Dissertation

Vision-Based Autonomous Robotic Floor Cleaning in Domestic Environments

Der Technischen Fakultät
der Universität Bielefeld
vorgelegt von

David Roman Fleer

zur Erlangung des Grades eines
Doktors der Ingenieurwissenschaften.

März 2018

Gedruckt auf alterungsbeständigem Papier gemäß ISO 9706.

Acknowledgments

This dissertation is built around the cleaning-robot framework developed by the Computer Engineering Group at Bielefeld University. As seen in Section 2.1.3, this framework combines the work of many contributors. The author is especially grateful to Ralf Möller, Michael Horst, Klaus Kulitza, Martin Krzykawski, Lorenz Hillen, Tim Köhler, and Frank Röben. Their efforts within the project all added to the foundations that made this work possible.

While working on the present monograph, the author was also supported in numerous other ways: As the adviser for this dissertation, Ralf Möller deserves credit for many valuable discussions, comments, and pieces of advice. The author is especially grateful to Michael Horst, who supported the experiments shown in Figure 2.26. He also contributed towards acquiring the maps from the `Real1`–`Real3` environments in Chapter 5. Furthermore, Michael provided many interesting and informative conversations, as well as valuable feedback regarding Chapter 3. This work involves numerous experiments with a cleaning-robot prototype. Here, Klaus Kulitza provided invaluable assistance by making many timely adjustments and repairs to the robot's hardware. The experiments in Chapter 3 make use of a visual robot-tracking system. Special thanks go to Michael Horst and Andreas Stöckel, who developed this system together with the author; Klaus Kulitza also contributed some of the system's hardware.

The Cluster of Excellence Cognitive Interactive Technology (CITEC) at Bielefeld University kindly allowed us to conduct experiments in their Cognitive Service Robotics Apartment. Credit goes to Marian Pohling of the Ambient Intelligence research group for arranging our use of this facility. The author also wishes to thank the research group of Barbara Hammer at Bielefeld University for valuable advice regarding the use of machine learning in Chapter 5. Some of the research for Chapter 3 was supported by Deutsche Forschungsgemeinschaft (grant no. MO 1037/10-1).

On a more personal note, the author offers his gratitude to the fellow member of the Computer Engineering Group. Over the years, they provided many hours of encouraging words, engaging discussions, and productive collaborations. Finally, I give my heartfelt thanks to my parents, my family, and all my friends for their support while creating this dissertation. Although they may have made no direct contributions, their help was perhaps the most important of all.

Contents

List of Figures

List of Tables

1. Introduction

In recent years, floor cleaning has become a popular application for autonomous mobile robots. This task is commonly solved by a wheeled robot, which carries a mechanism that cleans any surface across which it drives. Consequently, the goal for such a robot is to completely traverse an initially unknown floor space. As a secondary goal, the robot should minimize the time and travel distance required for this task. In this dissertation, we focus on *domestic* floor-cleaning robots which are designed for typical, unmodified home or office environments. These robots are comparatively small, with a diameter of about $30\,\mathrm{cm}$ and a height of $10\,\mathrm{cm}$. The commercial market for such robots has recently undergone rapid growth. At the time of writing, a cursory search showed over 60 different models from more than a dozen companies, typically selling for 300 to 1000 Euros. Indeed, this seems to be the only type of autonomous robot that currently sees widespread use in domestic, indoor environments.

Based on both industry-funded and independent research, our group has developed an intelligent floor-cleaning robot. During this research project, we encountered several unsolved problems and open questions. In this work, we study and solve some of these problems within the context of our cleaning robot. This includes questions concerning our cleaning robot's strategy, visual localization, and room segmentation. Like its commercial counterparts, our robot carries only limited onboard resources. It is equipped with only a low-power embedded computer, and carries a single panoramic camera as its main sensor. Even with these limited capabilities, our robot must be able to solve its cleaning task in real time. Consequently, we focus on novel and unconventional solutions that work within these constraints: Our robot uses a topo-metric map which does not try to achieve global metric consistency; this greatly simplifies map maintenance and construction. The images recorded by the single monochrome camera are used for localization, navigation, mapping, and obstacle detection. Unlike much of the literature, we emphasize appearance-based vision techniques to solve these tasks. Such methods do not need to build a map of external landmarks or reconstruct depth information. However, the present work is not strictly limited to this cleaning-robot context. Where appropriate, we also seek to provide knowledge and techniques which are useful for domestic robots in general. For example, Chapter 3 includes general advice on choosing a method for visual relative-pose estimation in indoor environments.

Besides this introduction, this thesis contains five subsequent chapters: We begin with an introduction of our group's cleaning-robot framework in Chapter 2. Here, we also outline the improvements which the author contributed to the project. We then study three specific problems in depth during Chapters 3 to 5. Chapter 6 provides a final summary, conclusions, and outlook. Below, we briefly introduce each of these chapters and its contents. For more in-depth information, please refer to the abstract at the beginning of Chapters 2 to 5.

Chapter Overview

Chapter 2 introduces our cleaning-robot framework, as well as our physical prototype. Besides giving a general overview, this introduction provides the necessary background for the subsequent chapters. We discuss both our framework's overarching design philosophy and its specific hardware and software components. Since our cleaning robot was developed by a number of people, we also point out the specific contributions by the author. These include major advances in the robot's

collision handling, as well as the mapping, detection, and avoidance of obstacles. The author also improved the robot's planning process and control automatons, which also enhanced the performance in narrow spaces. We then present cleaning runs performed during real-world and simulator experiments. Within these experiments, our robot demonstrates successful cleaning in complex, multi-room domestic environments.[1] To our knowledge, this is the first academic project that describes a complete framework with this capability. We also discuss specific limitations and trade-offs based on a series of examples.

In Chapter 3, we evaluate several methods for visual relative-pose estimation within the domestic robot context. As discussed in Chapter 2, relative-pose estimation is one of the core components in our robot's localization and mapping system. The literature contains numerous holistic and feature-based solutions for this problem. However, a lack of comprehensive comparisons makes it difficult to select an appropriate method. To fill this gap, we evaluate several candidates for their accuracy, speed, and robustness. To this end, we use our robot to collect novel panoramic image databases from domestic environments. As far as we know, our study is the first specific comparison between holistic and feature-based pose-estimation methods. We also investigate aspects which are especially important for domestic robots: Since real-world applications often encounter strong illumination changes, we test the methods on day-night image pairs. We also measure each candidate's execution time on our robot's onboard computer. This lets us determine which candidates can meet real-time constraints while using limited computing power. Our study also includes methods which only estimate a limited relative pose within a 2D plane. Such candidates are especially suitable for domestic robots which move across a floor plane; our prototype also employs such a planar-motion method. Throughout these experiments, we found that no method is superior in all respects. Particularly, more accurate and robust candidates are usually also much slower. Consequently, our detailed results should be useful when choosing a visual pose-estimation method for a specific application.

In Chapter 4, we visually estimate our robot's tilt angle and direction relative to its movement plane. Since our robot usually moves on an even floor, our framework assumes that this tilt is zero. This planar-motion assumption allows for faster and more accurate visual pose estimation (Chapter 3), and is also used in visual place recognition [69]. However, tilting the robot violates this assumption, and can have a large negative impact on the performance of these techniques. We therefore present two schemes that estimate the tilt angle and direction from the panoramic images recorded by our robot. These schemes rely on the detection of vertical edges, which commonly occur in domestic indoor environments. We improve the quality and speed of our methods by carefully utilizing the properties of our panoramic images. To evaluate our solutions, we use our robot to capture images with known tilts in several domestic spaces. In subsequent experiments, we found that our methods are both accurate and fast. This should make them suitable components for real-time tilt correction in visual planar-motion methods. However, surroundings that lack clear vertical edges can be problematic.

During its cleaning run, our robot creates a map of its environment. Chapter 5 presents a novel method that segments this map into individual rooms. Such a room segmentation has several applications, including user interaction and hierarchical planning. We thus seek to produce a segmentation that corresponds to a human-derived concept of rooms. To achieve this goal, our method learns to detect room borders from training segmentations provided by a human operator. In a novel approach, we integrate this machine-learning capability without the complexities of full-scale semantic mapping. To thoroughly utilize the data collected by our robot, our solution incorporates the map's topology and geometry, panoramic images, and obstacle data. We evaluate the proposed method using maps from real-world and high-detail simulated environments. In

[1] Since our prototype lacks an actual cleaning unit, we consider a location to be cleaned if it would have been covered by a hypothetical 30 cm wide cleaning orifice.

general, our segmentation successfully reproduces the rooms identified by a human user. However, unusual room borders which are not well-represented in the training set remain a challenge.

Chapter 6 briefly reviews the previous chapters and summarizes their results. We also consider the main conclusions drawn from these results, and formulate future research goals. Finally, we consider some overarching lessons and insights that go beyond beyond the individual chapters.

Style and Notation

This work contains a significant volume of mathematical expressions in several different contexts. It is thus difficult to ensure that each mathematical symbol denotes only one specific entity. Most likely, this would only be possible by using many different and potentially obscure symbols or subscripts. However, many variables and terms only appear within a specific section or subsection. We therefore choose to reuse some symbols, especially generic ones like i, N, or $\vec{x}$. Nevertheless, we ensure that the notation is always consistent within each section or subsection. When in doubt, the reader should thus always use the most recent definition of any mathematical symbol.

Note that this work uses the *authorial we* commonly found in the academic literature. Unless stated otherwise, this *we* does not imply the authorship, contributions, or opinions of a third party.

2. An Introduction to our Autonomous Cleaning Robot Framework

In this chapter, we provide a general overview of our autonomous cleaning robot and its control framework. This robot cleans an unknown floor space by fully traversing it in a systematic manner. We combine three distinct components to achieve this goal: First, the robot builds a topo-metric map of its environment while cleaning. This map does not require global metric consistency, which simplifies its construction and maintenance. Second, a planner uses this map to decide where to clean and how to navigate there. Here, our robot attempts to systematically cover the floor with a meandering trajectory. Following this meandering-lane strategy constrains the planning problem, thus reducing its complexity. Finally, control automatons execute the plan by translating it into specific motor commands. Throughout this work, we emphasize visual methods to sense the robot's environment. Consequently, we can collect most of the required information from a single panoramic camera. In combination, these design choices allow our robot to function with limited onboard resources. During experiments, we found that our prototype is able to perform its cleaning task in a variety of complex environments. However, some remaining issues can reduce the robot's effectiveness or efficiency.

2.1. Introduction

In this project, our group developed a prototype for an intelligent mobile cleaning robot. Our prototype is a wheeled robot (Figure 2.1) which autonomously cleans the floor of an unknown indoor environment. Specifically, we aim to maximize the area covered by a hypothetical cleaning apparatus mounted on the robot. The robot should accomplish this task in the shortest possible travel distance and time. This increases the area that can be cleaned before the robot must recharge its batteries. Furthermore, a shorter cleaning process is less disruptive to normal human activity, and reduces wear on the robot's components.

To achieve these goals, our robot executes a systematic cleaning strategy. As part of this strategy, it creates a map of the target environment. This map enables the robot to plan and execute the actions required to solve its cleaning task. In this project, the cleaning strategy is built on the concept of meandering lanes, as shown in Figure 2.2: The robot covers uncleaned space by driving straight lanes parallel to the border of the previously-cleaned area. Note that we developed the robot's hardware and software as one combined system. Within this work, we therefore use the term *robot* for both our physical prototype and the framework which controls both this physical robot and any simulations.

However, the size, power consumption, and cost of the prototype and its components are limited: The robot should be small enough to clean within narrow spaces, for example between the legs of chairs or tables. It should also be low enough to fit under furniture such as beds or sofas. To extend the robot's battery life, the power consumption by onboard sensors and computers should be minimized. Furthermore, the prototype should be a feasible starting point for developing a future mass-production model. These requirements influence the design of our robot's components: In general, we prefer solutions and heuristics that are comparatively easy to design and implement, even if the cleaning performance is somewhat reduced. This should lower overall resource requirements

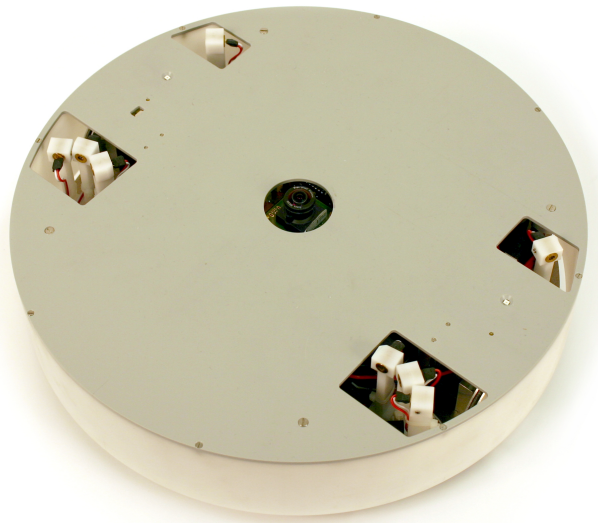


Figure 2.1.: Our cleaning robot prototype, with its outer cover installed. Our robot measures the movement of this spring-mounted cover to detect collisions (Section 2.3.2.3). The panoramic camera can be seen through a hole in the center. Furthermore, eight white pillars extend above the cover's upper lid. Each pillar contains a laser diode, which is used for obstacle detection (Section 2.3.2.2).

