orientation (state = `REACHED_OTHER_ROOM`). If the location directly in front of the doorway is inaccessible, goal poses shifted to the side are tested until an accessible pose is found. A state change happens if the goal is reached or the map switch occurs. At any state change a new goal pose is calculated and sent to the move_base. Furthermore, every time a `dorway_poses` message arrives a new goal pose is calculated. This goal pose is only sent to the move_base if the goal has changed noteworthy. If the move_base fails to execute a movement, the task is aborted. Otherwise,

after reaching the final goal, SUCCEEDED is returned to the high_level_planner node.

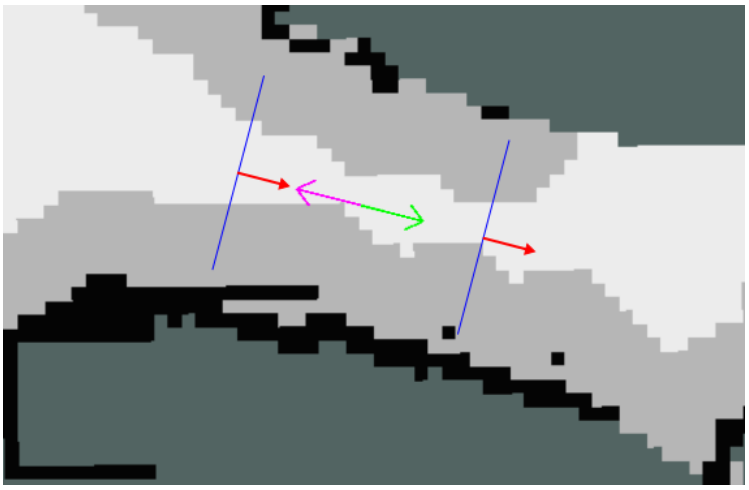


Figure 6.10: Goal calculation during drive-through-doorway task: The map and the accessible cells (white) are shown in the background. The doorway poses (green arrow before map switch, purple arrow afterwards) and the goal poses (red arrows) are shown. If the preferred goal pose is not accessible, an accessible pose on the blue lines is chosen.

6.7.2 Peek Task

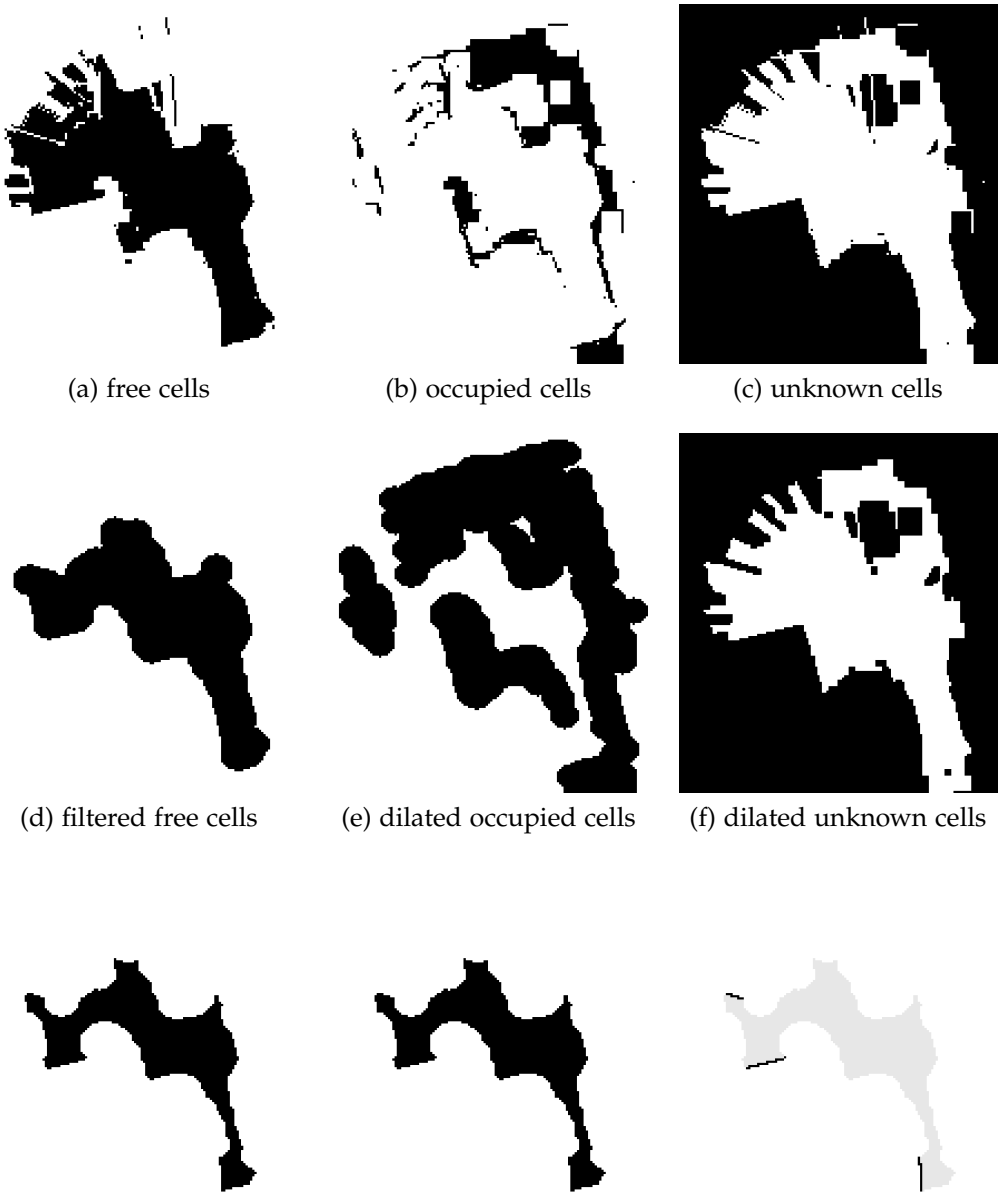
In this task velocity commands are sent directly to the Turtlebot interface because collisions are nearly impossible during the in-place rotations executed in this task. At the start of the task the current orientation of the robot is stored. Then a velocity command with zero translational velocity and a low negative rotational velocity is sent repetitively to the robot until a rotation of -90° is done. This is repeated with a positive angular velocity until a rotation of 90° compared to the starting orientation is reached. With an again negative angular velocity the robot is rotated back to the starting position and SUCCEEDED is reported to the high-level planner. During the execution high_confidence_objects messages can be received. If a confident detection of a searched object is received, the task is stopped and OBJECT_FOUND is reported to the high-level planner.

6.7.3 Explore Room Task

Four events can trigger a new goal pose during the execution of the exploration task. The first is reaching the current goal pose. The second trigger is the reception of a new map. In this case a new goal pose is calculated. If the new pose differs significantly from the old goal pose, the goal is sent to the move_base. The third trigger is a failed movement execution. In this case a new goal pose is calculated with exclusion of poses similar to the failed goal pose and sent to the move_base. The fourth event is triggered if the robot has not moved for a specified time. In this case a problem in the path planning is assumed and a new goal is calculated like in the previous case.

The calculation of the next goal pose is done according to Section 5.7.3. The calculation of the frontiers is shown in Figure 6.11. At first three `cv::Mat`-images are created from the map, one containing the free cells, one the occupied cells and one the unknown cells. The image of the free cells is filtered by the morphological operation opening with a circular kernel. The diameter of the kernel is the robot diameter. The image of the occupied cells is dilated using the same kernel. The image of the unknown cells is dilated with a 3x3 kernel, to get an overlap with the free cells. The not-forbidden cells are the filtered free cells minus the dilated occupied cells. Not-forbidden cells are valid locations for the robot. However, these cells need not be accessible because obstacles could block the way to some of those cells. Therefore a flood fill algorithm is executed on the image of the not-forbidden cells with the current robot location being the starting pixel for the flood fill. This results in the cells accessible by the robot, which are identical to the not-forbidden cells in the example shown in Figure 6.11. The frontiers are then the intersection of the dilated unknown cells and the accessible cells.

The goal pose is not set to the nearest frontier. Instead poses near the frontiers are considered, which are depicted in Figure 6.12. This is done to avoid goal poses at the border of the accessible space, which have a higher probability for a failed navigation. Additionally a penalty distance is added to cells near the border of accessible space, so those cells are only chosen as goal locations if necessary. The resulting goal pose is the location of the



(a) free cells (b) occupied cells (c) unknown cells

(d) filtered free cells (e) dilated occupied cells (f) dilated unknown cells

(g) not-forbidden cells (h) accessible cells (i) frontiers

Figure 6.11: Calculation of the frontiers: In those binary images black is 1 and white is 0; the gray accessible area in the last image is just for clarity

nearest possible goal cell, including the penalty, with an orientation towards the frontier.

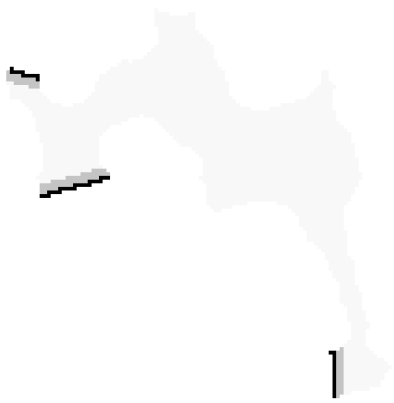


Figure 6.12: Possible goal cells are shown in dark gray, accessible cells in light gray and frontiers in black

A special case are goal poses within the robots footprint. This can happen because no clearing of the footprint is done in the mapping. In this case the goal is set to a pose outside the footprint facing towards the frontier.

During the execution, `high_confidence_objects` messages can be received. If a confident detection of a searched object is received, the task is stopped and `OBJECT_FOUND` is reported to the high-level planner. Also `doorwy_found` messages can be received. If a `doorwy_found` message is received, the exploration is stopped and `NEW_DOORWAY_FOUND` is reported to the high-level planner.

6.7.4 Search Room Task

The search room task is implemented using the `Searcher` class, which contains and maintains the data needed for the search planning. In Listing

6.21 the most important member variables and the callback functions for the necessary messages are shown.

Listing 6.21: Searcher

```
ObjectMap object_map
OctoMapper octo_mapper
ObjectMap mapping_object_map
cv::Mat accessible_map
cv::Mat border_map
cv::Mat direction_map
cv::Mat[] previous_pose_maps
cv::Mat[] seen_maps
cv::Mat not_fully_viewed_border_map

mapCallback()
visionResultCallback()
```

The `object_map` and the `octo_mapper` variables are used for the creation of an Object-Map and a 3D map like in the mapping node. The same classes are used, except the `OctoMapper` class has an additional value per node for counting the number of updates. The mapping has only to be done for the robot pose, instead of all particles, as the localization is assumed to be accurate enough after the exploration. Furthermore, only the Object-Map for the searched object type has to be created. This means that more samples from the object detection can be used without slowing down the node. Instead of using the point cloud from the camera the samples from the `VisionResult` message are used as input for the 3D mapping. When a `VisionResult` message arrives the two maps are updated. After the update the object probability is calculated for each cell and if it is greater than a threshold, the object is found. In this case `OBJECT_FOUND` is reported to the high-level planner and the search is stopped.

The `mapping_object_map` variable stores the response of the last `ObjectProbabilityMapSrv` service call. Storing the response speeds up the pose calculation because the service can be skipped for some pose calculations until it is done again.

The `accesible_map` stores the area accessible by the robot. It is calculated

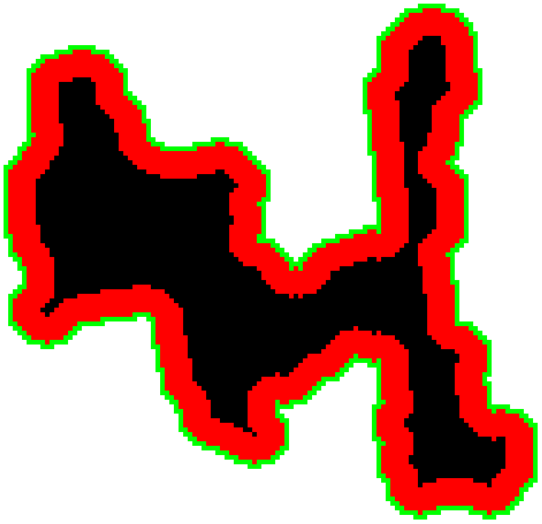


Figure 6.13: Border calculation: In black is the `accesible_map`, in the red the area, which is added by the first dilation and the green area is the border, which is added by the dilation with the 3x3 kernel

like in the exploration task. The `border_map` stores the border described in Section 5.7.4. To get a smoother result a dilated version of the `accesible_map` is created. This dilated map is further dilated with a 3x3 kernel and the less dilated image is subtracted. The result is the border. The operations are shown in Figure 6.13. The `direction_map` contains the directions away from the accessible area for each cell, as described in Section 5.7.4. To calculate this map a distance transform is applied on a dilated version of the `accesible_map`. The gradient of the distance transform result is calculated by filtering the image with Sobel kernels, once to calculate the x-derivative and once to calculate the y-derivative. The direction of the gradient is the `direction_map`. The three maps described in this paragraph are calculated when new `map` messages are received.

The remaining maps are updated every time a `VisionResult` message is processed. The `previous_pose_maps` store the poses at which an image, whose `VisionResult` was used, was taken. The 360° are split into sectors and for each sector a map is stored in the `previous_pose_maps` array. If a

VisionResult message is processed, the time stamp of the message, which is also the time stamp of the image, is used to get the robot pose at that time from the tf-library. The corresponding cell in the previous_pose_maps is then set to one. The seen_maps array has the same structure, one 2D map for each sector. It stores which cells were seen from which direction. This is important for the decision if a cell on the border was seen well enough. The cells within the view cone and with a distance to the robot of less than 2.2 m are determined. The reason to limit the distance at 2.2 m instead of 3.5 m, which is the specified maximum range of the RGBD-camera, is that not the border is of interest, but the area behind the border. In the seen maps of the corresponding sector and its neighboring sectors the values in the determined cells are incremented. The neighboring sectors are also updated, so the angle constraint is not that strict. The not_fully_viewed_border_map is then created based on the border_map, the direction_map and the seen_maps. It contains border cells, which were not seen well enough. For each cell set in the border_map the direction in the direction_map is used to find the correct cell in the seen_maps. If the value of the cell in the seen_maps is lower than a threshold, the cell is set in the not_fully_viewed_border_map.