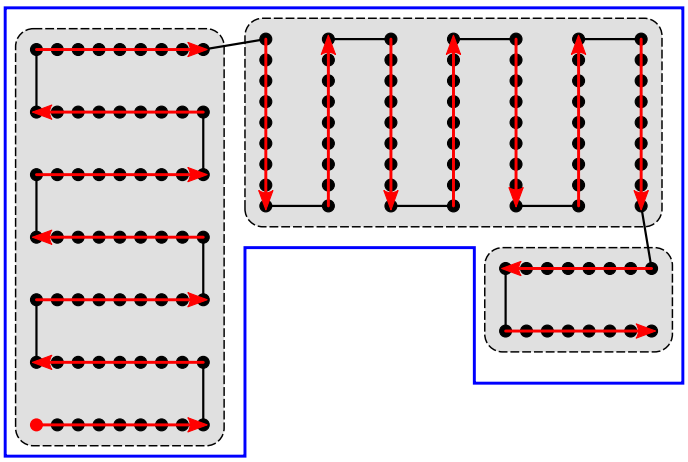


Figure 2.2.: In this idealized illustration, our robot has covered a room (blue lines) with meandering lanes. Starting from the bottom left (red dot), our robot drove a series of parallel lanes (red arrows). The robot's trajectory between the lanes is shown as a black line. Within the robot's map, these lanes consist of nodes (black dots), which are placed at regular intervals. Adjacent parallel lanes form a part, here represented by gray boxes. Note that every node belongs to a lane, and every lane belongs to a part. Thus, the robot's map consists of a part-lane-node hierarchy.

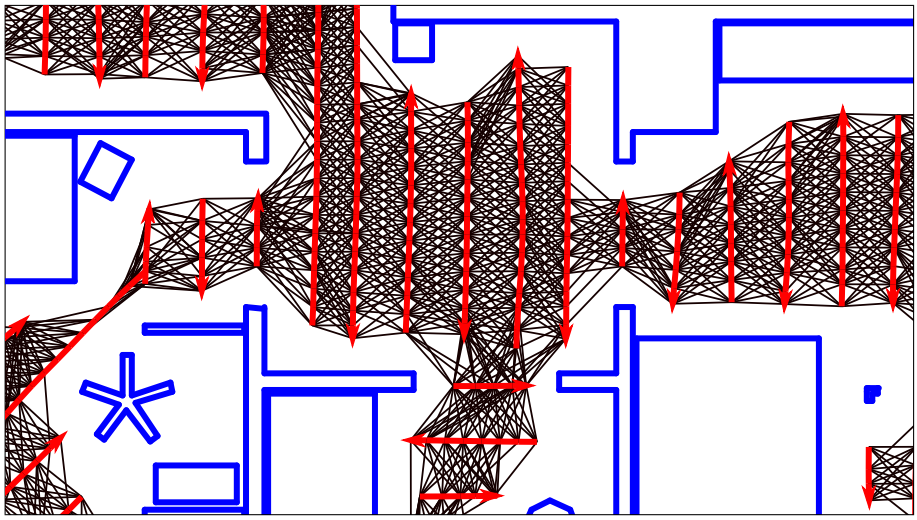


Figure 2.3.: The topo-metric map graph generated by our cleaning robot while cleaning a simulated apartment. Blue lines denote obstacles which restrict the floor space accessible to the robot. Unlike the idealized illustration in Figure 2.2, this figure shows a section of the actual map generated by our framework. The meandering lanes are drawn as red lines, while black lines represent the edges between nodes. For the sake of clarity, we have omitted the actual map nodes, which are located at the intersections between edges.

and system complexity.[1] Additionally, all computations must be fast enough to run in real-time on an onboard low-power CPU. Due to their space, power and cost requirements, we also want to limit the number of sensors. This makes cameras highly suitable, as they are parsimonious, compact, low-power and fairly cheap. We therefore favor visual techniques that operate on monochrome, panoramic images.

2.1.1. System Overview

To execute a systematic cleaning strategy, the robot requires a map of its environment. For this project, we use a topological map with local metric information. Within the graph of this topo-metric map, nodes represent locations previously visited by the robot. Neighboring nodes are connected by edges if the robot can travel directly between them. Figure 2.3 gives an example for the resulting map graph. Each node contains a camera image captured at that location, as well as a metric position estimate. In addition, we use a time-indexed obstacle map for free-space detection and collision avoidance.

Since these maps do not guarantee global metric consistency, metric position estimates are only accurate relative to nearby locations. In contrast, the relative position between distant locations may contain larger errors. This greatly simplifies map construction, since we do not need to maintain global metric consistency. Specifically, we only estimate the metric position of a node when adding it to the map. Thus, we do not need to correct these position estimates based on sensor data received later on. Our prototype uses panoramic camera images for localization within the topo-metric map: Based on a newly-captured image, the robot visually estimates its pose relative to nearby map nodes. It then corrects its estimated metric position based on these relative poses. Lacking global metric consistency, this estimate is only accurate relative to nearby nodes and obstacles.

The robot extends this map through straight, meandering *lanes*, as shown in Figure 2.2: Each lane is followed by a parallel one in the opposite direction, thus achieving gapless cleaning. In the

[1] As the developers of the successful Roomba cleaning robot have noted: “Complexity kills.” [75]

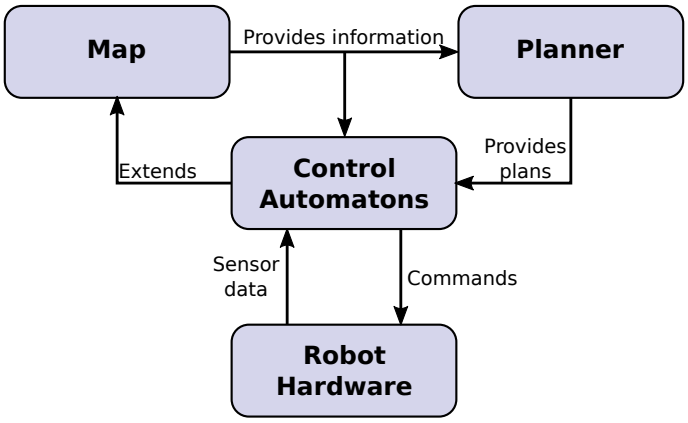


Figure 2.4.: The main interactions between the core components of our cleaning robot framework. Based on information from the map (Section 2.3), a planner (Section 2.4) chooses the robot's next action. Control automatons (Section 2.5) execute this action by generating commands for the robot's hardware (Section 2.2). These commands take into account the most recent information from the map. The automatons also process sensor data, and use it to extend the map.

topo-metric map, lanes are stored as nodes placed at regular intervals. Images from these nodes are used for localization while driving subsequent lanes. A set of uninterrupted meandering lanes forms a *part*, which is extended until no more lanes can be added to it. The robot then adds a new part by driving a lane in parallel to an existing part's border. To reach a lane's starting position, the robot navigates through the previously-covered area using the topo-metric map. If no further parts can be added, the robot returns to its home location. In practice, these actions are executed by a system of automatons that generate the appropriate motor commands.

2.1.2. Subsequent Sections

We discuss our cleaning-robot prototype throughout the following sections: In Section 2.1.4, we give a brief overview on existing cleaning robot projects, both commercial and academic. Next, we introduce our prototype's hardware in Section 2.2. The subsequent sections deal with the three main components of our software framework, as shown in Figure 2.4: Section 2.3 describes both the topo-metric and time-indexed obstacle map constructed by our robot. We also explain how obstacle checks are performed within this map. In Section 2.4, we consider how the robot plans its cleaning run. This includes the covering of uncleaned floor space, as well as planning paths through previously-mapped areas. The automatons in Section 2.5 translate these plans into robot movements. These automatons may also make limited adjustments to the planned actions. Controlling the robot involves self-localization, which we also introduce in this section. Section 2.6 contains some of the results achieved by our robot in both simulated and real-world environments. Finally, we discuss the performance of our prototype in Section 2.7.

In this chapter, we will focus primarily on the core components governing the robot's behavior. We also emphasize aspects that are important in subsequent chapters, or that contain major contributions by the author. In return, we omit some details from both the design and implementation of our prototype.

2.1.3. Other Contributors

This cleaning-robot research project grew from the continuous efforts of many contributors. The author's contributions built upon a pre-existing, initial version of the system. This earlier prototype

implemented the basic cleaning functionality, covering an unknown area by driving meandering lanes. It also contained basic versions of the map, planner and control mechanisms. However, this earlier prototype was controlled by an external laptop, and thus required a constant wireless network connection. Earlier contributors also designed and constructed the physical robot used in this work. Additionally, they introduced the robot simulator utilized in some of the experiments. In the rest of this chapter, we treat the cleaning robot framework and prototype as one cohesive system. We have therefore added this section to differentiate the contributions of the author from those made by others.

Regarding the robot's map (Section 2.3), the author greatly extended the obstacle map described in Section 2.3.2. This includes the entire obstacle-checking mechanism from Section 2.3.2.1. The author also integrated the pre-existing contact sensor (Section 2.3.2.3) into the robot's control framework. For the planning component (Section 2.4), the author made minor improvements to the path planner (Section 2.4.1). Furthermore, the lane planner (Section 2.4.2) was extensively redesigned, incorporating major changes. This included an enhanced method for generating lane candidates (Section 2.4.2.1). The author also added a novel lane-selection scheme (Section 2.4.2.2) and the local lane adjustments from Section 2.4.2.3. Within the robot control mechanism (Section 2.5), collision handling (Section 2.5.3) was developed from scratch. Moreover, the path-following automaton (Section 2.5.4) was improved. Most importantly, the robot gained the ability to circumnavigate unexpected obstacles (Section 2.5.4.2). Additionally, the author contributed improvements to the automaton for driving new lanes (Section 2.5.5). This includes the driving of piercing lanes to clean through narrow openings (Section 2.5.5.2).

2.1.4. Related Works

Both academic and commercial projects have studied autonomous floor-cleaning robots. Commercial models are now widely successful, with numerous products available on the consumer market. In general, in-depth technical details of these robots are not known to the public. This section is based on examinations of and preliminary experiments with several commercial models (not presented here), and by necessity remains somewhat speculative.

In general, commercial domestic cleaning robots can be categorized into two generations. First-generation models do not construct comprehensive maps of the environment, and thus cannot plan for complete coverage. Instead, such robots tend to rely on random walks to traverse the cleanable space. However, a first-generation model may still adjust its behavior to the environment. For example, it may perform a different type of random walk within narrow spaces. Early members of the iRobot Roomba family are a prominent example of the first generation. Additionally, Prassler et al. [120] provide a historical overview on the first commercial cleaning robots. Although they lack a systematic strategy, these first-generation robots nevertheless required a sizable research effort. For example, developing the first Roomba model from an initial prototype took a team of eight people three years [75].

In contrast, second-generation cleaning robots construct maps and plans to achieve a complete coverage. Examples include Miele's Scout RX1, Vorwerk's Kobold VR200, Dyson's 360 Eye, as well as Samsung's Navibot and LG's Hom-Bot series. Although not a commercial product, our own cleaning-robot prototype also belongs to this group. Complete coverage entails exploration and mapping of a previously unknown environment. The robot must also be able to localize itself within the resulting map. This commonly involves path integration through wheel odometry and inertial measurement units (IMUs). These sensors are subject to drift, which introduces errors in the estimated robot position. Robots from the second generation commonly correct these errors through exteroceptive sensors. These sensors include ceiling cameras (Miele Scout RX1, Samsung Navibot Silencio, LG Hom-Bot 2.0), laser scanners (Vorwerk Kobold VR200), or panoramic cameras (Dyson 360 Eye, as well as our prototype). To detect free space, these robots also carry long-ranged

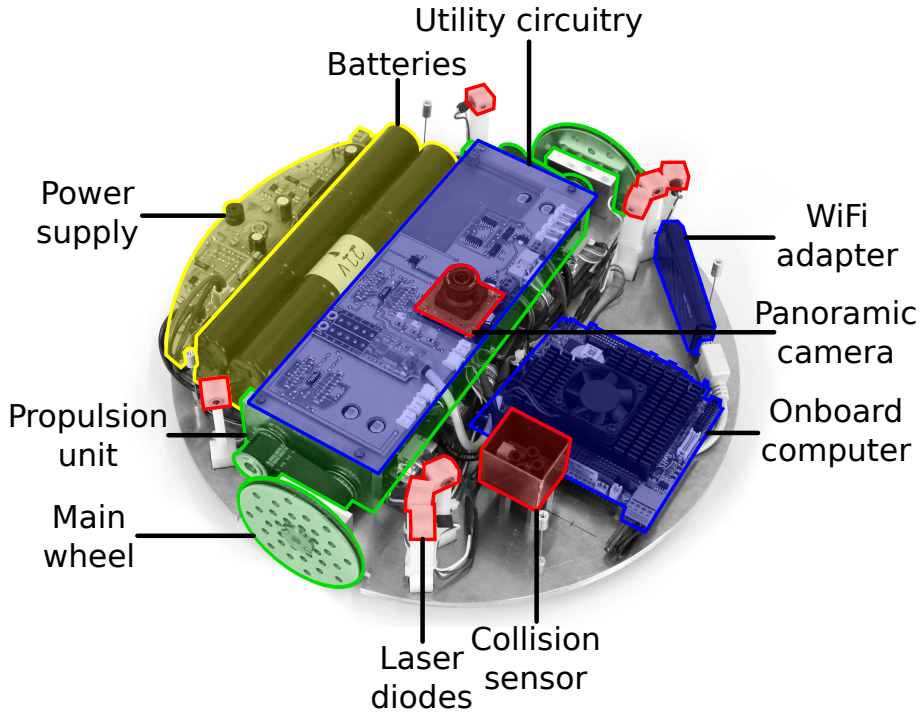


Figure 2.5.: Our robot prototype from Figure 2.1, with its outer cover removed. The core components are labeled and colored according to their function: A propulsion unit (green) contains the motors and motor controllers, which drive the side-mounted main wheels. Sensors and related parts are shown in red, with the robot's panoramic camera visible in the center. Computation and processing components are highlighted in blue, and include the onboard computer installed near the front. A power supply and battery provide the necessary electrical energy, and are colored yellow.