The calculation of a new goal pose is triggered by three events. The standard trigger is that a VisionResult message was received from the last goal pose. This design ensures that a view is done at the calculated pose. The other two are the same as in the exploration task, the failed movement execution and no movement for some time.

In Listing 6.22 the general execution of the calculation of the next pose is shown. The get2DProbabilityMap function, which calculates the 2D probability map, is described below. For every sector in the previous_pose_maps and every cell in the grid map a view pose is generated, where the position is the center of the cell and the orientation the angle in the middle of the sector. If the cell is accessible and the view pose was never visited, the view pose is evaluated by the calculateViewValue function, which is also described below. The getPose function calculates then the goal the robot should drive to from the best evaluated cell. If no pose has a value greater than 0, the room is completely searched, SUCCEEDED is reported to the high-level planner and the task is stopped. Otherwise, the goal is sent to the move_base.

Listing 6.22: Calculation of the next pose

```
prob_map = get2DProbabilityMap();
float best_value = 0.0;
Pose best_pose;
for(s in sectors){
  for(cell in map){
    if(accessible_map(cell) and not previous_pose_maps[s](cell)){
      float value = calculateViewValue(cell);
      if(value > best_value){
        best_value = value;
        best_pose = getPose(cell, sector);
      }
    }
  }
}
if(best_value > 0.0)
  sendGoal(best_pose);
```

The calculation of the 2D probability map can be seen in Listing 6.24. The `getMapFromMapping` function, which calls the `ObjectProbabilityMapSrv` service, is only executed every fifth time a new pose is calculated. The 3D probability grid map is then collapsed to 2D as can be seen in the for-loop. The line `if(octo_mapper.getUpdateCount(x,y,z) < threshold)` checks if the number of updates of a cell in the 3D map is higher than a threshold value, which was set to 20 after tests. If this is true, the probability for the object being in that cell is 0 because the cell was seen often enough to make the assumption that the object is not there.

Listing 6.23: get2DProbabilityMap()

```
cv::Mat get2DProbabilityMap(){
  if(goal_calculation_counter%5 == 0)
    mapping_object_map = getMapFromMapping();
  goal_calculation_counter++;

  for((x,y) in map){
    float prob = 1.0;
    for(z=0; z<object_map.z_steps; z++){
      if(octo_mapper.getUpdateCount(x,y,z) < threshold){
        prob = prob*(1-mapping_object_map.getProbability(x,y,z));
      }
    }
    2Dmap(x,y) = 1-prob;
  }
  return 2Dmap(x,y);
}
```

The `calculateViewValue` can be seen in Listing 6.24. The probability of detecting an object `P` is calculated according to Equation 5.33 and the term for coverage `B` is calculated according to Equation 5.35. The variable `seenKernelValues` contains pre-calculated values for the probability of detecting an object in a cell according to Figure 5.19 and the variable `seenKernelPoints` is a list of offsets, where those are non-zero. The function `getVisibilityProb` implements the calculation of the probability of an object being visible according to Equation 5.34. The time to get to the view pose `T` is calculated according to Equation 5.32 and the return value according to Equation 5.31.

Listing 6.24: calculateViewValue()

```
float calculateViewValue(cv::Point cell, int sector, cv::Mat prob_map,
                         Pose current_pose){
  float B = 0.0, P = 1.0;
  for(int i=0; i<seenKernelPoints.size(); i++){
    cv::Point offset = seenKernelPoints[sector][i];
    cv::Point c = cell+offset;
    P = P * (1-getVisibilityProb(sector,offset)*
               seenKernelValues[sector][i]*prob_map(c));
    B = B + V_B*not_fully_viewed_border_map(c);
  }
  P = 1-P;
  if(B == 0.0)
    return -Infinity;

  float T = calculateTime(getPose(cell, sector), current_pose);
  return (P+B)/T;
}
```

7 Evaluation

In this chapter we present an evaluation of the active object search system described in this thesis. A detailed and meaningful quantitative evaluation of the complete system is very demanding for multiple reasons. The test conditions, like the environment, the searched object type, the object location and the starting pose, have a huge impact on the search time and the utility of an intelligent behavior. The result is also influenced by random factors, like measurement noise, inaccuracies of the robot movement, and timing of actions, which can lead to very different results even with similar settings. Furthermore, test runs can easily take more than half an hour, so extensive testing needs time. Therefore this evaluation is mostly done qualitatively by investigating representative runs and discussing special cases.

The evaluation is split into three parts. The first part covers the basic functionality, where the results of the computer vision and the performance of the navigation are discussed. In the second part the quality of the generated maps is investigated. The last part covers the performance of the actual object search.

Three environments were available for testing. Environment 1 is a home environment in a small flat, consisting of three rooms. Environment 2 is an atypical university environment consisting of some offices, and a robotics lab. Environment 3 is a part of an university floor with rooms of various types. A more detailed description of the environments can be found in Section 7.2.1. For a first testing of the active object search system, environments with more space for navigation and less clutter would be better to reduce the influence of disturbing factors, like navigation problems, on the test results. As such environments were not available the robot has to cope with the difficulties and these have to be considered in the results. Some minor modifications were made to the environments, like fencing off obstacles

the robot cannot detect neither with the LIDAR nor with the camera and dangerous areas like stairs.

7.1 Basic Functionality

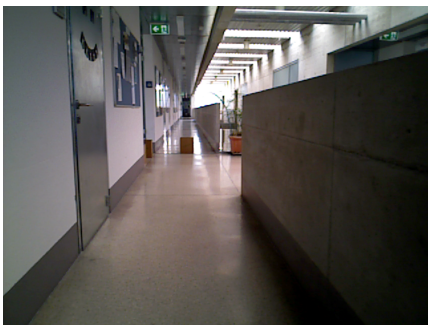
In this section an overview is given of how well the basic components of the developed system work in practice. This should help to get a better understanding for the results presented in the following sections.

7.1.1 Computer Vision

Room type classification

In most cases the room type classifier returns reasonable results. If there is a clearly correct room type, the probability for this room type in the result of the room type classifier is high, as can be seen in Figure 7.1.

(a) living room: 0.708, parlor: 0.239

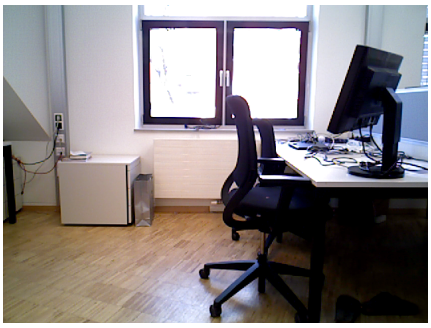


(b) corridor: 0.956

Figure 7.1: Images with a high probability for the correct room type in the room type classification result; the result of the room type classifier is shown in the caption (only for room types with a significant probability)

In many cases multiple similar room types, e.g. kitchen and kitchenette, are fitting. In this case most of the probability in the room type classification result is split between the fitting room types, as can be seen in Figure 7.2.

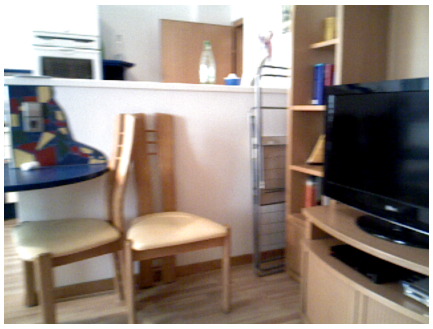
(a) kitchen: 0.579, kitchenette: 0.399



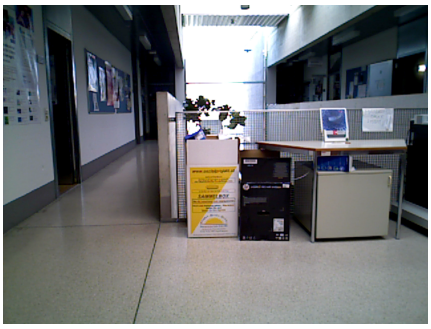
(b) office: 0.561, home office: 0.424

Figure 7.2: Images, where most of the probability in the room type classification result is split between two fitting types; the result of the room type classifier is shown in the caption (only for room types with a significant probability)

In some images the room type is not clear. In this case multiple different room types have a significant probability in the result of the place classifier and some of those are not fitting. In the image on the left of Figure 7.3 a living room area with a kitchen area in the background is show. The result of the room type classifier does not reflect this. In the image on the right the room types office and corridor are reasonable, but the other room types which have a significant probability, are wrong.



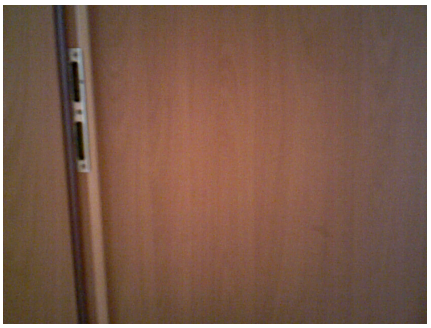
(a) office: 0.242, home office: 0.240, pantry: 0.104, dorm room: 0.083, kitchen: 0.072, kitchenette: 0.062



(b) office: 0.347, reception: 0.178, corridor: 0.126, art gallery: 0.120, lobby: 0.067, art studio: 0.063

Figure 7.3: Images with no clear room type; the result of the room type classifier is shown in the caption (only for room types with a significant probability)

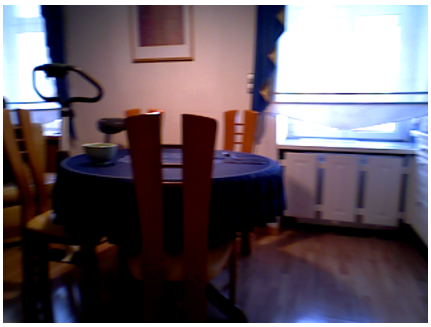
The place classifier has obviously problems when the robot is in front of a wall or another object which covers the whole image, as can be seen in Figure 7.4. In this case the room types closet, corridor, shower and basement have usually a high probability in the room type classification result.



(a) closet: 0.535, corridor: 0.206, shower: 0.092

Figure 7.4: Image taken directly in front of a wardrobe; the result of the room type classifier is shown in the caption (only for room types with a significant probability)

(a) Image showing a dine room area and a kitchen area: kitchen: 0.459, kitchenette: 0.397, dine room: 0.060



(b) Image showing only a dine room area: dine room: 0.591, kitchen: 0.066

Figure 7.5: Image showing two areas of different types and comparison image; the result of the room type classifier is shown in the caption (only for room types with a significant probability)

If multiple areas of different room types are visible in one image, this is most of the time not reflected in the result of the place classifier because the more prominent room type overrules the other one. This can be seen in Figure 7.5: on the left, where also a kitchen area is visible, the probability for dine room is low, whereas in the right image, where no kitchen area is visible, the probability for dine room is high.

(a) dark and noisy image:
kitchen: 0.481, kitchenette: 0.151,
restaurant kitchen: 0.115

(b) comparison image:
kitchen: 0.579, kitchenette: 0.399

(c) blurry image:
kitchen: 0.594, kitchenette: 0.355

(d) comparison image:
kitchen: 0.511, kitchenette: 0.441

Figure 7.6: Impact of bad lighting, noise and motion blur on the place classifier; the result of the room type classifier is shown in the caption (only for room types with a significant probability)

The place classifier is quiet robust against bad lighting and noise in the image, as can be seen by comparing image (a) and (b) in Figure 7.6. Also blurry images caused by the robot movement have no big impact, as can be

(a) The mouse takes little space in the image, but was still detected with high confidence

(b) The cutlery is perfectly visible, but only the spoon was detected correctly

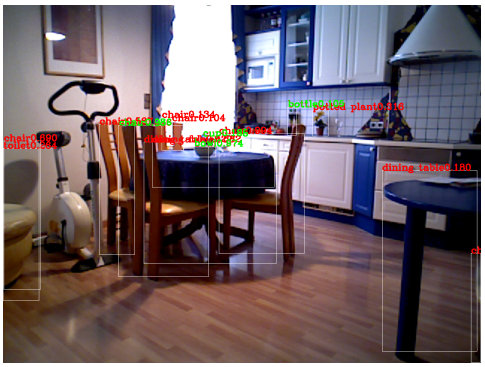
Figure 7.7: Detected objects (given by bounding box, type and confidence) with a confidence greater than 0.1

seen by comparing image (c) and (d) in Figure 7.6.

Object detection

A quantitative evaluation of the used YOLOv3 object detection CNN can be found in the related paper [19]. There can be seen that the object detection works better for large objects than for small ones. This trend was also observed during the test runs. During tests was found that not all small objects are equally difficult to detect for the CNN. E.g. remotes and especially mouses were detected most of the time with high confidence, even from some distance, as can be seen in image (a) of Figure 7.7. On the other hand other small objects, e.g. spoons, knifes and forks, were only detected when the robot was right in front, as shown in image (b) of Figure 7.7, and then only sometimes with high confidence.

The confidence of the detections was found to vary a lot, like in image (a) of Figure 7.8. Also some completely wrong detections were encountered, e.g the refrigerator in image (b) of Figure 7.8.



(a) High confidence for the chairs in front and the bowl, low confidences for all other objects; the wine glass on the right of the potted plant was not detected at all

(b) Refrigerator detected at wall

Figure 7.8: Detected objects (given by bounding box, type and confidence) with a confidence greater than 0.1

The performance of the object detection drops significantly with bad lighting, as can be seen in Figure 7.9. This is especially a problem because the sensitivity of the used RGBD-camera is not very high and the dynamic range is low.

The performance of the object detection also drops with motion blur, as can be seen in Figure 7.10. This, in combination with the low sensitivity of the camera and the therefore necessary high exposure time, made the limitation of the rotational velocity during movement necessary.

In general can be said that the object detection is sufficient if no objects, which are difficult to detect, are searched. However, a better object detection would make the problem easier and the object search more reliable.

Doorway detection

During the nearly 50 test runs with enabled doorway detection done for the evaluation, more than 100 doorways were correctly inserted into the maps with no false negatives and only one false positives. Also during the

(a) Many detections with good lighting

(b) Few and wrong detections with bad lighting

Figure 7.9: Detected objects (given by bounding box, type and confidence) with a confidence greater than 0.1

(a) Many detections without motion blur

(b) Few detections with motion blur

Figure 7.10: Detected objects (given by bounding box, type and confidence) with a confidence greater than 0.1

development process only one false positive was encountered after good parameters were found.

7.1.2 Navigation

Most of the navigation goals are reached without difficulties or major delays. The global planning in the 2D map of the room works without problems. However the local planner and the robot movement cause some problems. The high payload brings the robot to its limits. On even and clean floors the robot moves fine, but on slippery or uneven floors the robot has not enough grip on the active wheels to execute the movements as planned because too much weight is on the passive wheels. Due to the robot's design, with the passive wheels being fixed in forward direction, rotational movements are more difficult to execute for the robot. Thus, the problem with slipping wheels only affects the rotational component of the movement. The result is that rotations are slower, curves are driven wider as planned and changes in the direction are delayed. In open areas this does not cause problems but in narrow areas the local planner sometimes fails to keep enough distance between the robot and the obstacles. In this case the planner fails to find a valid path because every path is assumed to cause a collision. The robot can also get into this situation if it drifts towards an obstacle during an in-place rotation which happens if only one active wheel is slipping. The recovery behaviors of the `move_base` and short backward movements are usually sufficient to recover from this situation, but these take some time. This is a major factor for the varying search times for the same test scenario. The local planner has in general a big impact on the search time. The smoother it can navigate the robot through narrow areas and around obstacles, the faster the search will be.

7.2 Mapping

The quality of the maps is investigated after a complete exploration of the environment, as the goal of the exploration is to gather information for an efficient object search. Therefore, a modified high-level planner is used which does not generate search tasks. This was done in all three environments multiple times. In this section typical results of those exploration runs are presented and also the encountered problems are described. At first the

metric maps are investigated, followed by an investigation of the Room-Type-Maps and the Object-Probability-Maps. At the end of this section the estimations after peeking into a room are presented.

7.2.1 Hybrid Metric Map

In Figure 7.11 - 7.13 maps automatically created in the three test environments are shown. The 2D maps of the rooms have in general a good quality, especially for small rooms. In all maps some cells outside the room were set to free. This is due to invalid range measurements which are handled as described in Section 6.2. In general these free cells have no significant negative impact. In larger rooms the quality of the maps decreases, which can be seen in the map of room *lab1*. There the top part leans to the left while the bottom right part is rotated towards the bottom left part. Another example can be seen in the map of the room *institute1*, where the bottom corridor is curved upwards. The reason for these distortions is probably the limited range of the LIDAR, which results in many invalid range measurements in open areas. In further consequence less distinctive features are seen by the LIDAR and so the localization of the robot has to rely on the odometry, which can lead to inaccurate localization and therefore also to inaccurate maps. The larger the rooms are, the bigger is the impact of those inaccuracies. This shows the benefit of the hybrid map design which splits larger environments into easier to handle room maps.

The doorway mapping works nicely in all explored environments, as can be seen in the hybrid maps in Figure 7.11 - 7.13. The doorway positions and orientations are very accurate in almost all cases, except for the doorway between rooms *lab1* and *lab3*. There a door closer mounted on top of the door confused the doorway detection and was responsible for the doorway detections being not in the center of the doorway. In this case the doorway pose is still within the doorway and as the shift is the same in both rooms, it causes no problems.

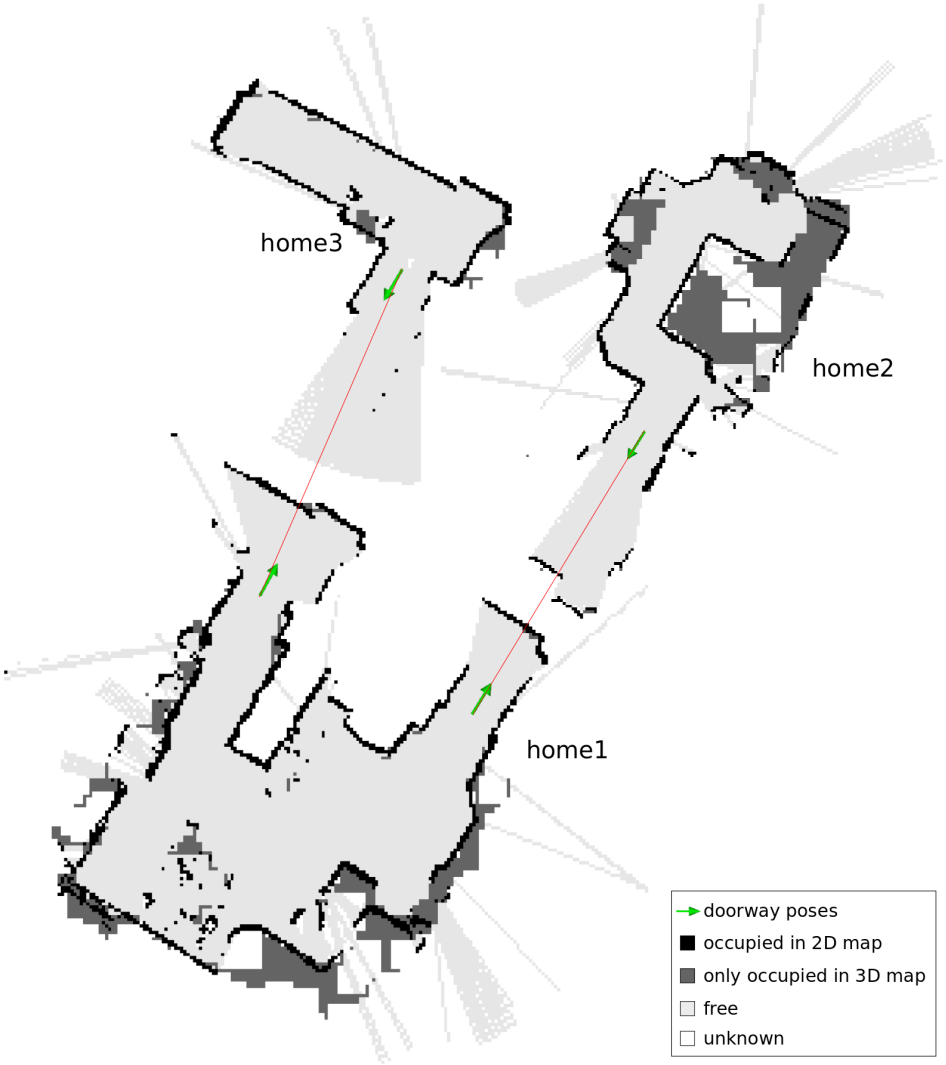


Figure 7.11: Hybrid map of the home environment: Room *home1* is a combination of a kitchen and a living room, room *home2* is a bedroom with an office-like area in the upper right corner and room *home3* is an anteroom. The topological map consists of the three rooms and the two edges represented by the red lines between the doorway poses.

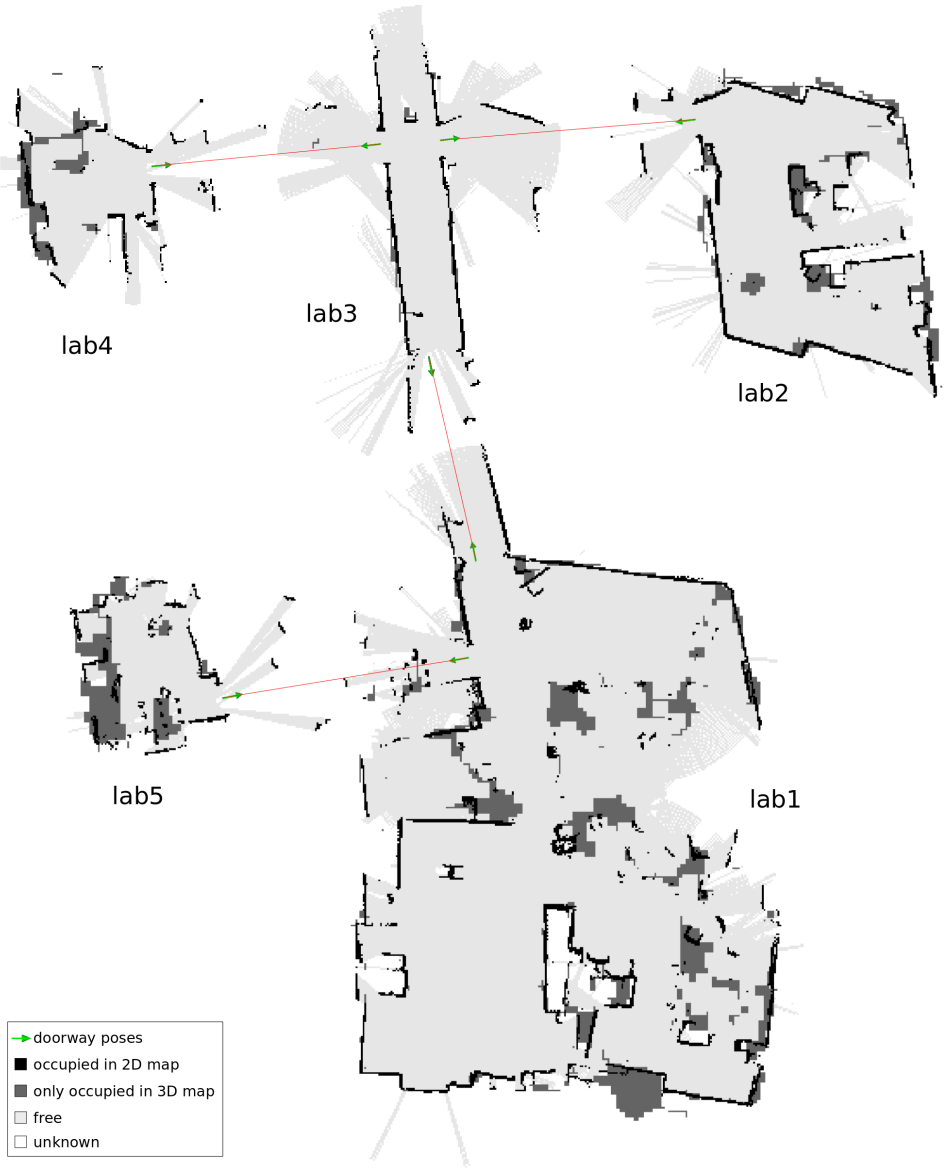


Figure 7.12: Hybrid map of robotics-lab environment: Room *lab1* is the robotics-lab which consists mainly of an office area. It has free space at the bottom left and a sofa and a kitchen area at the bottom right. Room *lab2* is an office, room *lab3* is a corridor, room *lab4* is also an office and room *lab5* is a workshop/storage room. The topological map consists of the five rooms and the four edges represented by the red lines between the doorway poses.

Figure 7.13: Hybrid map of a part of an university floor: Room *institute1* is a corridor with a copier in the open area on the left, room *institute2* consists of a kitchen area and a dining table, room *institute3* is a meeting room, room *institute4* is a bathroom and room *institute5* is a secretariat. The topological map consists of the five rooms and the four edges represented by the red lines between the doorway poses.

7.2.2 Room Type Map

The room *home1* was chosen for the evaluation of the quality of the Room-Type-Maps because in this room areas of different room types exist. Assigning a ground truth Room-Type-Map is only possible to a limited extend, because for some areas multiple room types are fitting and the boundary between areas of different room types is even hard to judge for humans. Instead of a somewhat arbitrary ground truth, images covering the whole room are used in the evaluation as comparison for the generated maps. These images are shown in Figure 7.14. In Figure 7.15 the poses, where the images were taken, are depicted and also the room types of the main areas are marked.

In Figure 7.16 the most likely room types for different runs and settings are shown. In general the main areas of the room are correctly classified, though the borders of the areas vary a lot for different runs and settings. After the exploration in run 1, the kitchen area, the dining table and the living room area are correctly classified. The areas near the doorways are classified as closet, probably because of the wooden door frames. The area around the TV is classified as dine room, which is definitely not correct. However, this area was only seen at the edge of a few images which were mostly classified as dine room.

In image (b) of Figure 7.16 the Room-Type-Map created without using the correlation between room types described at the end of Section 5.5.3 is shown. On the one hand the dining room area is much larger there. This is caused by the uncertainty of the room type classification between the similar room types kitchen and kitchenette. Without considering this similarity a third room type, in this case the type dine room, becomes more likely. On the other hand the closet area is much larger because it is not penalized for not really fitting to the other room types. In general the impact of using the correlation between room types is not that big, but the tests showed at least a small positive effect.