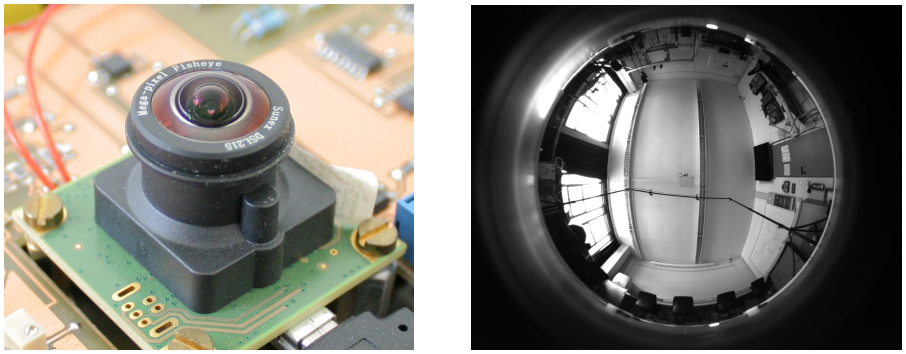
obstacle sensors, such as laser rangefinders.

The academic literature covers many problems which are relevant to cleaning robots, such as localization, mapping, navigation, and coverage planning. Due to the large number of such publications, we do not include a comprehensive survey within this introductory chapter. Please refer to the work by Hillen [67] for an overview on these topics within the cleaning-robot context. In subsequent chapters, we will examine selected parts of the literature in depth. Here, we focus on autonomous robots that systematically clean the floor in unknown, real-world environments with multiple rooms. However, we are not aware of any academic publications that demonstrate such a fully-featured cleaning robot for domestic use.

2.2. Robot Hardware

In general, our physical robot follows the design commonly found in commercial models. However, our prototype contains no actual cleaning equipment, such as brushes or suction units. Its internal components can be roughly subdivided into three major sections: Structure & propulsion, computation & sensors, and power supply. Refer to Figure 2.5 for an annotated overview of these components.

Our robot is built on top of a circular base plate, which provides rigidity and carries the other components. This plate is supported by three wheels: Two large, powered wheels on opposite sides of the robot, and an unpowered ball caster which supports the rear end. The main wheels are powered by two independent motors, allowing for differential drive propulsion. Due to its circular



(a) Camera and lens **(b)** Panoramic image

Figure 2.6.: Figure (**a**) shows the panoramic camera and lens mounted above the center of our robot. (**b**) is a typical example for the images taken by this camera. It contains the entire hemisphere above our robot in equidistant projection.

footprint, our robot can turn on the spot without risking a collision; this simplifies motion planning. However, a D-shaped robot with a square front end may be more suitable for cleaning corners. The internal components are protected by a cylindrical cover with a flat lid (Figure 2.1). Our robot also uses this spring-mounted cover to detect collisions with obstacles (Section 2.3.2.3). With its cover installed, the robot has an outer radius of $r_r = 17.3\,\mathrm{cm}$. The total height from the floor to the top of the lid is $h_r \approx 8.5\,\mathrm{cm}$. This relatively low profile allows our prototype to drive underneath some furniture.

The robot is controlled by an onboard embedded computer, which contains an Intel Atom N2600 1.6 GHz dual-core CPU. This computer is connected to the motor controllers via an RS-232 serial interface, enabling it to move the main wheels. Using a wireless network interface, the computer can also transmit real-time telemetry data. To execute its cleaning strategy, our robot also carries several sensors. Most importantly, the robot carries a uEye UI-1246LE-M-HQ monochrome CMOS camera (Figure 2.6a). This camera is located at the robot's center; thus its position remains fixed if the robot turns on the spot. Together with a Sunex DSL215 fisheye lens, the camera provides a $\approx 185°$ field of view in equidistant projection. The camera is mounted vertically, with the optical axis facing upwards. Its projection center lies $\approx 9\,\mathrm{cm}$ above the ground, and thus slightly above the robot's body. Consequently, the entire hemisphere above the robot is visible in the images, as seen in Figure 2.6b. The camera has a maximum resolution of 1280×1024 pixels, with a depth of 8 bits per pixel. It also contains a $650\,\mathrm{nm}$ infrared cutoff filter. This camera serves as the robot's main sensor, and provides most of the required information. Our prototype also carries passive laser diodes and an electro-optical contact sensor for collision detection. We describe these in more details while discussing obstacle detection in Section 2.3.2.2 and Section 2.3.2.3. Furthermore, the robot uses encoders to measure the rotations of its main wheels. As we explain in Section 2.5.2, this information is used during self-localization.

To allow for independent operation, our robot prototype is powered by lithium polymer batteries. We carry two independent batteries, one for the propulsion system and another for the computer and sensors. A purpose-built power supply provides each component with power at the required voltages. For stationary tests, the robot can also be connected to an external supply via a cable.

2.3. Mapping

Our prototype maintains both a topo-metric map and a time-indexed obstacle map (TIOM). The topo-metric map (Section 2.3.1) consists of a map graph annotated with metric information. While

metric position estimates within the map share one global coordinate system, they are only consistent relative to nearby locations in the map graph. This map is mainly used for localization, navigation and planning. The TIOM (Section 2.3.2) stores obstacles detected by the robot. This information is used for free-space detection during planning and to avoid collisions. However, the obstacle information is not used for robot self-localization.

2.3.1. Topo-Metric Map

The topo-metric map is an undirected graph that consists of nodes connected by edges. Each node represents a distinct location within the robot's environment. An edge connecting two neighboring nodes indicates that direct travel between the corresponding locations is possible. Every node contains a panoramic image captured at that location, as well as an estimate of its metric position. Lacking global metric consistency, this estimate is only accurate relative to other nearby locations. Consequently, the relative metric positions between two far-away nodes may be incorrect. The robot also stores the set of timestamps at which each node was created or revisited. These timestamps connect the topo-metric map graph with the time-indexed obstacle map from Section 2.3.2.

As shown in Figure 2.2, the nodes also form the lowest level in a hierarchy of parts, lanes and nodes. A lane consists of several nodes in an uninterrupted, straight line. Similarly, an uninterrupted set of meandering, parallel lanes forms a part. Lanes and parts are attached to each other, resulting in a secondary map graph. This hierarchy is primarily used for planning, and is also stored within the topo-metric map.

2.3.2. Time-Indexed Obstacle Map

The time-indexed obstacle map (TIOM) contains all obstacles detected by the robot. Our robot uses this map to detect free space for cleaning, and for collision avoidance while moving. Obstacles are represented as points in a global Cartesian coordinate system. The map combines obstacle points from all sensors, without considering their sensor of origin. Instead, the confidence in any given point is expressed through a weight factor. To calculate an obstacle's position $\vec{o}_i$, we combine the robot-relative obstacle location with the estimated robot position $\vec{p}_r$. As we later discuss in Section 2.5.2, $\vec{p}_r$ is not globally consistent. For this reason, the positions $\vec{o}_i$ are only accurate in the vicinity of the robot position from which the obstacle was detected.

Each obstacle point i in the TIOM is labeled with a timestamp τ_i, specifying the time at which it was detected. We also record the set of timestamps $T_k = \{t_{k,l}\}$ at which the robot created or revisited the topo-metric map node k; here, l serves as an index for the multiple timestamps in T_k. Given a short time span ϵ_τ, all obstacle points $\vec{o}_i$ with

$$(\exists t_{k,l} \in T_k)[|t_{k,l} - \tau_i| < \epsilon_\tau] \tag{2.1}$$

were therefore detected shortly before or after the robot was located at k. Since our robot moves at a finite speed, it can travel only a limited distance within the time span ϵ_τ. Thus, any obstacle point i that satisfies Equation (2.1) must have been detected while the robot was close to the node k. The positions of these obstacle points form the set

$$O_k = O_{T_k} = \{\vec{o}_i | (\exists t_{k,l} \in T_k)[|t_{k,l} - \tau_i| < \epsilon_\tau]\}\,. \tag{2.2}$$

In our implementation, we use the robot's main-loop cycle count as the timestamp. One main-loop iteration requires $\approx 100\,\mathrm{ms}$, and our prototype uses $\epsilon_\tau = 70$. Thus O_k contains the positions of all obstacle points detected within $\approx 7\,\mathrm{s}$ of creating or visiting the node k.

The robot may also detect obstacles in the vicinity of k without actually visiting the node k. This most commonly occurs if the robot visits another node k' that is close to k. The obstacle set $O_{k'}$

may contain some points that are missing from O_k. To reduce the risk of overlooking an obstacle, we now combine O_k with the sets $O_{k'}$ from all nearby nodes k': First, we construct the *joined* timestamp set

$$T'_k = T_k \cup \left(\bigcup \left\{ T_{k'} \middle| \left(\mathrm{part}(k) = \mathrm{part}(k') \right) \wedge \left(d(k,k') < \epsilon_d \right) \right\} \right), \tag{2.3}$$

which combines the timestamps from the nodes k' close to k. Here, $\mathrm{part}(k)$ is the map part that contains the node k. $d(k,k')$ is the Euclidean distance between the estimated positions of the nodes k and k'. Since k and k' belong to the same part, this distance measure is sufficiently accurate. We limit the distance $d(k,k')$ to $\epsilon_d = 0.5\,\mathrm{m}$; this should offer a fair compromise between obstacle consistency and the area from which obstacle points are combined.

Next, we speed up subsequent steps by reducing the size of T'_k. First, we sort T'_k in descending order, resulting in the sorted timestamp list

$$T'_k = \left\{ t'_{k,1}, t'_{k,2}, ..., t'_{k,L} \right\}. \tag{2.4}$$

Iterating over an index l, we first remove any timestamp $t'_{k,l}$ from T'_k for which $t'_{k,l-1} - t'_{k,l} < 10$. Since $t'_{k,l-1}$ and $t'_{k,l}$ are less than $\approx 1\,\mathrm{s}$ apart, the corresponding time intervals $(t'_{k,l} - \epsilon_\tau, t'_{k,l} + \epsilon_\tau)$ and $(t'_{k,l-1} - \epsilon_\tau, t'_{k,l-1} + \epsilon_\tau)$ are similar, thus making $t'_{k,l}$ redundant. Afterwards, we truncate the remaining T'_k to the fifty most recent timestamps. While this may exclude some older obstacle points, it also keeps T'_k from growing impractically large over time.

Finally, we construct the obstacle set $O'_k = O_{T'_k}$ from T'_k using Equation (2.2). To reduce the time spent on this process, we store the final T'_k in a cache. We invalidate this cache entry if the robot creates or visits a nearby node k' with

$$\left(\mathrm{part}(k) = \mathrm{part}(l') \right) \wedge \left(d(k,l') < \epsilon_d \right). \tag{2.5}$$

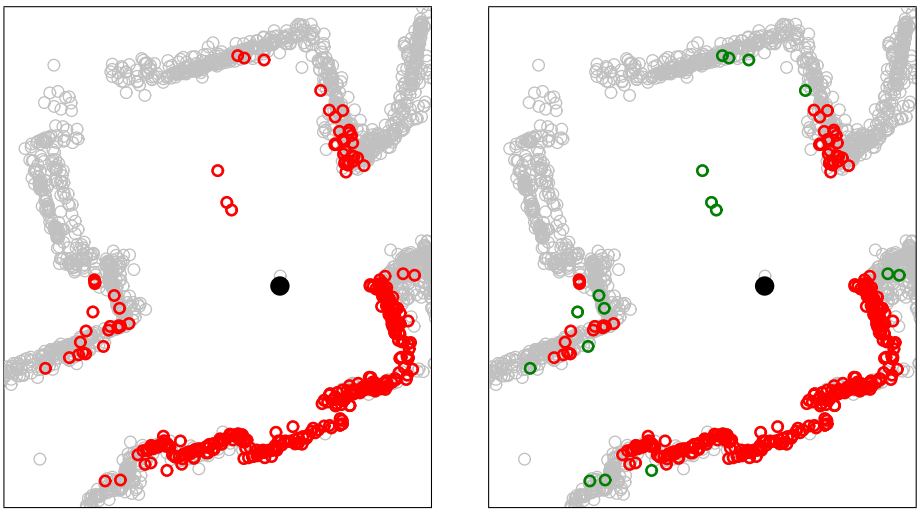
In this case, we reconstruct T'_k the next time the local obstacle set O'_k is needed.