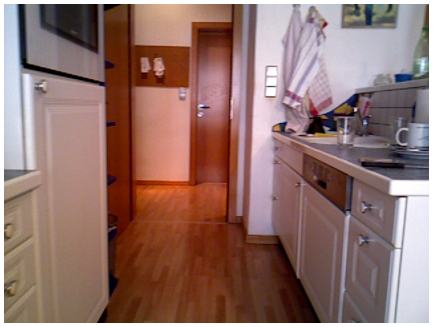
In the images (c) and (d) of Figure 7.16 the resulting Room-Type-Maps for a second and third run are shown. After run 2 the living room area is much larger than after run 1, whereas in run 3 the kitchen and kitchenette area is much larger. This shows the impact of the trajectory during the

(1) kitchen: 0.414, kitchenette: 0.401, dinette: 0.087, dine room: 0.077

(2) kitchenette: 0.468, kitchen: 0.245, living room: 0.066, galley: 0.053

(3) kitchen: 0.444, kitchenette: 0.418

(4) living room: 0.708, parlor: 0.239



(5) kitchen: 0.481, kitchenette: 0.297, galley: 0.205

(6) closet: 0.243, pantry: 0.144, living room: 0.092, basement: 0.076, lobby: 0.069, reception: 0.060, kitchenette: 0.052

Figure 7.14: Images of the room *home1* with place classification results

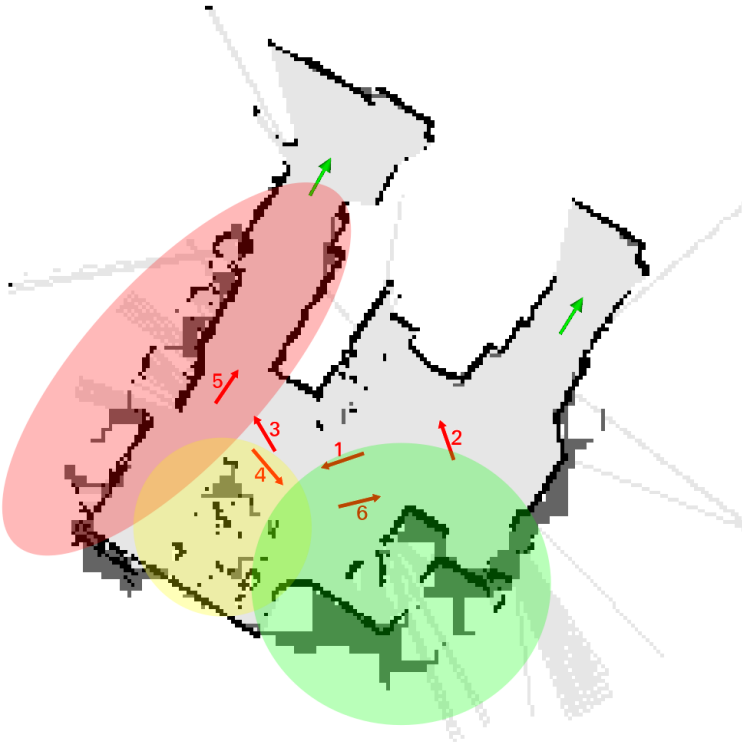
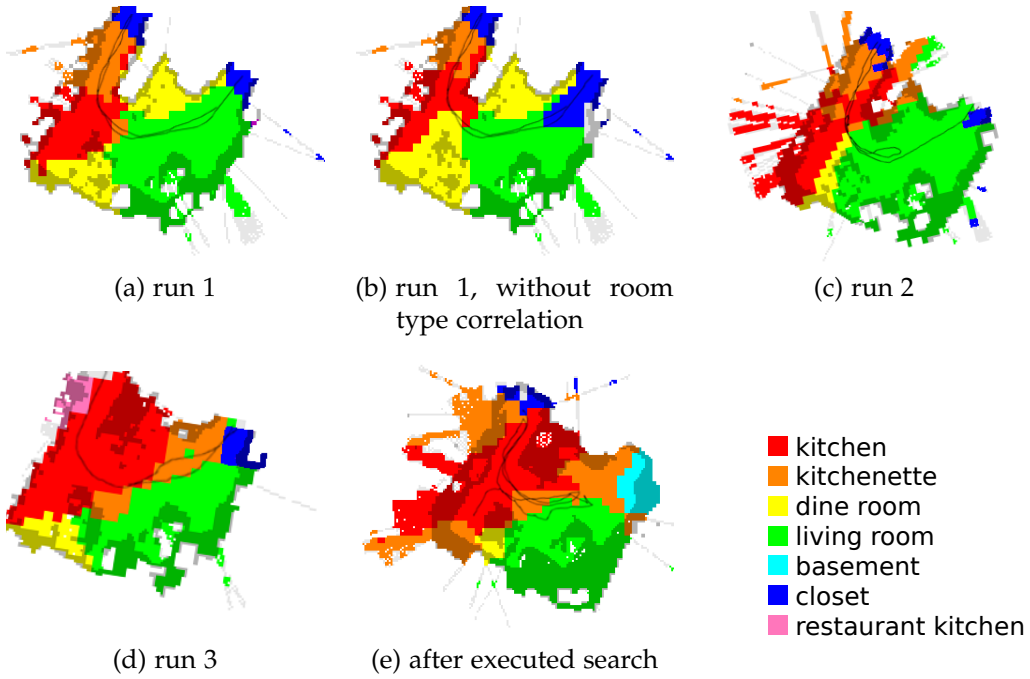


Figure 7.15: Detailed map of the room *home1*: Green arrows mark the doorways and the numbered red arrows are the view poses of the corresponding images in Figure 7.14. The red area is a kitchen area, the yellow area is a dining table and the green area is a living room.



(a) run 1

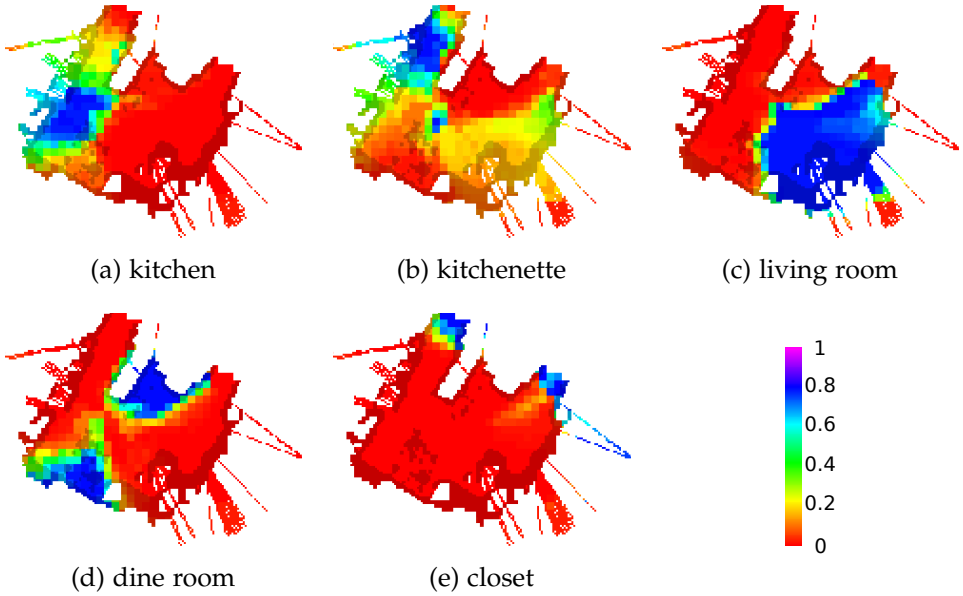
(b) run 1, without room type correlation

(c) run 2

(d) run 3

(e) after executed search

Figure 7.16: Map of the most likely room types for the room *home1*; robot trajectory is depicted as black line



(a) kitchen (b) kitchenette (c) living room

(d) dine room (e) closet

Figure 7.17: Probabilities of the stated room types in room *home1* after run 1; in the legend the probability values for the different colors are stated; blue areas have a high probability of being of the given type, red ares a low probability; The probabilities fit to the most likely room types shown in image (a) of Figure 7.16.

exploration on the result. The trajectory decides which areas are seen how often and from which direction and which areas are seen together in one image. Especially the room type of areas, where the room type classifier is not sure, depends highly on the trajectory. Also the border between areas of different room type is shifted depending on the taken images because one room type might be overruled by the neighboring room type, as described in Section 7.1.1.

In image (e) of Figure 7.16 the result after an object search is shown. Here the favor of the room type classification to detect kitchenettes can be seen. Also noteworthy is the nice shape of the living room area and the basement area, which is probability caused by the robot looking into the wall.

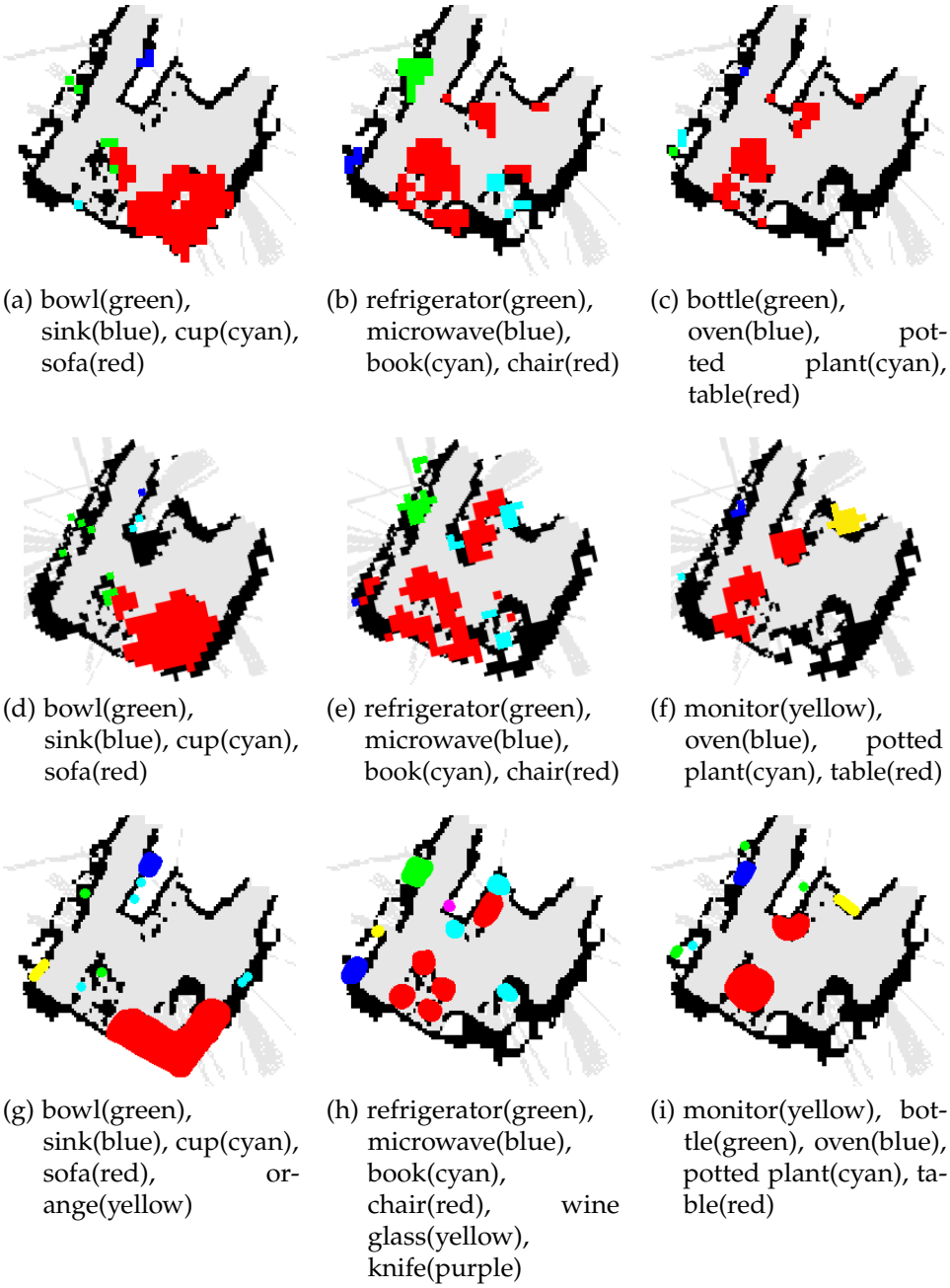
The maps containing the probabilities of the most important room types in the room *home1* after exploration run 1 are shown in Figure 7.17. When

comparing with image (a) in Figure 7.16, it can be seen, as expected, that at every cell the probability of the most likely room type is high. Except at the border of areas, the probability of the most likely room type is very high. This is because the parameters have to be chosen in a way that a quick estimation of the room type during the peek task is possible. Therefore, the probability of the most likely room type of a cell increases relatively quickly to the saturation value of 0.8. Also noteworthy is the relatively high probability for kitchenette on the right side of the room. As can be seen in image (6) of Figure 7.14 the room type classification is not sure about the room type there and proposes multiple room types, including kitchenette.

7.2.3 Object Probability Map

In this section the Object-Probability-Maps are evaluated based on the results of exploration runs in room *home1*. This room is suitable for the evaluation because it consists of areas of different room types and also a variety of different object were present during the runs. For the evaluation in this section the 2D projections of the Object-Probability-Maps are used because the search planning uses the 2D projections and a 3D Object-Probability-Map is difficult to visualize.