2.3.2.1. Obstacle Checking

We can use the TIOM to check for obstacles along a given path. In general, we assume that the robot only moves along a straight line. We approximate curved movements by multiple straight-line segments.

To check the map for obstacles along a given line segment, we perform three steps: First, we construct the timestamp list T_K from the node set K. Depending on the robot's current task, K consists of one or more reference nodes. For example, our robot may be following a path through the map by moving from node to node. Here, K contains the node along the path which the robot is currently approaching. To include the most recent obstacle data, we also add the current timestamp to T_K. While the robot is extending the map by driving a new lane, K consists of the most recent node added. If K includes multiple nodes k, we combine the individual timestamp lists T'_k to form $T_K = \bigcup_{k \in K} T'_k$. The robot then extracts the set of local obstacles $O = O_{T_K}$ according to Equation (2.2).

O may contain isolated obstacle points from incorrect sensor readings. In a second step, we therefore apply DBSCAN clustering [39] to the obstacle points $\vec{o} \in O$. This method identifies obstacle clusters with a high weight density. Obstacle points that do not belong to any such high-density cluster are eliminated from O, resulting in the set $\hat{O}$. As shown in Figure 2.7, this removes these isolated obstacle points. For the minimum cluster density, we require a summed weight of at least $w_{\mathrm{min}} = 2.0$ within a radius of $r_o = 10\,\mathrm{cm}$. With typical obstacle weights, this corresponds to clusters of 3 or more obstacle points.



(a) Local obstacle points **(b)** After applying DBSCAN

Figure 2.7.: We use DBSCAN clustering to reject isolated obstacle points. Example (**a**) shows the local obstacle map O'_k (red circles) for a single map node k (black circle) in a real-world hallway. For context, we also include other obstacle points (gray circles) which are not part of O'_k. Some of the points in the middle of the hallway originate from incorrect range-sensor measurements. In (**b**), we have identified isolated obstacle points (green circles) using DBSCAN. The robot subsequently ignores these points, thus excluding the incorrect range-sensor readings. However, we also reject a few points which correspond to actual walls near the edge of the local obstacle map.

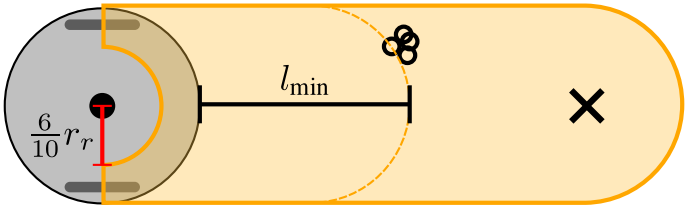


Figure 2.8.: An illustration of our obstacle-checking process. Here, the robot intends to move to the cross-marked location on the right. For this planned movement, we use our map to check the orange area for obstacles. This area encompasses the entire space covered by the robot's external cover, assuming movement in a straight line. Note that we do not check for obstacles within $0.6r_r$ (red line) of the starting position: Since the robot cannot physically overlap with an obstacle, points within this radius are probably incorrect. In this example, the planned path is blocked by a small cluster of obstacle points (black rings). The obstacle limits the distance our robot can travel, as shown by the dashed orange line. This line corresponds to the robot's leading edge after moving the maximum obstacle-free distance l_{min}.

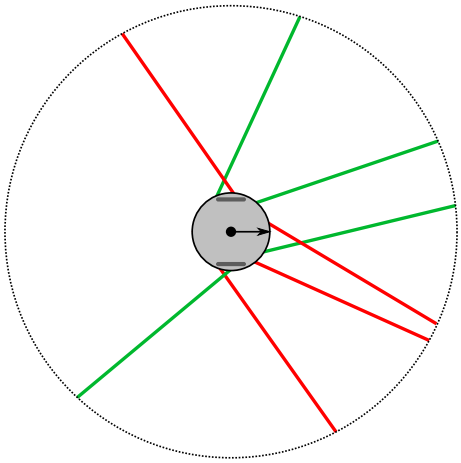


Figure 2.9.: The laser beams used for obstacle detection on our prototype robot, illustrated as colored lines. An arrow on our robot corresponds to the forward movement direction. To prevent ambiguities between overlapping beams, only lasers shown in the same color are active at any given time. Our obstacle range measurements are less accurate at longer ranges. We therefore limit the obstacle-detection range to $1\,\mathrm{m}$, as indicated by the dotted circle.

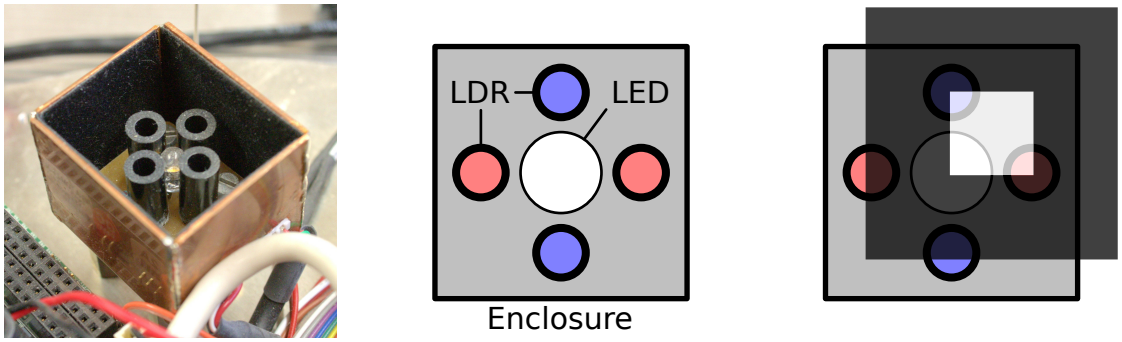
Finally, we check $\hat{O}$ for obstacle points in the robot's path. Specifically, we check the area covered by the robot during a planned straight movement, as shown in Figure 2.8. The length of this area is equal to the planned movement distance l, while the width is twice the robot radius $r_r = 17.3\,\mathrm{cm}$. Since the obstacles in $\hat{O}$ belong to dense clusters, we consider the area blocked if it contains any point $\vec{o} \in \hat{O}$. We also determine the maximum obstacle-free distance l_{min}, shown in Figure 2.8. This distance is useful when evaluating candidates for future cleaning lanes. Note that we ignore obstacles within $0.6r_r$ of the movement's starting position: At the beginning of the movement, the robot already occupy this location. Consequently, no obstacles can actually be present at that position.

Our path-following automaton (Section 2.5.4) also requires a high-speed check for nearby obstacles. Here, we merely consider obstacle points recorded in the last $\epsilon_\tau = 70$ main loop iterations. From these most recent points, we select those within $r_r + 10\,\mathrm{cm}$ around the estimated robot position. Our heuristic detects an obstacle if the total weight of these recent, close-by points exceeds $w_{\text{min}} = 2.0$. Note that this heuristic only considers obstacles within a compact volume, and thus omits the DBSCAN clustering step.

2.3.2.2. Laser Ranging

As we have discussed previously, we want to utilize the robot's camera as much as possible. By mounting laser diodes on our robot (Figure 2.5), we can use the camera to detect distant obstacles. In keeping with our design goals, these lasers are simple, cheap, compact, low-power, and require no computational resources. The diodes emit laser beams that radiate away from the robot, as illustrated in Figure 2.9. If one of the beam hits an opaque obstacle, some of the light is reflected as a bright dot. Our camera captures underexposed images at regular intervals, which clearly show these dots against a dark background. Using calibration data, we then calculate the range to an obstacle from the image position of each dot. The resulting obstacle point is then added to the TIOM, as described in Section 2.3.2.

We only enable these lasers while the camera captures the underexposed obstacle-detection images. They therefore emit only short pulses, and the bright dots do not appear in any other camera images. Note that some of the laser beams in Figure 2.9 cross each other. This prevents obstacles



(a) Contact sensor **(b)** Sensor layout **(c)** During a collision

Figure 2.10.: Our robot's contact sensor in (**a**) consists of an light-emitting diode (LED) surrounded by four light-dependent resistors (LDR). Each individual LDR is housed within a narrow black cylinder, and thus can only be illuminated from above. An outer enclosure coated with a dark material protects the sensor from stray light. This sensor faces a light-reflecting marker on the underside of the robot's lid. (**b**) illustrates the layout of this sensor and its major components. In (**c**), a collision has shifted the spring-mounted cover, and with it the reflective marker (which was omitted in (**b**)). Consequently, additional light from the LED reaches the top and right LDRs, while less light is reflected towards the lower and left LDR. We can estimate a collision's direction and magnitude from the resulting change in LDR resistances.

from being lost in the gaps between adjacent beams. Unfortunately, this can also cause ambiguities when associating the laser dots with their respective beams. We avoid this problem by forming two distinct subsets, each of which contains four non-crossing beams. We then activate just one of the subsets when capturing an obstacle-detection image. A circuit automatically switches between these two subsets after each image. Due to the low refresh rate, beam count, and a maximum range of $1\,\mathrm{m}$, our laser-derived obstacle data is fairly sparse. We therefore combine obstacle data from multiple readings taken across a larger area, as explained in Section 2.3.2.1.

2.3.2.3. Contact Sensor

In practice, the laser-based detection from Section 2.3.2.2 may fail for some obstacles. Examples include a transparent glass door, furniture with a laser-deflecting chrome finish, or a light-absorbing dark velvet curtain. For this reason, our robot carries an additional sensor that detects actual collisions with obstacles. This electro-optical contact sensor (Figure 2.10) measures relative movements of the robot's cover. Since this cover is mounted on springs, a collision with an obstacle causes it to shift. The underside of the cover's lid carries a reflective marker, which is illuminated by a light-emitting diode (LED). Four light-dependent resistors (LDR) capture the light reflected by this marker. In Figure 2.10c, shifting the cover and marker changes the amount of light that reaches each LDR. We measure the resulting changes in resistance through analog-digital converters. From these measurements, we can estimate the direction and strength of the collision. Unlike ordinary contact switches, this sensor thus continuously measures the collision direction and magnitude. In practice, we found the contactless electro-optical design to be very reliable.

To make measurements more accurate, we calibrate the sensor in two different ways. Before each cleaning run, the stationary robot takes multiple sensor readings while clear of obstacles. Averaging these measurements gives us the sensor output for the cover at rest. We then measure any displacements relative to this resting position. In practice, the cover's movement direction may differ from the collision direction. For example, friction with an obstacle may cause the cover to be dragged or rotated.

We can reduce this problem through a second, optional calibration step: First, the robot is made to collide with an obstacle at predetermined angles. For each angle, we store both the real and measured collision direction. Next, we create a mapping between these real and measured directions through linear interpolation. During normal operation, we use this mapping to estimate the calibrated collision direction. However, factors such impact speed and obstacle material also affect the cover's movement in a collision. Since our robot cannot measure these factors, the calibrated direction still contains some residual errors.

2.4. Planning

To fulfill its cleaning task, the robot needs to plan its future movements. Based on the map from Section 2.3, we recognize two major planning problems: Section 2.4.1 describes how the robot plans paths through the existing map. To extend this map, the robot also plans new parts and lanes according to Section 2.4.2.

To reduce the complexity of planning, our robot employs a two-stage approach: In this section, we discuss how the planners create coarse, high-level plans. The control automatons (Section 2.5) can then modify or improve the active plan while executing it. For example, the path planner provides a sequence of edge-linked map nodes which the robot should follow. Yet the path-following automaton may deviate from this path to avoid unexpected obstacles. Such deviations are purely local, and are not added to the high-level plan. Thus the robot can make fine-grained adjustments based on local conditions while keeping the overall plan simple.

2.4.1. Path Planning

We apply the well-known shortest-path algorithm by Dijkstra [37] to plan paths between two nodes of the topo-metric map. The result is a path $P = \{i_0, i_1, ..., i_N\}$ that begins with the node i_0 and ends with the destination node i_N. All subsequent node pairs (i_k, i_{k+1}) are connected by an edge; the robot can therefore follow the path by directly traveling from node to node. In general, the cost of traversing an edge (i, j) is $w_{i,j} = \|\vec{p}_i - \vec{p}_j\|$, where $\vec{p}_i$ is the estimated metric position of the node i. Dijkstra's algorithm identifies the shortest path, which minimizes the total cost

$$S = \sum_{k=0}^{N-1} w_{k,k+1}. \tag{2.6}$$

However, this shortest path may lead the robot through narrow spaces between obstacles. In practice, following such a difficult path can take additional time, and a longer path with greater obstacle clearance may thus be more efficient. For this reason, we plan paths with a modified cost function

$$S' = \sum_{k=0}^{N-1} w_{k,k+1} c_{k+1}. \tag{2.7}$$

For a path that is free of close obstacles, the obstacle-based cost factors are $c_1 = c_2 = ... = c_N = 1$, and thus $S' = S$. Obstacles in close proximity to the node position $\vec{p}_i$ increase the value of c_i, with closer obstacles resulting in higher values. By choosing the path that minimizes S' instead of S, we cause the planner to prefer paths through wide-open spaces.