In Figure 7.18 all cells with a probability greater than 0.25 in the Object-Probability-Maps of any object type are depicted together with the type of the object. In some cells multiple objects were detected and therefore three images are used for the visualization. These cells are basically where the robot thinks it might have seen an object. The results of the two depicted runs show that big objects, like sofas or tables, are reliably detected. However, the object probabilities of those objects are also high in the areas around the objects due to the missing per-pixel segmentation of the objects. For example parts of the dining table have also a high probability for being a sofa. For the search planning this is usually no problem as the object is either already found during the exploration or found quickly anyway. On the other hand only some smaller objects have a high probability in the Object-Probability-Map. The problem with small objects is that they are often not detected and so the object probability does not increase high enough. The trajectory has a big impact on what objects have a high probability



(a) bowl(green), sink(blue), cup(cyan), sofa(red)

(b) refrigerator(green), microwave(blue), book(cyan), chair(red)

(c) bottle(green), oven(blue), potted plant(cyan), table(red)

(d) bowl(green), sink(blue), cup(cyan), sofa(red)

(e) refrigerator(green), microwave(blue), book(cyan), chair(red)

(f) monitor(yellow), oven(blue), potted plant(cyan), table(red)

(g) bowl(green), sink(blue), cup(cyan), sofa(red), orange(yellow)

(h) refrigerator(green), microwave(blue), book(cyan), chair(red), wine glass(yellow), knife(purple)

(i) monitor(yellow), bottle(green), oven(blue), potted plant(cyan), table(red)

Figure 7.18: Objects probabilities higher than 0.25 after exploration; first row shows results of run 1, second row shows results from run 2, 3rd row shows the ground truth

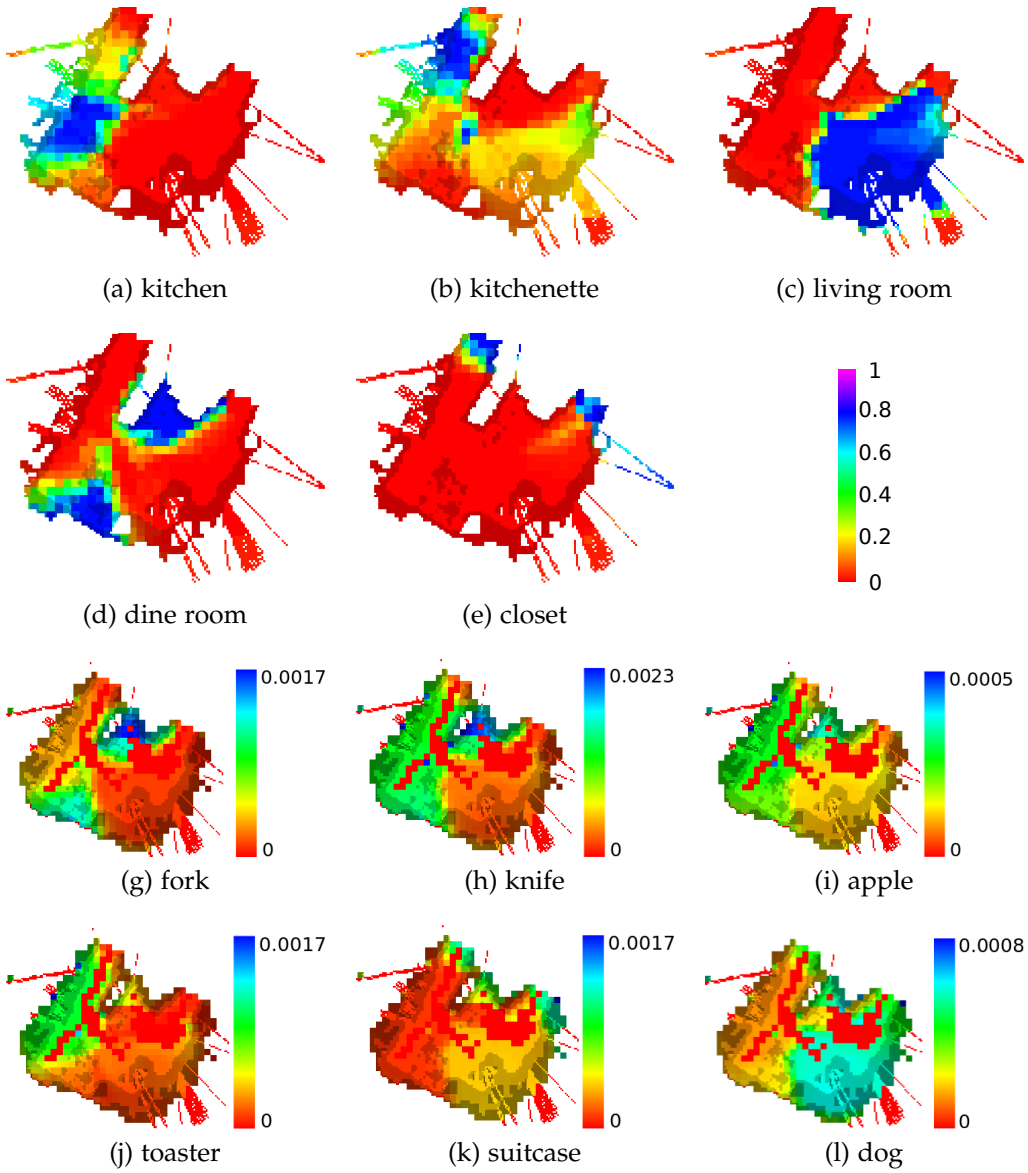
in the Object-Probability-Map. For example the monitor was not seen in run 1 and also not the same cups were detected in the two runs. This is as intended because the exploration is only done to quickly obtain enough data for an intelligent search planning and some areas are not at all or only vague seen in this process.

The probabilities in the Object-Probability-Map do not only depend on the object detections. The information in the Room-Type-Maps is also used to estimate the object probabilities, as described in Section 5.5.5. This indirect way of estimating object probabilities has the most impact if no detections of the given object type were made. In Figure 7.19 the Object-Probability-Maps of some objects not detected during the exploration are depicted, together with the most likely room types of the corresponding Room-Type-Map. In general those probabilities are very low, but many cells together may have a significant object probability. The connection to the Room-Type-Maps can be seen nicely in the images. For example a fork is more likely in a dine room than in areas of other room types or a dog is more likely in a living room. On the other hand a suitcase is very unlikely in a kitchen.

In the upper center the probabilities are relatively high, as can be seen especially in (g) and (h). This area was only seen by the LIDAR, but never by the camera. Therefore, the probabilities for those cells in the Object-Map are still the initial probabilities. In the other cells which were seen by the camera and no object of the given type was found, the probability is lower than the initial value. Therefore, also the values in the Object-Probability-Maps are lower in those cells, which means the cells only seen by the LIDAR have relative high probabilities for containing the object.

The resulting Object-Probability-Maps are reasonable at first glance and the search tests will show how much they improve the object search.

In Table 7.1 the probabilities for an object of the given type being anywhere in the room are shown for the object types in Figure 7.19. The probabilities are according to the Object-Probability-Map. The larger living room area in run 2 results in a larger probability for a dog being in the room. Also noteworthy are the unexpectedly low probabilities for apple and toaster. The reason is that according to the used commonsense knowledge apples or toasters are not very frequent in indoor environments.



(a) kitchen (b) kitchenette (c) living room

(d) dine room (e) closet

(g) fork (h) knife (i) apple

(j) toaster (k) suitcase (l) dog

Figure 7.19: Room type probabilities and object probabilities of given types after exploration run 1

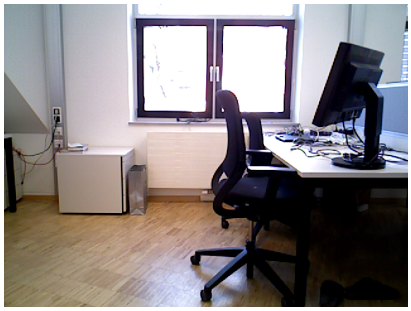
	run 1	run 2	run 3	average
fork	0.184	0.161	0.159	0.168
knife	0.315	0.331	0.314	0.320
apple	0.085	0.095	0.080	0.087
toaster	0.050	0.064	0.052	0.055
suitcase	0.076	0.089	0.061	0.075
dog	0.138	0.179	0.131	0.149

Table 7.1: Object probabilities in room *home1*

7.2.4 Probability Estimations after Peek Task

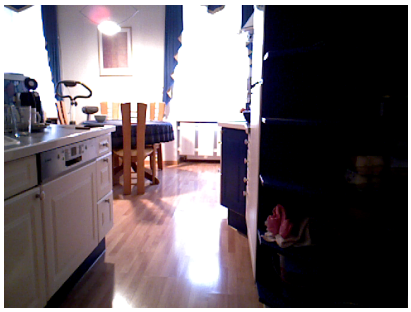
The estimation of the probability for a searched object being in a room after the peek task is very important for the system to work well. Using this estimation the robot decides if the room should be investigated further or if other rooms are more promising. During the peek task only a fraction of the room is seen and so the object probability in the whole room depends on the estimation of the object probability in the rest of the room. This estimation is done using the average room type probabilities of the explored area, as described in Section 5.5.7. In Figure 7.20 images taken at the doorways into different rooms are shown together with the average room type probabilities of the rooms. The average room type probabilities are reasonable, maybe with the exception of the images (e) and (f), but there is neither a meeting room category nor a bathroom category in the detectable room types. One thing to note is that the images (b) and (c) show the same room, but different areas are seen from the different doorways, leading to different results. This shows that the information gathered during the peek task is limited and wrong assumptions might be made.

In Table 7.2 the object probabilities of some selected object types after the peek task are shown. Except for the wine glass in room (d) and the toilet in room (e) none of those objects were seen during the peek task. The toilet was detected with high confidence, which can be seen on the high probability, while the wine glass was only detected in one image, therefore its probability is lower. The general trend of the probabilities looks good, though the accuracy of the probabilities is hard to evaluate without a huge amount of collected data. Only the relatively high probability for toilets is a



(a) office: 0.4442
home office: 0.3214
attic: 0.0174

(b) dining room: 0.3918
livingroom: 0.3093
closet: 0.0878



(c) kitchen: 0.4841
restaurant kitchen: 0.2690
kitchenette: 0.0439

(d) kitchen: 0.3211
dining room: 0.2779
restaurant kitchen: 0.0978

(e) shower: 0.7556
locker room: 0.1701
martial arts gym: 0.0038

(f) classroom: 0.6971
cafeteria: 0.0919
conference auditorium: 0.0256

Figure 7.20: Images taken at the doorway into a room and the average room type probabilities of the top 3 room types

object type	a	b	c	d	e	f
microwave	0.065	0.099	0.317	0.287	0.029	0.050
wine glass	0.088	0.201	0.128	0.223	0.029	0.095
spoon	0.087	0.190	0.323	0.324	0.036	0.156
apple	0.040	0.049	0.074	0.083	0.015	0.041
laptop	0.665	0.152	0.053	0.106	0.046	0.431
mouse	0.584	0.043	0.019	0.041	0.024	0.176
remote	0.288	0.255	0.048	0.103	0.057	0.258
toilet	0.075	0.056	0.098	0.104	0.999	0.065

Table 7.2: Object probabilities of selected object types after peeking into the rooms shown in Figure 7.20

bit weird, as toilets are very uncommon outside bathrooms.

7.3 Search

The search results of a series of test runs are presented in the first part of this section, where the proposed intelligent active object search system and an uninformed system which does not estimate likely object locations, are compared. In the second part of this section the behavior of the proposed system is evaluated in more detail based on individual runs.

7.3.1 Comparison between Intelligent and Uninformed Search

Settings

The test runs were carried out using two different active object search systems. One is the proposed intelligent active object search system which uses the gathered information about rooms, room types, and object locations to speed up the search. The second one is a modified version of the proposed system which does not use this additional information and is therefore called

uninformed search system. In this system the doorway detection is disabled, which means that the room structure is ignored, and all occupied cells have the same probability of containing the searched object. This results in a coverage maximizing search strategy because the objective function of the search planning in the uninformed system increases with the number of well seen cells in a view. The uninformed search system explores the environment before the search is started, so the high-level tasks are the same as for the intelligent search system within a single room.

The test runs were executed in the home environment. Eight different settings were chosen for the test runs, with four different searched objects and two different starting positions per object. Reliable detected objects were chosen to reduce the influence of the object detection and have more emphasis on the search planning. The objects were positioned in for the objects typical locations and between other objects, which means the robot really has to search for them and does not find the objects during the exploration. One example is shown in Figure 7.21, where some knifes are hidden behind some bottles. In Figure 7.22 the map of the environment is shown with marked room types, starting poses and object locations. In Table 7.3 the searched objects, their locations in the map and the starting poses are given for each setting. Also the areas where, according to the used commonsense knowledge, the object is more likely are stated. For each setting six search runs were done, three with the proposed intelligent search system and three with the uninformed search system, which makes in total 48 runs.

setting	start	object	location	likely locations
1	1	remote	1	living room, hotel room, home office
2	3	remote	1	living room, hotel room, home office
3	1	mouse	2	home office
4	3	mouse	2	home office
5	1	knife	3	kitchen, kitchenette, dine room
6	2	knife	3	kitchen, kitchenette, dine room
7	1	spoon	4	kitchen, kitchenette, dine room
8	2	spoon	4	kitchen, kitchenette, dine room

Table 7.3: The eight different settings of the test runs

Figure 7.21: Image of the location of the knifes searched with settings 5 and 6

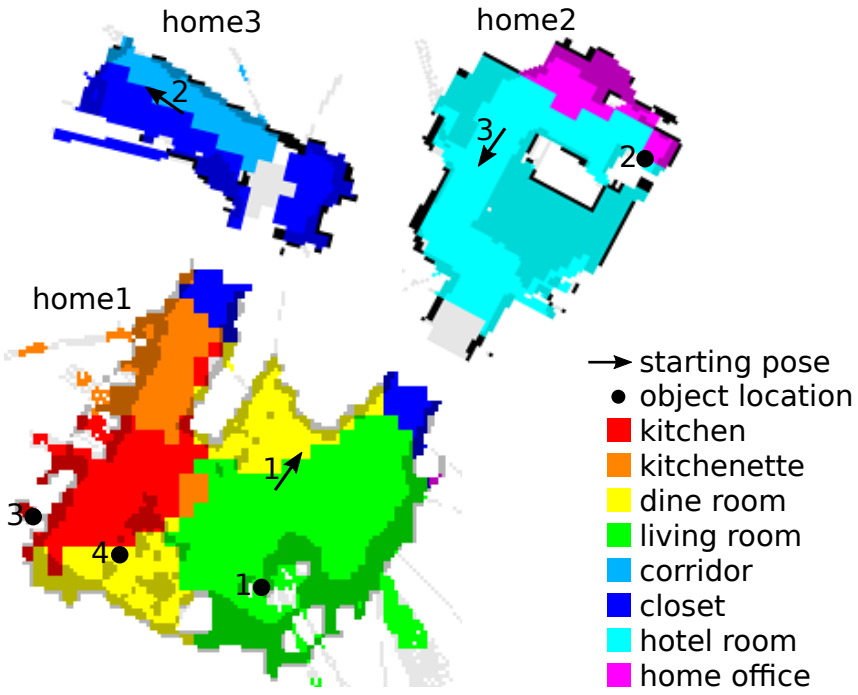


Figure 7.22: Test environment with room types, starting poses and object locations (1: remote, 2: mouse, 3: knife, 4: spoon)

Results

In Figure 7.23 the average search times and average trajectory lengths of the runs of the test series are shown.

The average search time of the intelligent search system was about 30% lower than the average search time of the uninformed search system and the difference between the systems was even higher in the trajectory length, with the intelligent search system traveling nearly 40% less distance. The reason for the greater difference in the trajectory length is that the switching of high-level tasks and the peek tasks take some time (about 5 seconds per high level task switch and about 20 seconds per peek task). This happens more often with the intelligent search system and during this time the robot stays at the same position. The variance in the search runs was high, therefore the search times and trajectory lengths are shown in more detail in Figure 7.24.

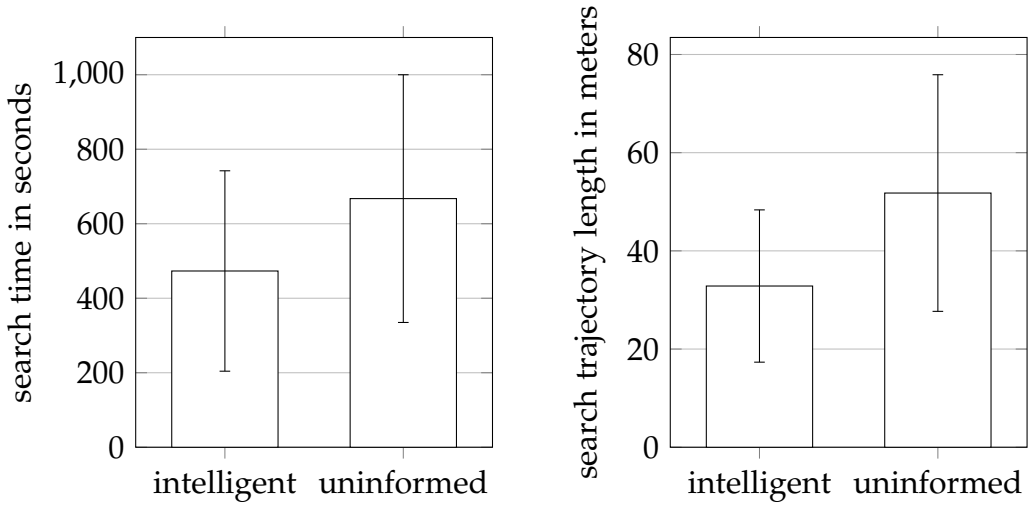


Figure 7.23: Average search times and average trajectory lengths of the test runs

With settings 1 and 2 the uninformed search system performed better than the intelligent search system. This has two reasons. On the one hand the intelligent search system decided in 4 of the 6 runs to search room *home2*

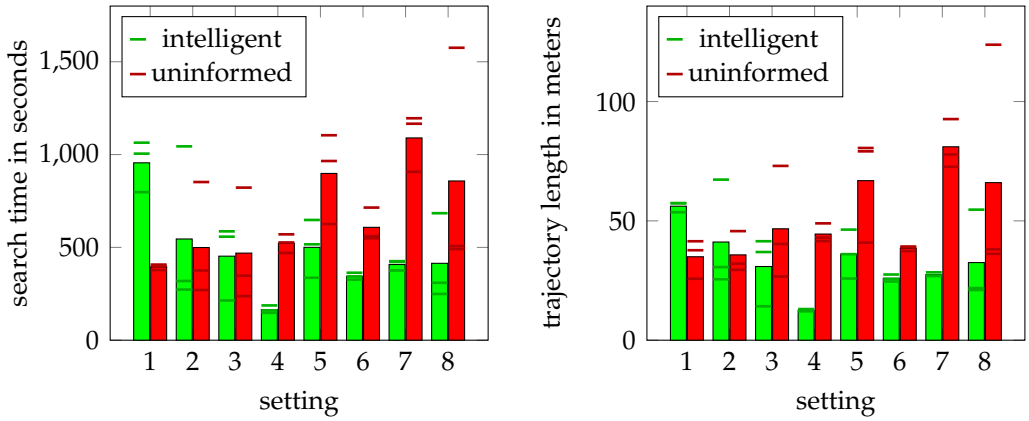
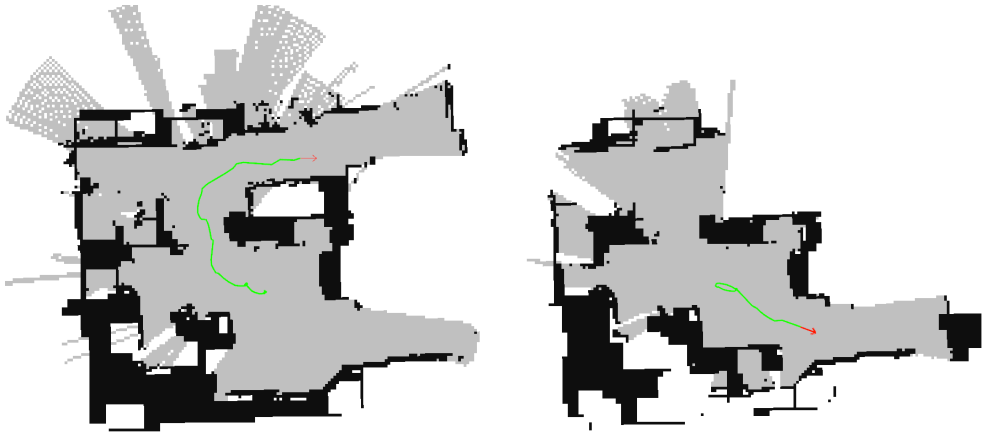


Figure 7.24: Search times and trajectory lengths in test series: The lines show the individual test runs and the bars show the average search times and trajectory lengths of a setting

first, so a lot of time was lost searching there. A more detailed description of the search behavior in these runs is given in the next section. In the other two cases, where room *home1* was searched first, the remote was found quickly. On the other hand the uninformed search system found the remote relatively quickly because the remote could be seen from a view pose, where the robot could cover many occupied cells, which makes it a preferred view pose. Only once the remote was missed at the beginning, which, together with navigation problems, resulted in the one longer search time of the uninformed search system.

The search times for setting 3 highly depended on the direction the robot started to explore. In Figure 7.25 can be seen that the exploration from starting pose 1 can go either first towards room *home3* (image (a)) or towards room *home2* (image (b)). The reason were probably small differences in the navigation. Case (a) was found to happen about twice as often as case (b). In case (a) the intelligent search system first searched room *home3*, which cost some time, before room *home2* was discovered and the object was found. This problem of exploration versus exploitation is described in more detail in the next section. In case (b) the mouse was found quickly. On the other hand the uninformed search system performed better in case (a) because then the exploration ended near the searched mouse and the mouse was

found quickly. This is shown in Figure 7.26.



(a) (b)

Figure 7.25: Possible exploration trajectories from starting pose 1

The intelligent search system found the object very quickly with setting 4 because the robot recognized that it was already in a promising room, so only this room was explored and the mouse was found quickly. The uninformed search system was also relatively quick in this setting as the mouse could be seen from a preferred view pose, where many cells were covered.

With test settings 5, 6, 7 and 8 the intelligent search system performed much better than the uninformed search system. The uninformed search system needed more time with these settings because the searched objects were positioned at locations where they could not be seen form view poses which covered a lot of cells. As the uninformed search used these poses in the beginning of the search it took some time in this mode to find the searched object. The intelligent search system always selected the correct room to search with these settings and most of the time the object was found quickly. In Figure 7.27 the search trajectories for setting 7 and 8 of the intelligent search system are depicted.

The search trajectory lengths show slightly different results because switching high-level tasks and executing the peek tasks have no impact on the

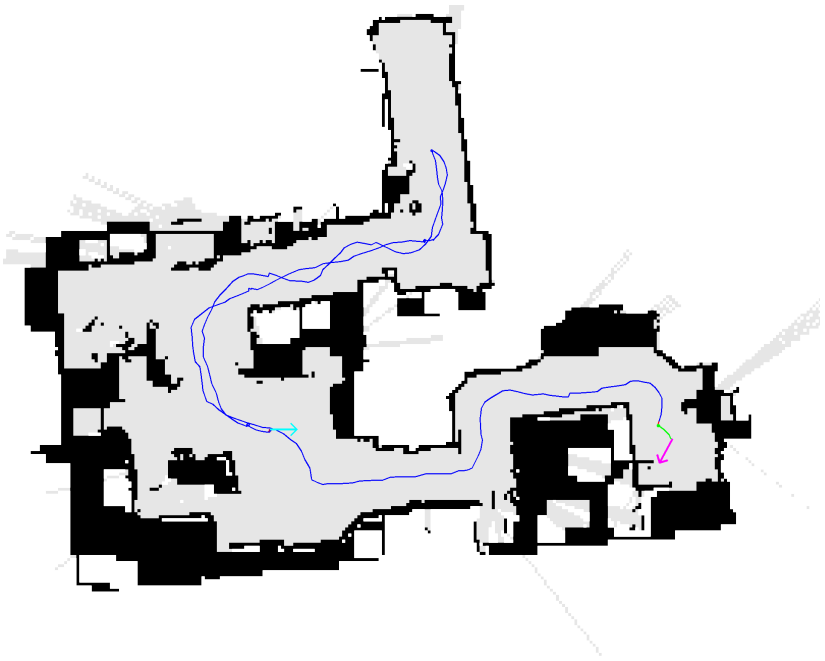


Figure 7.26: Uninformed search for a mouse: The trajectories during exploration (blue) and search (green) are show as well as the starting pose (cyan) and the final pose, where the object was found (purple); The search trajectory is very short because the exploration finished near the searched object

trajectory length. Further the search trajectory lengths are not affected by navigation problems because there the robot does not move. The navigation problems had a big impact on the time necessary to search room *home2* which consisted mainly of narrow areas, as can be seen in the results of the intelligent search with setting 1. However, the trajectory lengths ignore the rotations of the robot, so the search time is still the better measurement for the search performance.

To wrap things up, the intelligent active object search system performs much better than the uninformed system except if there exist multiple rooms with similar probability to contain the object. In this case a lot of time is lost by completely searching a room not containing the object.

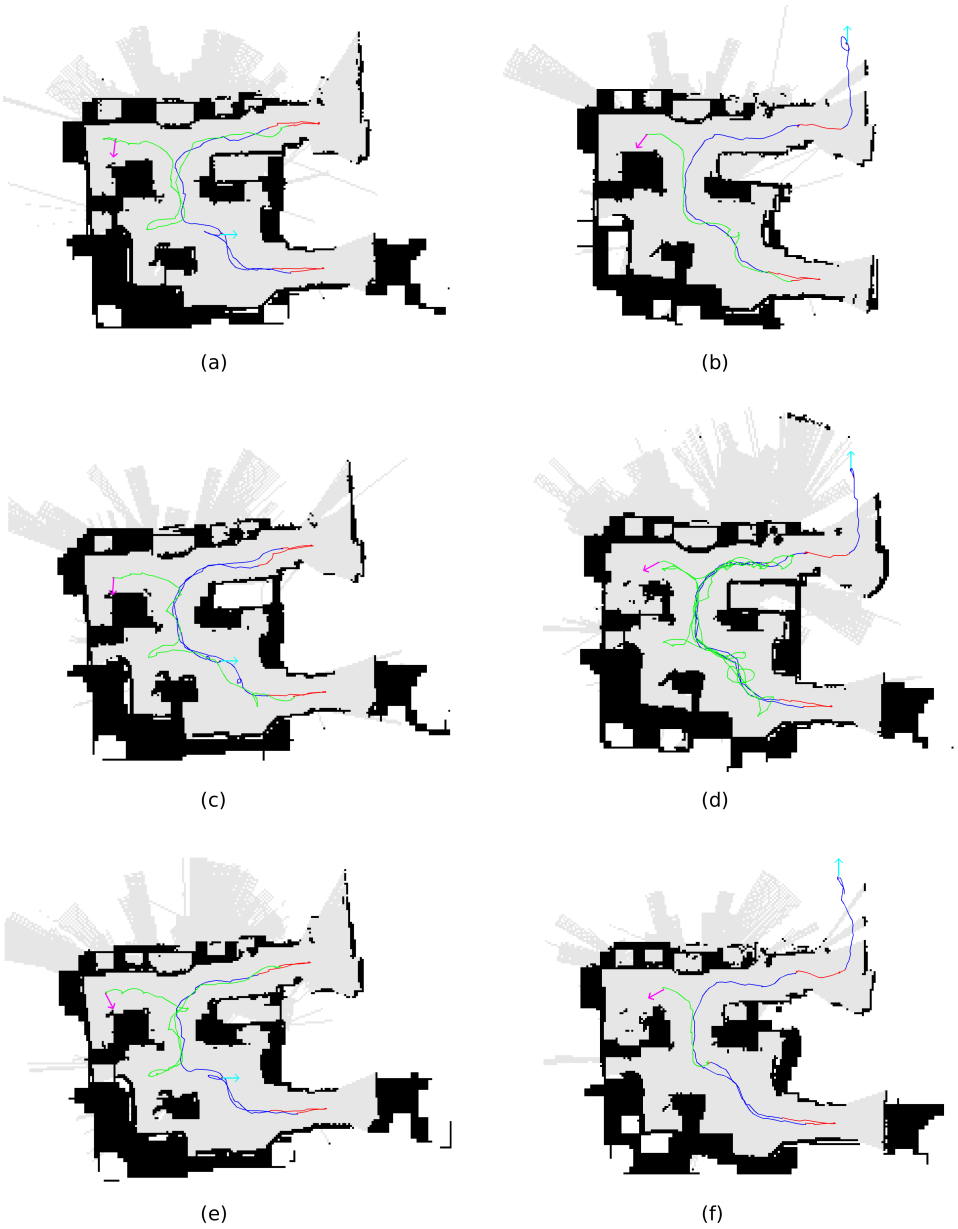
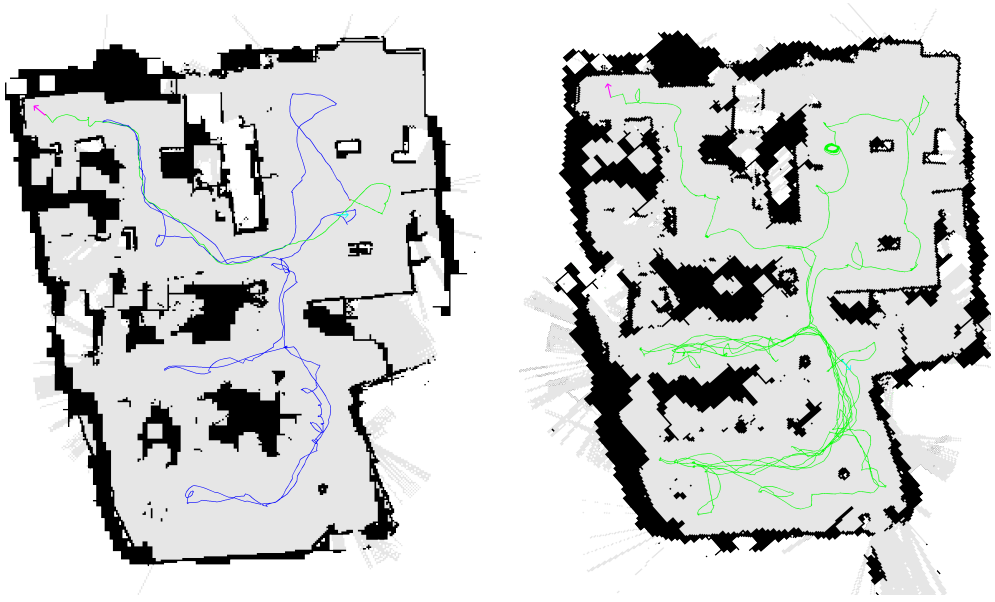