At times, the robot may need to move to an arbitrary position $\vec{p}$ that does not coincide with any existing map node. In this case, we first navigate to a nearby node i_N, and then approach $\vec{p}$ from $\vec{p}_{i_N}$. For example, when traveling to the start of a new lane, i_N is the existing node to which the lane is attached; this node is determined during lane planning (Section 2.4.2.1). Similarly, the robot may start at a position $\vec{p'} \neq \vec{p}_i \; \forall i$. Here, the robot first moves from $\vec{p'}$ to the position $\vec{p}_{i_0}$ of the first

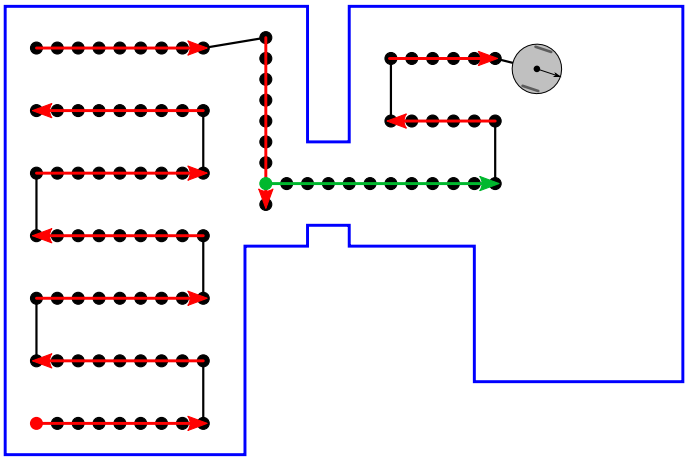


Figure 2.11.: This illustration shows a situation similar to Figure 2.2: Starting from the bottom left (red dot), the robot began to cover a room (blue lines) with meandering lanes (red arrows). Near the center of the room, the robot encountered a narrow passageway. Since the opening is too small to contain a regular meandering lane, our robot traverses it using a piercing lane. This piercing lane (green arrow) extends away from a position on an existing lane (green dot). Afterwards, the robot continues to clean by attaching a part of meandering lanes to the piercing lane.

node, and then follows the planned path. In this situation, we commonly choose i_0 to be the most recent node created or visited by the robot.

2.4.2. Part-Lane Planning

As shown in Figure 2.2, our robot covers uncleaned areas using meandering lanes. This is an efficient strategy, as there are neither gaps nor overlaps between properly-spaced lanes. Since these lanes run parallel to a boundary of the map, the robot always remains close to existing nodes. Using the visual localization method from Section 2.5.2, our robot can estimate its pose relative to these nodes. This way, the robot avoids getting lost while cleaning and mapping unknown areas. We also found that straight lanes are comparatively simple to planning and drive.

If no obstacles are present, these meandering lanes form a rectangular part. This closely matches the rectangular room shape commonly found in indoor environments. By attaching multiple parts to each other, we can cover large, complex layouts, as in Figure 2.2. Finally, we hope that human observers will regard such straight, parallel lanes to be systematic and well-organized. In practice, it may not be feasible to cover narrow or cluttered areas using parts of meandering lanes. We therefore incorporate occasional deviations from this idealized strategy. For example, the robot may extend the map through narrow spaces using a so-called piercing lane. As shown in Figure 2.11, these lanes are not parallel to a map border, instead pointing away from the map. To benefit from the advantages of parallel lanes, we will only use these piercing lanes sparingly.

To perform this cleaning strategy, the robot must plan the lanes and parts it should drive. In keeping with our design goals, we chose a straight-forward approach to planning: Since our robot has no pre-existing map, we cannot construct a cleaning plan in advance. Instead, we base our plans on the information added to the map while cleaning. Since our robot's sensor ranges are finite, we have only limited knowledge about the environment beyond the previously-cleaned area. Furthermore, our robot does not construct a globally consistent map of free, uncleaned space. We therefore do not attempt to create complex plans that contain many future parts and lanes. Building such a long-term plan would also be computationally expensive. This effort may be wasted if new information later invalidates the existing plan.

Instead, our robot plans only a single lane ahead: When extending the current part, the planner chooses merely the next meandering lane. Similarly, the robot plans a new part by deciding on the part's first lane. The planner then extends the new part by adding one lane at a time. In either case, we do not explicitly include subsequent lanes in our plan. This lean planning approach is fast, and requires only limited knowledge beyond the previously-covered area. However, we cannot guarantee an optimal cleaning of the whole environment by considering only one lane at a time. Instead, we select lightweight heuristics that lead to good overall cleaning results. Even so, solving this planning problem with limited computational resources was a significant aspect of the project.

To select the next lane, we perform three distinct steps: First, the planner inspects the map's border regions to generate a list of lane candidates. Next, it uses a quality criterion to select one of these candidates as the next lane. Finally, we apply an optional refining step to improve the chosen lane. We describe these three steps in Section 2.4.2.1, Section 2.4.2.2, and Section 2.4.2.3, respectively. The planning process generates a lane description, which is passed to the control automatons. To clean along the planned lane, our robot then moves to the planned starting position (Section 2.5.4). From there, it drives the planned lane according to Section 2.5.5.

2.4.2.1. Generating Lane Candidates

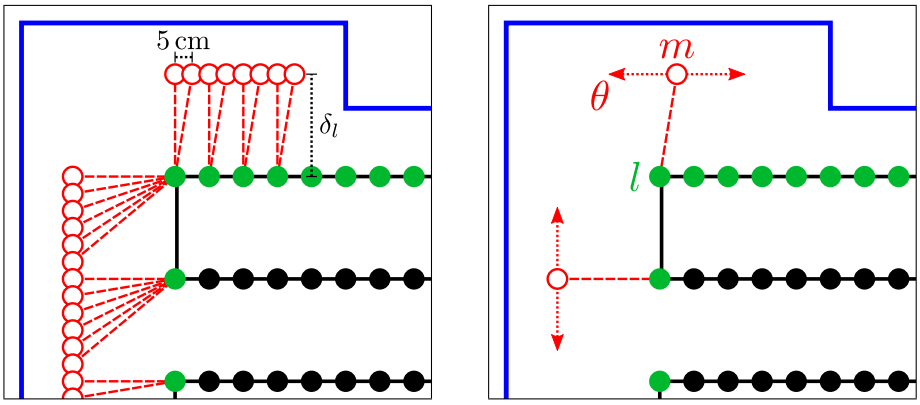
As a first step in lane planning, we generate a list of lane candidates K_i. Each of these candidates is specified by the tuple $K_i = (m_i, \theta_i, l_i, s_i)$. To drive a given lane K_i, our robot first moves to the starting position $\vec{p}_{m_i}$. Next, it creates the starting node m_i, which serves as the first node of the new lane. The robot then cleans along a straight line with direction θ_i and length s_i. Recall that new lanes must always be attached to the existing map. Each candidate therefore specifies an existing node l_i with which to connect the starting node m_i.

To generate the lane candidates, we first find all possible border nodes l within the map topology. These are nodes which are not fully surrounded by other nodes in the map graph. If the robot is trying to add another meandering lane to the current part, it will only consider nodes from the most recently completed lane. The specific border nodes are selected according to a complex decision tree, which we do not describe here. For each border node l, we then identify the various positions $\vec{p}_m$ at which a new node m could be attached. We generate these starting nodes m at $5\,\mathrm{cm}$ intervals using the scheme shown in Figure 2.12a: When planning a parallel lane, these potential nodes are placed at an intra-lane distance of $\delta_l = 30\,\mathrm{cm}$ from the map border. This ensures gapless and non-overlapping coverage for a hypothetical cleaning orifice with a width of $30\,\mathrm{cm}$. In contrast, piercing lanes extend directly away from the covered area, as shown in Figure 2.11. For a piercing lane, we thus position the possible starting nodes m directly on the map's border. The precise placement of the starting nodes is governed by a heuristic, which we chose to omit here.

For each pair (l, m), the robot must be able to reach the starting node m from the border node l. We therefore eliminate (l, m) where obstacles are detected between the positions $\vec{p}_l$ and $\vec{p}_m$. To speed up this process, we flag any border node l for which no valid starting nodes m remain. This lets us ignore such border nodes in the future, reducing the planning time. Because this depends on the lane type, we use separate flags for parallel and piercing lanes. Note that changes in the map may render this information out-of-date. In this case, the planner will remove the flags and reconsider the affected border nodes.

Each starting node may be the origin of several lane candidates K_i. While these K_i share the same m_i and l_i, they differ in their cleaning direction θ_i. As seen in Figure 2.12b, these θ_i are parallel to the map border which contains the border node l_i. Piercing lanes are only used at the start of a new part, and therefore follow the same general scheme. However, these lanes do not run in parallel, and should extend into uncleaned space. We therefore plan the piercing lanes at various steep angles relative to the map border.

We now have a list of lane candidates, each with a border node l_i, starting node m_i at position



(a) Border and starting nodes **(b)** Example lane candidates

Figure 2.12.: Generating candidates for a parallel lane that begins a new part. In (**a**), we first determine the border nodes l (green dots) from the existing map nodes (black dots). Next, we attach potential starting nodes m (red circles) to these border nodes; each resulting pair (l, m) is connected by a dashed line. Due to obstacles (blue lines), some of the border nodes have no starting nodes attached to them. The nodes m are placed $\delta_l = 30\,\mathrm{cm}$ from the map border, while adjacent m are $5\,\mathrm{cm}$ apart. Each of these m now serves as the starting node for one or more lane candidates. As seen in (**b**), the direction θ (red, dotted arrows) of each candidate is parallel to the map border containing the associated border node l. For clarity, we only show the lane candidates for two of the (l, m) pairs.

$\vec{p}_{m_i}$, and direction θ_i. Next, the planner uses the map to determine the maximum length s_i for each K_i. For a parallel lane candidate K_i, this length is limited by both a topo-metric and an obstacle constraint. As we see in Figure 2.13, the topo-metric constraint consists of two parts: First, the entire length of the planned lane must be close to nodes from the part that contains l_i. This prevents gaps in the cleaned area, and ensures that the robot remains close to the previously-mapped area. It also allows us to connect the new lane to the map by inserting edges between the new and existing nodes. Additionally, our robot uses these nearby nodes to maintain its position estimate while cleaning. Second, the lane should stay clear of existing nodes to avoid cleaning the same location more than once. Given these requirements, we now find the maximum length s_i' for which K_i fulfills two conditions: Firstly, every point along the lane K_i must lie within $50\,\mathrm{cm}$ of an existing map node. Secondly, K_i must not come closer than $r_r = 17.3\,\mathrm{cm}$ to an existing node.

As shown in Figure 2.14, each lane candidate must remain free of known obstacles. We therefore use the procedure from Section 2.3.2.1 to check for obstacle points that limit the length of K_i. For this check, we combine all obstacle points from map nodes that lie within $50\,\mathrm{cm}$ of the lane candidate. Next, we determine the maximum length $s_i \leq s_i'$ for which K_i remains free of obstacles. If s_i lies below a threshold, we mark the candidate K_i as too short. For a parallel lane extending the current part, this threshold is $20\,\mathrm{cm}$. In all other cases, we use a threshold of $40\,\mathrm{cm}$. We discard such short lanes, since they are both inefficient and difficult for the robot to drive. Specifically, the robot has little opportunity to correct its position estimate when driving a short lane.

The length-limiting constraints for a piercing lane are somewhat different, as seen in Figure 2.15: A piercing lane leads the robot away from the previously-cleaned area covered by the existing map. This makes self-localization using nearby nodes more difficult. As a result, our robot's pose estimate becomes less accurate when driving a piercing lane. Since the robot is moving far into an unmapped area, the chance of being lost also increases. We therefore limit the maximum piercing-lane length to $s_i \leq 140\,\mathrm{cm}$. Similar to the parallel lanes, piercing lanes need to be at

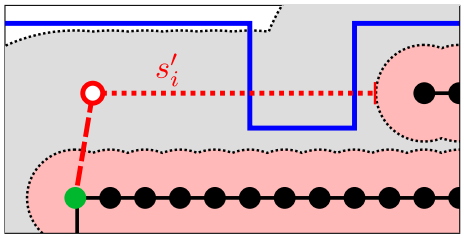


Figure 2.13.: The topo-metric constraints that limit the length of a potential parallel lane K_i. All points along a parallel lane candidate (red, dotted line) must remain within $50\,\mathrm{cm}$ of the existing map nodes (black dots). Here, the space that fulfills this constraint is marked in gray. Additionally, the lane candidate must keep a distance of $r_r = 17.3\,\mathrm{cm}$ or more from any map node. In this example, the length s_i' is thus restricted by the red areas. This constraint does not consider obstacles, allowing s_i' to extend beyond a wall (blue lines).

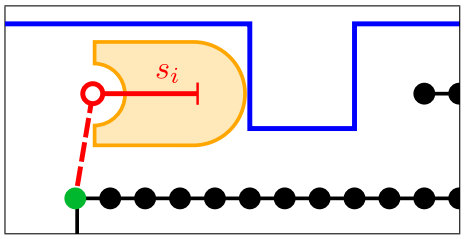


Figure 2.14.: We ensure that lane candidates remain free of known obstacles. As in Figure 2.8, the orange area represents the space covered by the robot while driving a lane candidate (solid red line). We choose the lane's length s_i so that this area is free of known obstacles (blue lines). s_i must also remain at or below the lane's length limit s_i' from Figure 2.13.

least $s_i \geq 60\,\mathrm{cm}$ long. Since these lanes point away from the map border, they do not have to remain close to any existing nodes. As with the parallel lanes, the area covered by the piercing-lane candidate must be free of obstacles. Since we prefer parallel lanes, piercing lanes should only be used to pass narrow openings. Thus the initial $60\,\mathrm{cm}$ of a piercing lane must pass within $25\,\mathrm{cm}$ of a known obstacle point. The obstacle points used for these checks are collected from the vicinity of the border node l_i.