Figure 7.27: Search trajectories for the intelligent search system with settings 7 and 8 (spoon hidden in room *home1*): The trajectories during the exploration (blue), the move through doorway tasks (red) and the search (green) are shown as well as the starting pose (cyan) and the final pose, where the object was found (purple); only the map of room *home1* is shown; in (b) and (f) the object was found very quickly and in (a) and (c) and (e) it took slightly longer; in (d) the robot had difficulties to find the object

7.3.2 Qualitative Evaluation of the Proposed System

Search within a room

The trajectories of the test runs of the intelligent search system with settings 7 and 8, where a spoon was searched, are depicted in Figure 7.27. From the figure it can be seen that the search within a room works very well most of the time. In run (b) and (f) the object was found in one of the first generated view poses, while in run (a), (c) and (e) the search took a bit longer. These small differences are mainly due to the starting position of the robot. In run (d) the object was missed at the beginning, so most of room was searched before the table with the searched spoon was investigated again.



(a) with exploration task; object was found quickly after exploration

(b) without exploration task; the room is only explored as a side effect of the search and therefore the kitchen area was not discovered for a long time

Figure 7.28: Search for a bowl in room *lab1*: The bowl is hidden in the kitchen area in the top left corner. The trajectories during exploration (blue) and search (green) are shown as well as the starting pose (cyan) and the final pose, where the object was found (purple)

Test runs for the search within a room were also executed in room *lab1*. A bowl was hidden in the kitchen part in the top left corner of the room. In image (a) of Figure 7.28 the trajectory of one of those test runs is shown. The exploration took most of the search time. After the exploration the robot drove to some view poses in its vicinity and then immediately to the kitchen area and found the object. Also some test runs were executed with disabled exploration. In this case the exploration and mapping of the room is done during the search and therefore the room is explored only slowly. The trajectory of one of those runs is shown in image (b) of Figure 7.28. In this run the kitchen area was not found for a long time and much time was lost searching in unlikely areas. This shows the benefit of executing an exploration task before the search task.

Search in environments with multiple rooms

Using the commonsense knowledge about which objects are likely in areas of which room types, the robot can decide in which rooms a search is promising. In Figure 7.27 tests runs are shown where the robot searched the room, where the object was located, first.

As already mentioned in Section 7.3.1, with setting 2, where a remote was hidden in room *home1*, the robot sometimes searched first in room *home2* and sometimes first in room *home1*. Runs for both cases are depicted in Figure 7.29. In Table 7.4 the object probabilities and expected search times of all rooms, and the expected search times of the path, where room *home1* was searched first, and the path, where room *home2* was searched first, are stated for both runs. In both runs room *home1* was found to be more likely to contain the object, but also larger, compared to room *home2*.

	home1			home2			home3			1-2-3	2-1-3
	P(o)	T	E(t))	P(o)	T	E(t)	P(o)	T	E(t)	E(t)	E(t)
run (a)	0.213	728	616	0.158	400	360	0.034	232	227	1124	1107
run (b)	0.249	684	563	0.153	484	438	0.041	256	250	1122	1152

Table 7.4: Object probabilities (P(o)), times to completely search the room (T) and expected search times (E(t) of the three rooms, and the expected search times for the search sequences *home1-home2-home3* and *home2-home1-home3* for the two runs shown in Figure 7.29

(a) robot decided to search room *home2* before room *home1*

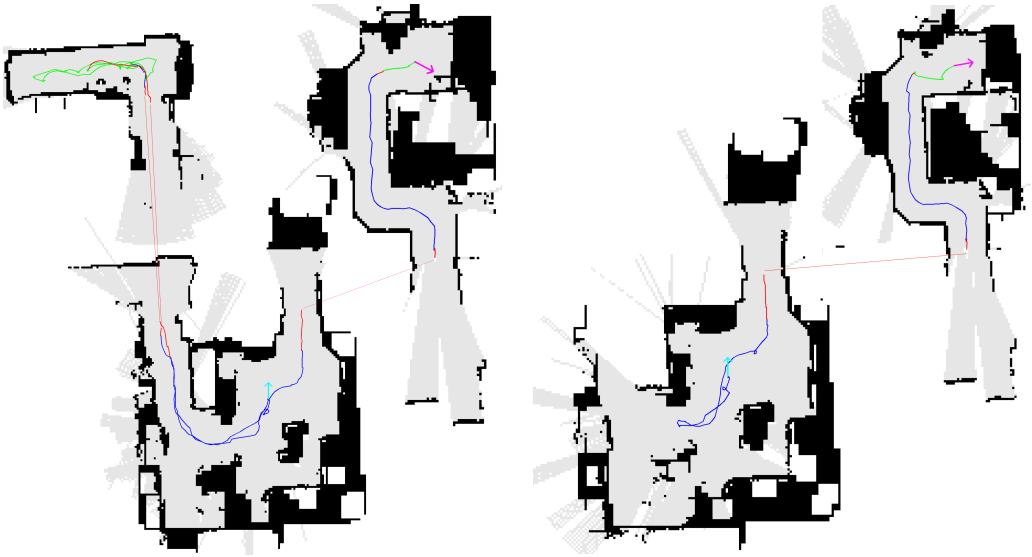
(b) robot decided to search room *home1* first

Figure 7.29: Example runs of searches for a remote: The trajectories during exploration (blue), move through doorway (red) and search (green) are shown as well as the starting pose (cyan), the final pose, where the object was found (purple), and the connections of the paths over room borders (red lines); room *home1* is at the bottom, room *home2* is at the top right and room *home3* is at the top left

Therefore, the expected search times for the search sequence *home1-home2-home3* and the search sequence *home2-home1-home3* are similar. The other possible search sequences have a higher expected search time because room *home3* is very unlikely to contain the object and is therefore better searched last. In run (a) the difference of the object probabilities in room *home1* and room *home2* is small and the difference in the expected search time is high, therefore searching room *home2* first is optimal. In run (b) the difference in the object probabilities is higher and the difference in the expected search times is lower. This makes searching room *home1* first the optimal decision because the probability that *home2* need not be searched is high enough to compensate for the higher expected search time in room *home1*.

The robot sometimes makes suboptimal decision when no likely room was discovered yet. In Figure 7.30 two search runs with test setting 3, where a mouse is hidden in room *home2*, are depicted. In both runs the robot started

in the middle of room *home1*, but in the run shown in (a) the exploration was done towards room *home3* and in the run shown in (b) towards room *home2*. Therefore, in run (a) room *home3* was entered first and as both visited rooms had a low probability for containing the searched mouse, the smaller room *home3* was searched. Only after that room *home1* was further explored, room *home2* was discovered, and the mouse was found. In run (b) room *home2* was discovered first and the mouse was found quickly. In both runs the robot correctly recognized that room *home1* was less likely to contain a mouse than room *home2*.



(a) room *home3* was discovered before room *home2*; the robot decided to search room *home3* because it is a small room and no room with a high object probability was yet discovered

(b) room *home2* was discovered before room *home3*; the object probability in room *home2* was high, therefore the robot searched there

Figure 7.30: Example runs of searches for a remote: The trajectories during exploration (blue), move through doorway (red) and search (green) are shown as well as the starting pose (cyan), the final pose, where the object was found (purple), and the connections of the paths over room borders (red lines); room *home1* is at the bottom, room *home2* is at the top right and room *home3* is at the top left

In Figure 7.31 a search for a wine glass hidden in the kitchen of the institute

environment is shown. In this run the kitchen was the last room which was discovered. The search in the bathroom was postponed, but in two other unlikely rooms a search was executed. This is similar to the above mentioned search run for the mouse, where the robot prefers to searches a small, unlikely room before exploring a larger, unlikely room. After discovering the kitchen the robot correctly recognized that the kitchen was the most promising room to search.

In Figure 7.32 a search for a toilet in the institute environment is depicted. In this run the robot correctly recognized all discovered rooms to be unlikely to contain a toilet until the bathroom was found, where the toilet was found.

Search in already explored environments

In Figure 7.33 two runs executed after the exploration of the institute environment are shown. The toilet which was searched in the first run, was seen during the exploration and the robot used this information and directly drove to the bathroom, where the toilet was. The wine glass which was searched in the second run, was put out of a cupboard after the exploration, so the robot had not seen it. Nevertheless, the robot recognized that the kitchen was the most likely location for a wine glass and immediately drove there. After a short search in the kitchen the wine glass was found.

Summary

To recap the findings in the test runs, the search within a room is executed efficiently. Areas within a room where the object is more likely are searched first, leading to low search times. In environments with multiple rooms the proposed system makes suboptimal decisions if no room is found where the object is likely. This problem of weighting exploration versus exploitation is difficult to solve. However, when the discovered rooms have either a high or very low probability to contain the object, reasonable decisions are made by the robot.

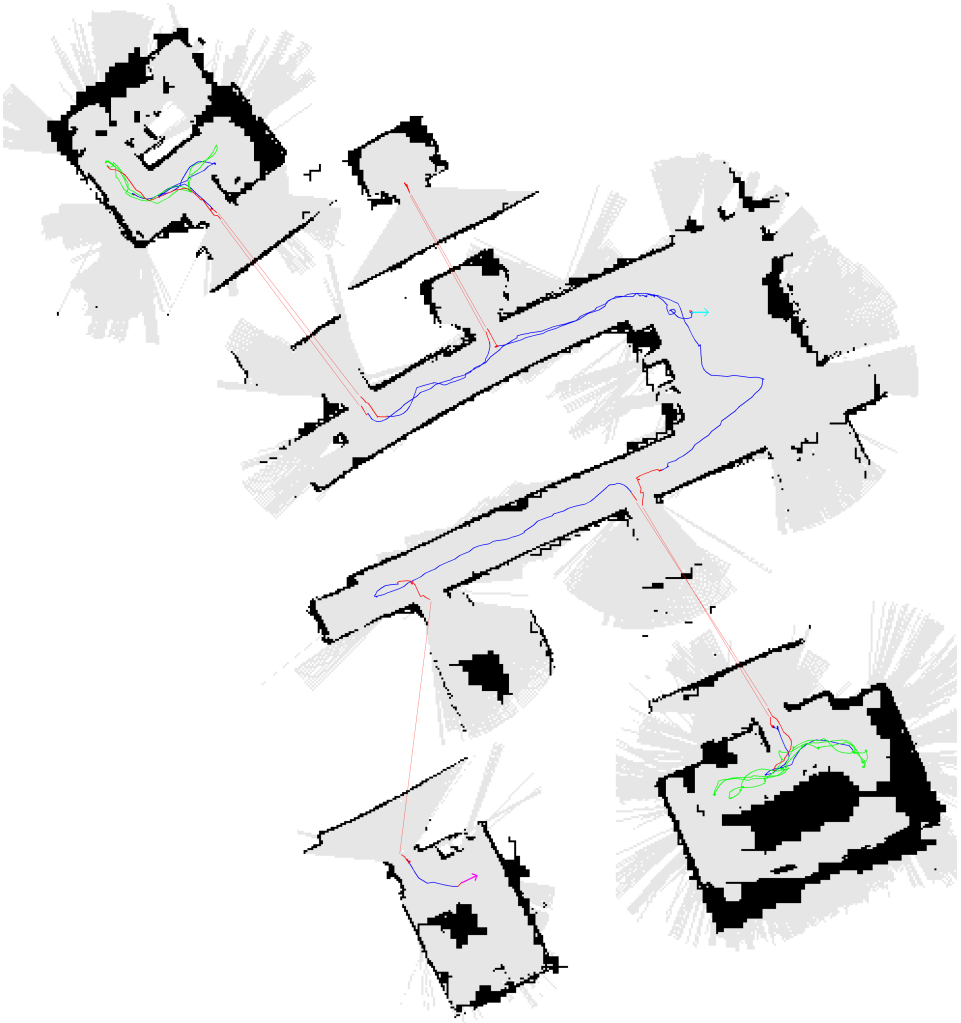


Figure 7.31: Search of a wine glass in the institute environment: The wine glass is in the kitchen (bottom left room). The trajectories during exploration (blue), move through doorway (red) and search (green) are shown as well as the starting pose (cyan), the final pose, where the object was found (purple), and the connections of the paths over room borders (red lines)