After determining the lane lengths, we commonly find that some of the parallel lane candidates overlap. Such lanes are attached to the same map border and run in the same direction θ_i, but differ in their starting node m_i. Some of these candidates only cover an area already covered by a longer alternative. To speed up subsequent planning steps, we discard such redundant candidates using the method depicted in Figure 2.16: First, we sort the lane candidates K_i according to their direction θ_i and the border to which they are attached. Within each resulting group $\mathbf{K}$, all K_i share a common θ_i, which we call Θ'. We now sort all $K_i \in \mathbf{K}$ in ascending order, based on the location of their starting positions $\vec{p}_{m_i}$ along an axis parallel to Θ'. Finally, we iterate over the sorted $K_i \in \mathbf{K}$, starting with the rearmost $\vec{p}_{m_i}$ along the direction Θ'. We then discard any K_j for which the starting position $\vec{p}_{m_{i'}}$ is covered by a longer candidate K_i. Note that we cannot simply select the longest candidate K_i from each group: Because constraints may limit the length of the lanes, a group $\mathbf{K}$ can contain multiple non-overlapping lane candidates.

2.4.2.2. Lane Selection

The robot now needs to select one of the N lane candidates $K_1, K_2, ..., K_N$ generated above. When extending the current part with another parallel lane, we use a simple heuristic: We first determine

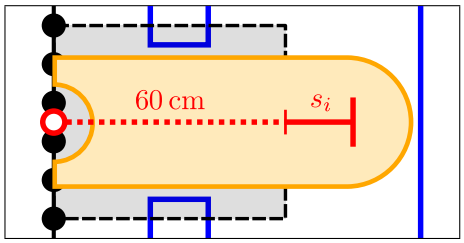


Figure 2.15.: In this illustration, we plan a piercing lane candidate (red line) through a narrow opening (blue lines). From the starting position (red circle), the lane extends away from the existing map (black dots). The first $60\,\mathrm{cm}$ of the lane (dotted line) must pass within $25\,\mathrm{cm}$ of an obstacle; here, we have marked this area in gray. This ensures that we only plan piercing lanes to pass through narrow openings. At the same time, the area covered by the lane (orange shape, see Figure 2.8) must remain free of obstacle points. The lane's length s_i must be at least $60\,\mathrm{cm}$, but may not exceed $140\,\mathrm{cm}$.

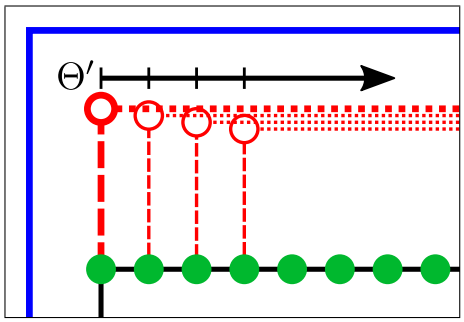


Figure 2.16.: We discard lane candidates that are mere fractions of a longer alternative. This illustration shows one group of candidates that share a common direction Θ' and map border (green nodes). Each candidate is represented by a starting position (red circle) and planned lane (red, dotted line). Note that for the sake of clarity, we show these candidates offset laterally from each other. Next, we sort the starting positions by their location along an axis with the direction Θ' (black arrow). We then check all lanes candidates in ascending order, beginning with the rearmost one (here highlighted in bold). Since the other starting positions are covered by this lane, they are redundant and will be rejected.

the maximum length $\breve{s} = \max_i s_i$ among all suitable parallel lane candidates K_i. Based on this, we then identify the set

$$\mathbf{C} = \{i | i \in [1, N] \wedge s_i \geq 0.7\breve{s}\}, \tag{2.8}$$

and select the target lane $\breve{K} = K_{\breve{i}}$ with

$$\breve{i} = \underset{i \in \mathbf{C}}{\arg\min}\, d_i. \tag{2.9}$$

Here, the geodesic distance $d_i = d(l_i)$ is the length of the shortest path from the robot's current position to the border node l_i; we calculate this length using Equation (2.6). Thus $\breve{K}$ is the closest candidate with a length that is at least $70\,\%$ of $\breve{s}$. We consider this a good compromise between driving long lanes and traveling short distances.

In contrast, selecting the first lane for a new part is somewhat more complex: These lanes can be attached to any border of the map that is adjacent to open space. Furthermore, we now have to choose between parallel and piercing lanes. We solve this problem by calculating a cost-benefit ratio $r_i = b_i / c_i$ for each candidate K_i. Our robot then simply selects the lane $\breve{K} = K_{\breve{i}}$ with

$$\breve{i} = \underset{i \in [1,N]}{\arg\max}\, r_i. \tag{2.10}$$

Planning a lane and moving to its starting location requires time and energy. We thus assume that a few long lanes are more efficient than many short lanes when cleaning a given area. Furthermore, driving a long lane gives the robot more time to update its pose estimate. Longer lanes should therefore have a higher benefit b_i. For our prototype, we use the simple heuristic

$$b_i = \log\left(\left\lfloor \frac{s_i}{10\,\mathrm{cm}} \right\rfloor + 5\right), \tag{2.11}$$

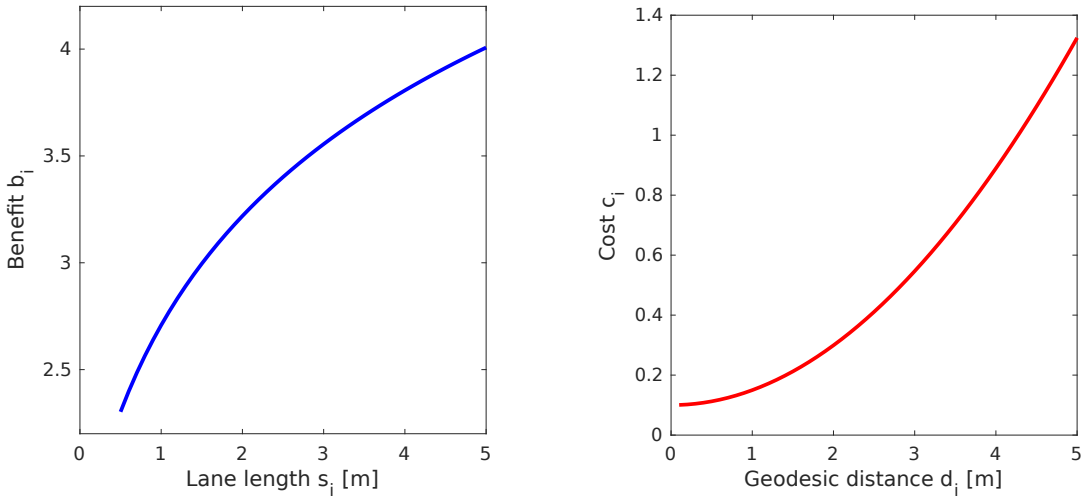
where s_i is the planned length of the candidate K_i. From Figure 2.17a, we see that this logarithmic term rewards longer lanes, without overemphasizing lanes of extreme length.

Before driving a given lane candidate, the robot must first travel to the starting position. Since the robot is moving through previously-cleaned space, this lowers the cleaning efficiency. As a heuristic, we therefore prefer nearby lane candidates. Consequently, the cost factor c_i should penalize long travel distances. Additionally, it should also discourage piercing lanes: Due to the issues discussed in Section 2.4.2.1, piercing lanes should be used sparingly. As mentioned, driving a piercing lane decreases the accuracy of our robot's pose estimate. This also increases the errors for the estimated positions of the nodes within the piercing lane itself. This reduced accuracy can also affect any parts which we later attach to the piercing lane. To keep this error from accumulating, our planner should thus avoid attaching piercing lanes to each other. For these reasons, we use the cost factor

$$c_i = \begin{cases} \left[\frac{1}{10} + d_i \tanh\left(\frac{d_i}{20\,\mathrm{m}}\right)\right] 2^{P_i} & \text{if } K_i \text{ is a piercing lane} \\ \frac{1}{10} + d_i \tanh\left(\frac{d_i}{20\,\mathrm{m}}\right) & \text{otherwise.} \end{cases} \tag{2.12}$$

Here, d_i is the travel distance to the border node l_i. From Figure 2.17b, we note that the hyperbolic tangent function ensures that c_i remains low for small travel distances d_i. For larger distances, c_i is approximately linear in d_i.

In Equation (2.12), P_i is the piercing depth of the lane candidate K_i. P_i depends on the number of piercing lanes between the first part of the map and the candidate K_i. In general, the term 2^{P_i} ensures that a piercing lane doubles the cost of all subsequent piercing lanes attached to it. This factor increases exponentially for every subsequent piercing lane: Recall that every new part is attached to an existing border node l_i. The part that contains l_i is called the *parent part* $j = \mathrm{part}(l_i)$ of the lane candidate K_i. We can represent the parts in our map as nodes within a separate part



(a) Benefit factor b_i **(b)** Cost factor c_i

Figure 2.17.: Benefit and cost factors for a lane candidate K_i, calculated from Equation (2.11) and Equation (2.12), respectively. The benefit factor b_i in (**a**) depends on the planned lane length s_i. The plot in (**b**) shows the cost factor c_i depending on the geodesic distance d_i. d_i is the length of the shortest path from the current robot position to the border node l_i. For this figure, we assume that the piercing depth from Equation (2.12) is $P_i = 0$.

graph. Within this graph, an edge indicates that one part is the parent of another. Those parts that begin with a piercing lane are called *piercing parts*. Every part k within this part graph is connected to the map's very first part by a specific path $\mathbf{P}_k$. The piercing depth $D(j)$ of the parent part j is equal to the number of piercing parts within the path $\mathbf{P}_j$. We then choose $P_i = D(j)$ for a regular lane candidate K_i, or $P_i = D(j) + 1$ if the candidate K_i itself is a piercing lane. For example, $P_{i'} = 1$ if $K_{i'}$ is a piercing lane attached to the first part in the map. If we now attach another piercing lane to $K_{i'}$, this lane $K_{i''}$ would have a piercing depth $P_{i''} = 2$. The resulting increase in c_i discourages the robot from planning long chains of piercing lanes.

As shown in Figure 2.18, we can now calculate the cost-benefit ratio r_i from b_i and c_i. For large maps, generating and selecting lane candidates in this manner may require a long time. From Figure 2.18, we note that far-away lanes are rarely selected over closer alternatives. We therefore implement a *priority planning* heuristic: When planning the first lane of a new part, we initially only consider candidates where $d_i \leq 2.5\,\mathrm{m}$. Note that we can perform this check without determining the lane length s_i. If the planner fails to find candidates with $d_i \leq 2.5\,\mathrm{m}$, we repeat the lane-selection process without this heuristic. By ignoring far-away candidates that are unlikely to be selected, we can considerably reduce the time required to plan a new part. Figure 2.19 provides an example for the parallel lane candidates considered when planning a new part.

2.4.2.3. Lane Adjustment

We have now selected the next lane $\breve{K}$ according to our cleaning strategy. Next, we adjust $\breve{K}$ before passing it to the robot's control automatons. While these adjustments can improve the cleaning performance, they are also computationally expensive. For this reason, it is not feasible to apply them to all possible lane candidates K_i. Instead, we only make these improvements to the chosen lane candidate $\breve{K}$.

As mentioned, parallel lanes $\breve{K}$ that begin a new part should be placed $\delta_l = 30\,\mathrm{cm}$ from the

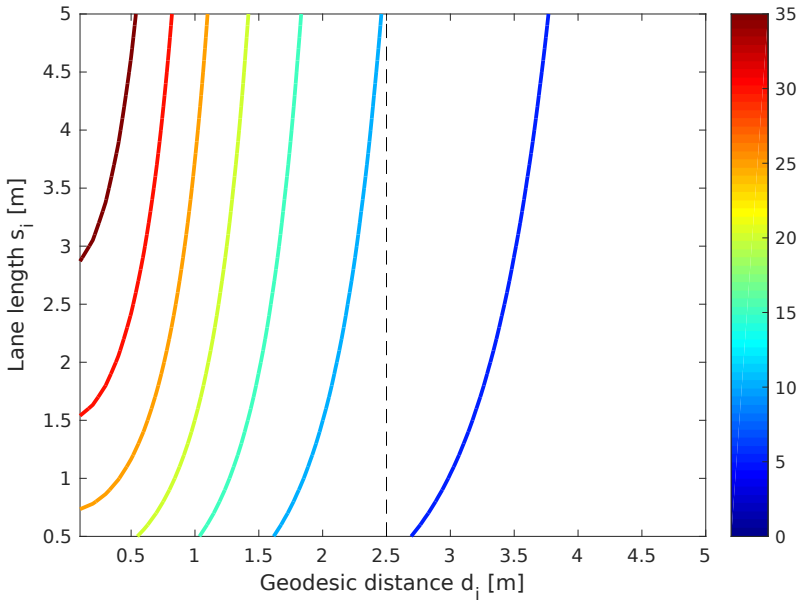


Figure 2.18.: The cost-benefit ratio $r_i = b_i/c_i$ for a lane candidate K_i. This ratio depends on the planned lane length s_i and travel distance d_i. The colored lines indicate combinations of s_i and d_i that result in the same r_i. Here, nearby lane candidates with $d_i \leq 2.5\,\mathrm{m}$ almost always score higher than those with $d_i > 2.5\,\mathrm{m}$. We indicate this threshold with a dashed, black line.

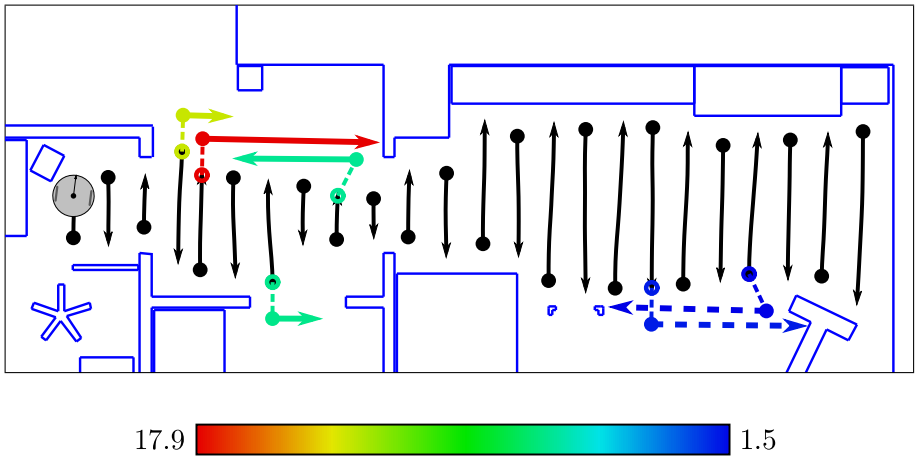
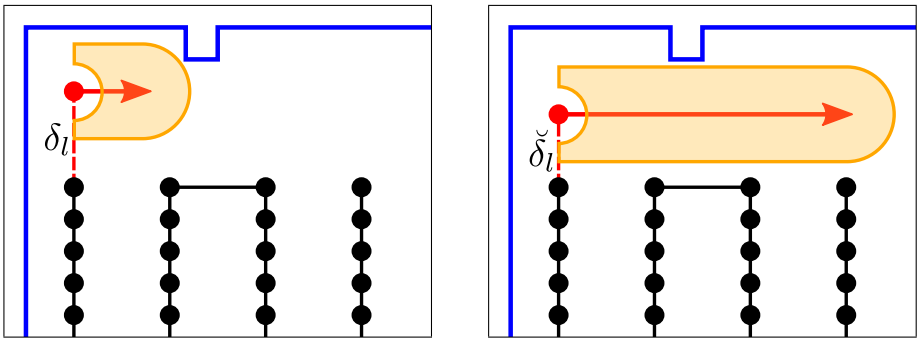


Figure 2.19.: In this example, our robot (gray shape) has just completed the first part (black arrows) within a simulated environment (blue lines). To identify the first parallel lane of the next part, the planner generates several candidates (colored arrows). For clarity, this figure does not contain the piercing lanes considered by the planner. Each candidate's starting position (colored dot) is attached to an existing map node (colored circle). The colors indicate the cost-benefit ratio r_i for a given candidate i. Here, our robot selects a long, nearby lane (red), while skipping a closer but shorter alternative (yellow). The rightmost lane candidates (blue, dashed arrows) lie beyond a distance of $2.5\,\mathrm{m}$. Consequently, they would not be considered during the initial priority-planning step.



(a) Normal lane spacing **(b)** Reduced lane spacing

Figure 2.20.: This illustration demonstrates the benefits of a variable lane spacing δ_l. In this example, we are planning the first lane in a new part (red arrow). The lane extends from an initial node (red dot) which is attached to an existing part (black dots and lines). In (**a**), we use the normal spacing δ_l (dashed line) between the new lane and parent part. However, an obstacle (blue lines) limits the area that can be covered by this lane (orange area). Using a reduced distance $\breve{\delta}_l$ leads to a much longer lane, as shown in (**b**). This way, a larger previously-uncleaned area may be covered by a single lane.

border of the existing map. Occasionally, reducing δ_l may allow such a lane to pass nearby obstacles, resulting in a greater lane length. The new lane may then cover a larger uncleaned area, even if the reduced spacing also causes some overlap. We give an example of this effect in Figure 2.20. Reducing δ_l may also allow our robot to insert additional map-graph edges between the new part and its parent. These edges can make it easier for the robot to plan paths through the map.

After selecting the first lane $\breve{K}$ for a new part, we therefore vary the lane-specific spacing $\breve{\delta}_l$. Here, we consider the lane variants $\breve{K}_\eta$, which use a reduced spacing of $\eta\delta_l$ with $\eta \in \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$. We then choose the distance factor $\breve{\eta}$ so that $\breve{K}_{\breve{\eta}}$ runs close to the largest number of nodes from the parent part's border. If the number of close nodes is maximized for several different values of $\eta \in H$, we select

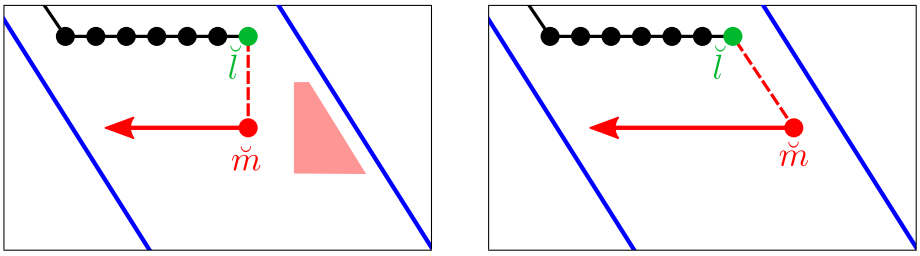
$$\breve{\eta} = \arg\max_{\eta \in H} \eta s_\eta, \tag{2.13}$$

where s_η is the length of the lane candidate $\breve{K}_\eta$. Here, ηs_η serves as an approximation for the uncleaned area covered by $\breve{K}_\eta$.

As seen in Figure 2.21a, there may be uncleaned gaps between a part and nearby obstacles. While adjusting a parallel lane $\breve{K}$, we attempt to close such gaps by moving the lane's starting node $\breve{m}$ towards the obstacle. Specifically, the planner can shift $\breve{m}$ up to $50\,\mathrm{cm}$ backwards along the planned lane. The example in Figure 2.21b shows that this can reduce the uncleaned area. Note that we do not shift $\breve{m}$ if there is no nearby obstacle. In the absence of obstacles, this leads to rectangular parts with straight borders. This makes it easy for our robot to attach parallel lanes to these straight borders later on.

2.5. Robot Control

In Section 2.4, we have discussed how to plan paths and cleaning lanes using the robot's map from Section 2.3. Next, we describe how the robot executes these plans using a number of state machines called *automatons*. Using these automatons, our robot translates the general plan into specific



(a) Without lane adjustment **(b)** After lane adjustment

Figure 2.21.: Adjusting the starting node $\breve{m}$ (red dot) of the chosen lane candidate $\breve{K}$. As seen in (**a**), there may be gaps between the starting node $\breve{m}$ and a nearby obstacle (blue lines). Subsequently, some of the floor space (red area) will remain uncleaned. As illustrated in (**b**), the planner can close such a gap by shifting $\breve{m}$ backwards. However, we only allow shifts that keep the path from the border node $\breve{l}$ to $\breve{m}$ free of obstacles.

motor commands. As long as the robot is operating, a main loop updates the active automaton and generates motor commands at a frequency of 10 Hz. In turn, the active automaton may run various sub-automatons to solve certain recurring problems. The main loop also performs various housekeeping tasks, such as position estimation and obstacle detection. These tasks are generally isolated from the automatons, which simplifies their design. In this section, we first introduce the main loop and its housekeeping task in Sections 2.5.1 to 2.5.3. We then discuss the core automatons for path following and cleaning in Section 2.5.4 and Section 2.5.5, respectively.

2.5.1. Main Loop

In our cleaning-robot prototype, the main loop is structured as follows: First, the robot checks for a collision using the contact sensor described in Section 2.3.2.3. If a new collision is detected, the robot halts all movement until it has been processed. Next, the main loop passes the commanded wheel speeds to the motor controllers. In case of a recent collision, this movement is governed by the collision-handling scheme from Section 2.5.3. Otherwise the wheel speeds are determined by the active automaton.

We now process sensor information and update the robot's internal state, beginning with the obstacle map. First, we add any new obstacle points detected by the contact sensor. If available, we also process a new obstacle-detection image according to Section 2.3.2.2. Next, the robot updates its pose estimate using the most recent wheel odometry readings. If necessary, it also corrects the estimate using the latest camera image. We discuss this probabilistic pose-estimation step in Section 2.5.2. After the obstacle map and position estimate have been updated, we now run the active automaton. This changes the automatons internal state and calculates the desired wheel speeds. The automaton may decide that a new node should be added to the topo-metric map. We process such requests at the end of the main loop, thus isolating individual automatons from the details of node creation.

2.5.2. Localization

Both the path-following (Section 2.5.4) and lane-driving (Section 2.5.5) automatons require an estimate of the robot's pose. The floor space traversed by our robot usually lies within a plane, and we can thus make a planar-motion assumption. Under this assumption, the robot only moves within a 2D ground plane, and rotates only around an axis normal to the ground. We can therefore represent the pose by the metric position $\vec{p}_r = (x_r, y_r)^T$ and the orientation angle θ_r.

To estimate this pose, we employ the probabilistic scheme presented in [104]: Here, a particle filter (introduction: Thrun, Burgard, and Fox [145, chapter 4]) models our belief regarding the pose using a cloud of particles. Each individual particle within the cloud represents one hypothesis for $(\vec{p}_r, \theta_r)$. We maintain this belief by repeatedly applying both prediction and correction steps. The prediction step incorporates the robot's motion, as measured by the wheel encoders. Utilizing a kinematic model of our robot, we calculate the new pose for each particle from the observed wheel rotations. We perform this prediction step for every iteration of the main loop. Here, we also add some noise to the pose of each particle to model uncertainties in the wheel odometry.

Over time, this noise disperses the poses within the particle cloud. We therefore correct our pose estimate by using nearby map nodes as landmarks. First, we estimate the relative pose between the current location and the location of a given map node. In our prototype, we employ the visual min-warping [100, 102, 101] relative-pose estimation method. By comparing the stored panoramic image from the node with the current camera image, this method gives an estimate for the robot's orientation and bearing relative to the node. When we created the node, we also stored the particle cloud describing the robot's pose at the node. Using this cloud, we can now correct the current pose estimate: First, we calculate the relative poses for all pairs of current and stored particles. Comparing these poses to the visual pose estimate gives us the likelihood that a given particle agrees with the panoramic images. Subsequently, we assign a greater weight to those particles that agree with the visual observations. When resampling the particle cloud, low-weight particles are likely to be omitted. As a result, we tend to eliminate those particles that disagree with the visual pose estimate.

Due to the unknown scale of the environment, our visual pose estimates do not include the distance from the node. To compensate for this, we correct the particle cloud based on multiple nodes in different directions. The specific nodes used for the correction are chosen by the active automaton. Accurate visual pose-estimation requires images which are somewhat close to each other. We must therefore base our correction step on nearby nodes within the map. Since our map lacks global metric consistency, our pose estimate is thus only accurate relative to these nearby nodes. Note that the particle cloud encodes many possible hypotheses, yet our strategy makes decisions based on a single pose estimate $(\vec{p}_r, \theta_r)$. We calculate this single pose estimate by averaging over all poses from the particle cloud. Such an averaged pose estimate is also used to calculate the positions of new nodes or obstacle points.

2.5.3. Collision Handling

Our cleaning robot always attempts to remain free of obstacles by using its obstacle map. In practice, position estimation errors and undetected obstacles may occasionally lead to collisions. Our robot detects these collisions with the contact sensor introduced in Section 2.3.2.3. After the sensor is triggered, the robot will stop and disable further movements. This prevents the robot from pushing the obstacle or spinning its wheels while processing the collision. The robot then confirms the collision by waiting for $200\,\mathrm{ms}$. If the contact sensor reports no collision during this time, the robot continues its previous task. Our collision handling routine also disregards rearward collisions detected during fast forward movements. These steps alleviate problems with vibration-induced false collisions when driving over rough ground.

In case the collision is confirmed, the robot adds the obstacle to the map. We determine the obstacle's position from the robot position and the contact direction measured by the sensor. Since this contact direction is somewhat uncertain, we represent the obstacle by several points placed within $\pm 3.75^\circ$ of the measured direction. Next, the robot uses a sub-automaton to back away from the obstacle by $1\,\mathrm{cm}$. If necessary, the automaton avoids recently-detected obstacles behind the robot by turning on the spot before reversing. After successfully moving away from the obstacle, the robot returns to its previous task. However, our robot will enter a failure state if obstacles make

reversing completely impossible.

2.5.4. Path Following

In general, the path-following automaton attempts to follow the path planned according to Section 2.4.1. Recall that a path is a sequence of nodes $P = \{i_0, i_1, ..., i_N\}$ within the topo-metric map, where each pair (i_k, i_{k+1}) for $k \in [0, N-1]$ is connected by a map edge. In general, the robot therefore follows this path by driving straight from node to node. However, the path-following automaton can make minor deviations from this planned, ideal path. Besides being more efficient, this also lets the automaton avoid unexpected obstacles without completely replanning the path. Note that such deviations are not represented in the idealized path P, which simplifies planning.