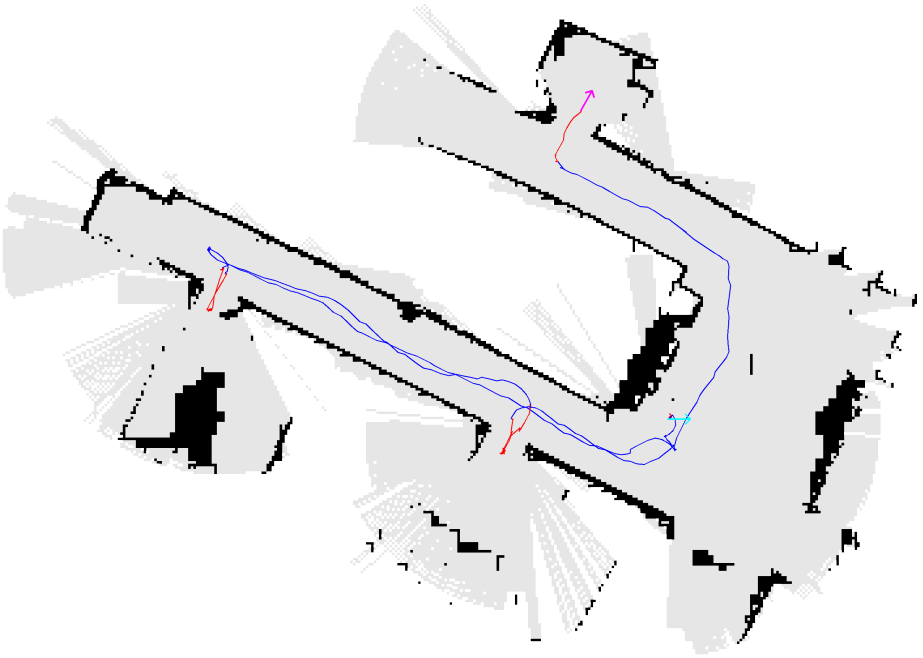


Figure 7.32: Search of a toilet in the institute environment: The trajectories during exploration (blue), move through doorway (red) and search (green) are shown as well as the starting pose (cyan) and the final pose, where the object was found (purple). Only the map of room *institute1* is shown because all other rooms are just entered and left immediately after the peek task.

(a) search for a toilet (b) search for a wine glass

Figure 7.33: Search runs in the explored institute environment: Search for a toilet (a) and search for a wine glass (b)

In general the performance of the proposed system depends on the environment and the searched object. In environments with similar rooms and when searching for objects which can be nearly everywhere in the environment, the proposed system has no advantages and might even perform worse than uninformed search systems. However, in most scenarios some rooms are very unlikely to contain a searched object and some rooms have a high probability to contain a searched object. Then the proposed system significantly outperforms an uninformed search system by searching in likely rooms and postponing searches in unlikely rooms.

8 Conclusion and Future Work

In this thesis an intelligent active object search system was presented. The information available about the search environment is important for an efficient execution of the object search. Therefore a hybrid, semantic mapping system was developed. The proposed hybrid map design which combines a topological representation of the room structure and separated semantic metric maps for each room, allows to split the search problem into two abstraction levels. Furthermore, the segmentation of the environment into rooms resulted in smaller metric maps which proved to be less prone to inaccuracies in the map creation procedure. The semantic maps of the rooms contain, in addition to occupancy information, information about where objects were seen, which is obtained using a CNN for object detection, and information about the room types of areas in the room which is obtained using a CNN for place classification. Using the information in the maps and the commonsense knowledge about the connection between room types and objects typically located there, likely object locations within a room and the probability of a searched object being in a room were estimated. This estimation is also possible in partially explored rooms, like after peeking into a room. The search is planned in two steps. At first the next room to search was selected based on the topological map and then the actual search was planned by a second planner which had only to consider the semantic metric map of the current room.

The generated maps appear reasonable, though an exact evaluation of the maps was not possible due to the lack of a reasonable ground truth. Comparisons with a coverage maximizing search system and the evaluation of the behavior of the robot using the proposed intelligent search system showed that this system improves the search performance, measured by the search time and the search path, significantly. However, also some problems and suboptimal behaviors were encountered. Basic components of

the system, like the navigation and the object detection, were still far from being perfect and problems of those components had a negative impact on the overall search performance. Also the search system itself has much room for further improvements. The information in the maps was found to be vague and the search planning has especially problems to find a good balance between exploration and exploitation. This resulted in some suboptimal decision about what room should be searched next.

The way to a human-like search performance is still very long and a lot of further research in many different areas is needed to achieve this. Better object detection systems are necessary to find objects faster and more reliably. More accurate and more extensive commonsense knowledge about indoor environments can help to improve the reasoning about object locations and also the interaction with humans or other robots can help to find objects faster. Also more sophisticated search planning approaches are necessary which can better handle the complexity of the problem and the uncertainty of the sensor data and the commonsense knowledge. While there is much room for improvements, this work showed some promising ideas and the potential of intelligent robot behavior in everyday human environments.

Appendix

Bibliography

[1] A. Aydemir, A. Pronobis, M. Göbelbecker, and P. Jensfelt, "Active visual object search in unknown environments using uncertain semantics," *IEEE Transactions on Robotics*, vol. 29, no. 4, pp. 986–1002, Aug. 2013, ISSN: 1552-3098. DOI: 10.1109/TRO.2013.2256686 (cit. on pp. 2, 8, 10).

[2] C. Dornhege, A. Kleiner, and A. Kolling, "Coverage search in 3d," in *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Oct. 2013, pp. 1–8. DOI: 10.1109/SSRR.2013.6719340 (cit. on pp. 7, 19).

[3] R. Karp, "Reducibility among combinatorial problems," vol. 40, pp. 85–103, Jan. 1972 (cit. on pp. 7, 19).

[4] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton, NJ, USA: Princeton University Press, 2007, ISBN: 0691129932, 9780691129938 (cit. on pp. 7, 19).

[5] T. Kollar and N. Roy, "Utilizing object-object and object-scene context when planning to find things," in *2009 IEEE International Conference on Robotics and Automation*, May 2009, pp. 2168–2173. DOI: 10.1109/ROBOT.2009.5152831 (cit. on pp. 7, 10, 11).

[6] L. Kunze, K. K. Doreswamy, and N. Hawes, "Using qualitative spatial relations for indirect object search," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 163–168. DOI: 10.1109/ICRA.2014.6906604 (cit. on p. 7).

[7] P. Viswanathan, D. Meger, T. Southey, J. J. Little, and A. K. Mackworth, "Automated spatial-semantic modeling with applications to place labeling and informed search," in *2009 Canadian Conference on Computer and Robot Vision*, May 2009, pp. 284–291. DOI: 10.1109/CRV.2009.49 (cit. on p. 8).

[8] A. Aydemir, K. Sjöö, J. Folkesson, A. Pronobis, and P. Jensfelt, "Search in the real world: Active visual object search based on spatial relations," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 2818–2824. DOI: 10.1109/ICRA.2011.5980495 (cit. on p. 8).

[9] K. Sjöö, A. Aydemir, and P. Jensfelt, "Topological spatial relations for active visual search," *Robotics and Autonomous Systems*, vol. 60, no. 9, pp. 1093–1107, 2012, ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2012.06.001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889012000851 (cit. on p. 8).

[10] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, Dec. 1989, ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541. [Online]. Available: http://dx.doi.org/10.1162/neco.1989.1.4.541 (cit. on p. 8).

[11] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013. arXiv: 1311.2524. [Online]. Available: http://arxiv.org/abs/1311.2524 (cit. on p. 8).

[12] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010, ISSN: 1573-1405. DOI: 10.1007/s11263-009-0275-4. [Online]. Available: https://doi.org/10.1007/s11263-009-0275-4 (cit. on p. 8).

[13] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. arXiv:

1405.0312. [Online]. Available: http://arxiv.org/abs/1405.0312 (cit. on p. 8).

[14] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. arXiv: 1504.08083. [Online]. Available: http://arxiv.org/abs/1504.08083 (cit. on p. 8).

[15] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. arXiv: 1506.01497. [Online]. Available: http://arxiv.org/abs/1506.01497 (cit. on p. 8).

[16] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015. arXiv: 1512.02325. [Online]. Available: http://arxiv.org/abs/1512.02325 (cit. on p. 8).

[17] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. arXiv: 1506.02640. [Online]. Available: http://arxiv.org/abs/1506.02640 (cit. on p. 8).

[18] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016. arXiv: 1612.08242. [Online]. Available: http://arxiv.org/abs/1612.08242 (cit. on p. 8).

[19] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. arXiv: 1804.02767. [Online]. Available: http://arxiv.org/abs/1804.02767 (cit. on pp. 8, 72, 112).

[20] M. Schwarz, A. Milan, P. A. S., and B. S., "Multi-class rgb-d object detection and semantic segmentation for autonomous manipulation in clutter," 2017 (cit. on p. 9).

[21] N. Sünderhauf, T. Pham, Y. Latif, M. Milford, and I. D. Reid, "Meaningful maps - object-oriented semantic mapping," *CoRR*, vol. abs/1609.07849, 2016. arXiv: 1609.07849. [Online]. Available: http://arxiv.org/abs/1609.07849 (cit. on p. 9).

[22] 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, pp. 34–46, Feb. 2007, ISSN: 1552-3098. DOI: 10.1109/TRO.2006.889486 (cit. on pp. 9, 15, 32).

[23] A. Doucet, N. de Freitas, K. P. Murphy, and S. J. Russell, "Rao-blackwellised particle filtering for dynamic bayesian networks," *CoRR*, vol. abs/1301.3853, 2013. arXiv: 1301.3853. [Online]. Available: http://arxiv.org/abs/1301.3853 (cit. on pp. 9, 16, 32).

[24] H. Moravec, "Sensor fusion in certainty grids for mobile robots," *AI Mag.*, vol. 9, no. 2, pp. 61–74, Jul. 1988, ISSN: 0738-4602. [Online]. Available: http://dl.acm.org/citation.cfm?id=46184.46187 (cit. on pp. 9, 35).

[25] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2o: A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 3607–3613. DOI: 10.1109/ICRA.2011.5979949 (cit. on p. 9).

[26] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "Fastslam: A factored solution to the simultaneous localization and mapping problem," in *In Proceedings of the AAAI National Conference on Artificial Intelligence*, AAAI, 2002, pp. 593–598 (cit. on p. 9).

[27] M. Montemerlo, S. Thrun, D. Roller, and B. Wegbreit, "Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, ser. IJCAI'03, Acapulco, Mexico: Morgan Kaufmann Publishers Inc., 2003, pp. 1151–1156. [Online]. Available: http://dl.acm.org/citation.cfm?id=1630659.1630824 (cit. on p. 9).

[28] A. Kleiner, R. Baravalle, A. Kolling, P. Pilotti, and M. Munich, "A solution to room-by-room coverage for autonomous cleaning robots," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 5346–5352. DOI: 10.1109/IROS.2017.8206429 (cit. on p. 9).

[29] S. Friedman, H. M. Pasula, and D. Fox, "Voronoi random fields: Extracting topological structure of indoor environments via place labeling," in *IJCAI*, 2007 (cit. on p. 10).

[30] K. Wada, K. Okada, and M. Inaba, "Probabilistic 3d multilabel real-time mapping for multi-object manipulation," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 5092–5099. DOI: 10.1109/IROS.2017.8206394 (cit. on p. 10).

[31] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, Jul. 1997, pp. 146–151. DOI: 10.1109/CIRA.1997.613851 (cit. on pp. 10, 54).

[32] S. L. Lauritzen and T. S. Richardson, "Chain graph models and their causal interpretations," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 64, no. 3, pp. 321–348, DOI: 10.1111/1467-9868.00340. eprint: https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/1467-9868.00340. [Online]. Available: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/1467-9868.00340 (cit. on p. 10).

[33] S. Yoon, A. Fern, and R. Givan, "Ff-replan: A baseline for probabilistic planning," in *In ICAPS*, 2007 (cit. on p. 10).

[34] M. Quigley, K. Conley, B. P Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y Ng, "Ros: An open-source robot operating system," vol. 3, Jan. 2009 (cit. on p. 13).

[35] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment," in *2010 IEEE International Conference on Robotics and Automation*, May 2010, pp. 300–307. DOI: 10.1109/ROBOT.2010.5509725 (cit. on p. 14).

[36] L. Wang, S. Guo, W. Huang, and Y. Qiao, "Places205-vggnet models for scene recognition," *CoRR*, vol. abs/1508.01667, 2015. arXiv: 1508.01667. [Online]. Available: http://arxiv.org/abs/1508.01667 (cit. on pp. 23, 71).

[37] M. Hanheide, C. Gretton, R. Dearden, N. Hawes, J. Wyatt, A. Pronobis, A. Aydemir, M. Göbelbecker, and H. Zender, "Exploiting probabilistic knowledge under uncertain sensing for efficient robot behaviour," in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three*, ser. IJCAI'11, Barcelona, Catalonia, Spain: AAAI Press, 2011, pp. 2442–2449, ISBN: 978-1-57735-515-1. DOI: 10.5591/978-1-57735-516-8/IJCAI11-407. [Online]. Available: http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-407 (cit. on p. 24).

[38] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013, Software available at http://octomap.github.com. DOI: 10.1007/s10514-012-9321-0. [Online]. Available: http://octomap.github.com (cit. on p. 34).

[39] A. Aydemir, "Exploiting structure in man-made environments," QC 20121105, PhD thesis, KTH, Computer Vision and Active Perception, CVAP, 2012, pp. vi, 103, ISBN: 978-91-7501-549-1 (cit. on p. 47).

[40] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *CoRR*, vol. abs/1408.5093, 2014. arXiv: 1408.5093. [Online]. Available: http://arxiv.org/abs/1408.5093 (cit. on p. 71).

[41] J. Redmon, *Darknet: Open source neural networks in c*, http://pjreddie.com/darknet/, 2013–2016 (cit. on p. 72).