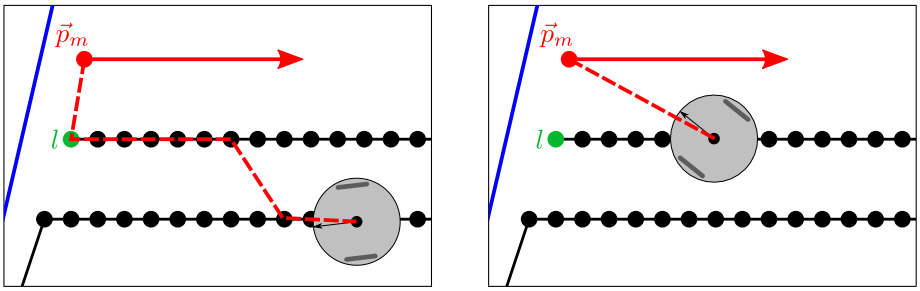
Since the nodes (i_k, i_{k+1}) are connected by an edge, the distance between their positions $|\vec{p}_{i_k}, \vec{p}_{i_{k+1}}|$ is bound to be low. While moving from i_k to i_{k+1}, the robot should therefore always be close to the current subgoal node i_{k+1}. Our topo-metric map is approximately metrically consistent at this small scale. The automaton thus generates motor commands by comparing the estimated robot pose $(\vec{p}_r, \theta_r)$ to the subgoal position $\vec{p}_{i_{k+1}}$. In a naive approach, our robot first makes a stationary turn towards $\vec{p}_{i_{k+1}}$, and then moves forward until it reaches the subgoal node i_{k+1}. After incrementing k, the robot repeats this process until it reaches the goal node i_N. While following the path, the robot continuously updates its pose estimate $(\vec{p}_r, \theta_r)$ according to Section 2.5.2. For the correction step, the automaton chooses three map nodes from the robot's vicinity.

In practice, frequent stopping and turning can be time-consuming, and may seem inelegant to a human observer. To mitigate this problem, our robot turns towards the subgoal node i_{k+1} while also moving forward. Yet unlike turning on the spot, such an approach result in a curved robot trajectory. This trajectory deviates from the planned path P, which assumes that the robot moves from node to node in straight lines. The robot may thus face obstacles that were considered unimportant while planning P. Considering these factors, we use a compromise that combines both types of turning: If the robot has to turn by a large angle $\Delta_\theta > 40°$, it will turn on the spot. Once $\Delta_\theta \leq 40°$, the robot will instead turn while driving forward. This keeps the trajectory reasonably close to the planned path P, while limiting the time spent on stopping and turning. However, if the robot is in close proximity to obstacles, it should adhere to P more strictly. We therefore use a threshold of $\Delta_\theta \leq 5°$ when in close proximity to recently-detected obstacles. This check for nearby obstacles uses the high-speed heuristic introduced in Section 2.3.2.1. Unfortunately, unexpected obstacles may still prevent a direct approach to the next subgoal i_{k+1}. After approaching such an obstacle to within $10\,\mathrm{cm}$, our robot thus switches to the obstacle avoidance scheme from Section 2.5.4.2.

2.5.4.1. Off-Map Path Following

Beyond the final subgoal i_N, a path may continue to a location that does not correspond to any node i_k. This usually occurs when traveling to the starting position $\vec{p}_m$ of a new lane. Here, the automaton generates motor commands based on $\vec{p}_m$ and the estimated robot pose $(\vec{p}_r, \theta_r)$. It is important that we precisely place the robot at the position $\vec{p}_m$. If the robot is placed incorrectly, this error will affect the entire lane extending from $\vec{p}_m$. The automaton therefore reduces the robot's speed and prefers precise, stationary turns once within $5\,\mathrm{cm}$ of $\vec{p}_m$.

When traveling to the starting position $\vec{p}_m$ of a new line, the robot follows the plan shown in Figure 2.22a: First, the robot travels along the path P to the border node l, from which it directly approaches $\vec{p}_m$. To reduce travel times, the robot may however skip the remainder of P and approach $\vec{p}_m$ immediately (Figure 2.22b). This shortcut is taken if fewer than $N - k \leq 8$ subgoals remain on the path, and if the distance is $\|\vec{p}_m - \vec{p}_r\| \leq 80\,\mathrm{cm}$. Furthermore, the proposed shortcut from $\vec{p}_r$ to $\vec{p}_m$ must be free of known obstacles. If the robot detects such obstacles, it will instead continue the planned path P to the border node l.



(a) Regular approach **(b)** Shortcut approach

Figure 2.22.: Our robot reaches the start of a new lane by following a planned path P through the existing map (black dots). In (**a**), P is shown as a dashed line, which leads the robot to the planned border node l (green dot). From there, the robot can approach the starting position $\vec{p}_m$ (red dot) and drive the planned lane (red arrow). To reduce the travel time and distance, our robot may take a shortcut once it is close to $\vec{p}_m$. This case is shown in (**b**), where the robot skips the remainder of the path P.

2.5.4.2. Obstacle Avoidance

Within our topo-metric map, edges indicate that direct travel between two nodes is possible; consequently a planned path P should be free of obstacles. In practice, our robot may still encounter obstacles while approaching the next subgoal i_{k+1}. Note that this problem has at least two distinct causes: In some cases, the robot incorrectly detects a non-existing obstacle. Alternatively, a new or previously undetected obstacle may block the planned trajectory. The path-following automaton can use two different schemes to deal with this problem.

To correct false-positive obstacle points, we first attempt to rebuild the nearby obstacle map. We begin by removing all nearby obstacle points $\vec{o} \in O$, as selected in Section 2.3.2.1. After deleting these obstacle points, a sub-automaton steers the robot through a slow, stationary turn of $\pm 90°$. This gives the sensors time to repopulate the obstacle map with new readings. The robot then makes another attempt at following the planned path.

However, we may still detect obstacles that block a straight approach towards the subgoal i_{k+1}. In this case, we employ another sub-automaton to reach i_{k+1} using a hybrid technique: This automaton still follows the path from i_k to i_{k+1}, but is not restricted to a straight approach. Instead, it employs a vector field histogram (VFH) to avoid obstacles and reach i_{k+1} through an indirect trajectory. Specifically, we use a modified version of the VFH+ method presented by Ulrich and Borenstein [150]. Figure 2.23 illustrates the resulting obstacle avoidance procedure.

To deal with changing environments, the obstacle-avoidance sub-automaton only considers obstacles detected within the last 12 s. Additionally, we limit the histogram to local obstacle points within a radius of

$$d_v = \max(\min(\|\vec{p}_r - \vec{p}_{i_{k+1}}\|, 100\,\text{cm}), 5\,\text{cm}) \tag{2.14}$$

from the estimated robot position $\vec{p}_r$. We therefore do not consider obstacles that lie far beyond the current path-following subgoal i_{k+1}.

Next, we add the selected obstacle points to a robot-centered angular histogram with a resolution of $5°$. To account for the size of the robot, we surround each obstacle point with a disc of radius $r_v = r_r - 2\,\text{cm} = 15.3\,\text{cm}$. Due to uncertainties in the estimated obstacle positions, this r_v is lower than the robot radius r_r. Each obstacle point is now added to all bins that overlap with this disc-shaped area.

Next, we calculate a summed weight from the obstacle points within each histogram bin. For the

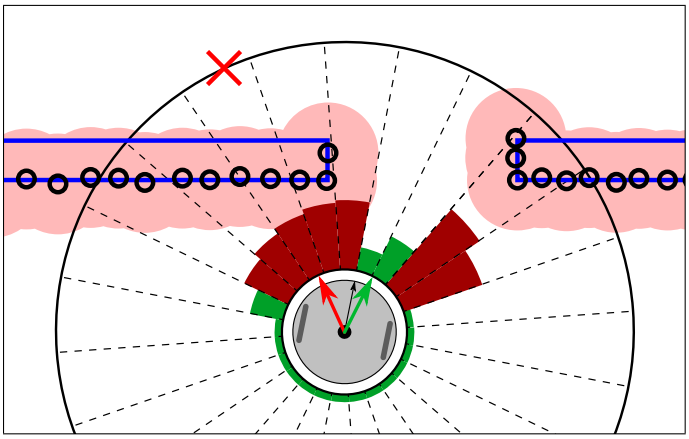


Figure 2.23.: In this illustration, the robot is trying to reach the goal position marked with a red cross. However, a straight approach (red arrow) is blocked by a wall (blue lines) with a narrow opening. Within the robot's map, this wall is represented by obstacle points (black circles). To construct a vector field histogram, we first expand each obstacle point to a circle with a radius of $r_v = r_r - 2\,\mathrm{cm}$ (light red area). Next, we add these expanded points to an angular histogram, here illustrated using dashed lines. We only consider local points within a radius d_v (outer circle), as specified in Equation (2.14). Each bin a is shown partially filled in red or green, where the filled proportion represents the weight ω_a of that bin. For sufficiently high ω_a, the bin is considered blocked and shown in red; open bins are shown in green. From this binary histogram, the robot selects an open movement direction (green arrow) towards the goal.

bin with index a, this weight is

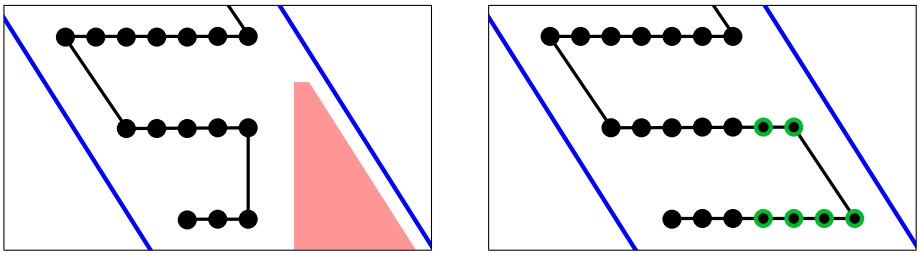
$$\omega_a = \sum_t w_{a,t}^2 \max\left(1 - \frac{d_{a,t} - r_v}{d_v}, 1\right), \tag{2.15}$$

where $w_{a,t}$ is the weight of the tth obstacle point within the bin a. Similarly, $d_{a,t} = \|\vec{o}_{a,t} - \vec{p}_r\|$ is the distance between the robot and the obstacle position $\vec{o}_{a,t}$. Thus, each obstacle point contributes the square of its weight, scaled linearly with its distance.

After constructing the histogram from our obstacle map, we follow the regular VFH+ scheme: First, we create a binary histogram by applying a threshold to the bin weights ω_a. In our case, we consider bins with a weight of $\omega_a \geq 2.8$ to be blocked by obstacles. The robot may therefore not move in the direction associated with a blocked bin a. Such a blocked bin a will only be considered open once ω_a falls below 2.0. This hysteresis reduces the effect of minor variations in ω_a. VFH+ then generates movement candidates φ_i from the valleys formed by adjacent open bins. For narrow valleys (< 5 bins), only the valley's center is a valid direction of movement. Finally, VFH+ selects the best movement direction $\breve{\varphi}$ based on the cost function introduced in [150]. This cost function takes into account the subgoal position $\vec{p}_{i_{k+1}}$, the current robot orientation θ_r, and the previously selected movement direction. To avoid collisions, we reduce the robot's speed whenever it is not driving directly towards the subgoal position $\vec{p}_{i_{k+1}}$.

Note that VFH is a purely reactive scheme, and may move the robot far away from the planned path. While some deviations from the path are necessary to avoid obstacles, the robot should not get lost. We therefore stop the VFH sub-automaton if it fails to reach the subgoal i_{k+1} after 30 s. Similarly, we stop the automaton if the robot moves more than 30 cm from the map edge (i_k, i_{k+1}). In both cases, the robot will attempt to return to the node i_k and replan its path from there.

If both of these obstacle-avoidance schemes fail, the robot must take a different path. We therefore block the impassable map-graph edge (i_k, i_{k+1}) by greatly increasing its travel cost. The



(a) Without lane extension **(b)** With lane extension

Figure 2.24.: In this illustration, the robot cleans a diagonal corridor (blue lines) from the top to the bottom. As usual, the resulting map nodes are shown as black dots, connected by black lines. In the naive approach shown in (**a**), lanes do not extend beyond their immediate predecessors. This leaves a gap in the coverage (red area), which the robot would have to cover separately. We therefore allow the automaton to ignore this constraint to reach an obstacle up to $50\,\mathrm{cm}$ ahead. In (**b**), this is the case for the middle lane, which now ends close to a wall. This proximity to the wall also lets the planner apply the adjustment from Figure 2.21 to the subsequent lane. As shown by the additional nodes with green borders, the covered area is thus increased.

robot then replans the path to the current goal node i_N using the regular planner from Section 2.4.1. If this fails to find a traversable path, the goal node i_N is marked as unusable. Such unusable nodes are no longer considered as border nodes when planning new lanes. In case uncleaned space remains, the robot simply selects a new lane according to Section 2.4.2. However, if the robot cannot return to its home location after cleaning is complete, it stops and enters a failure state.

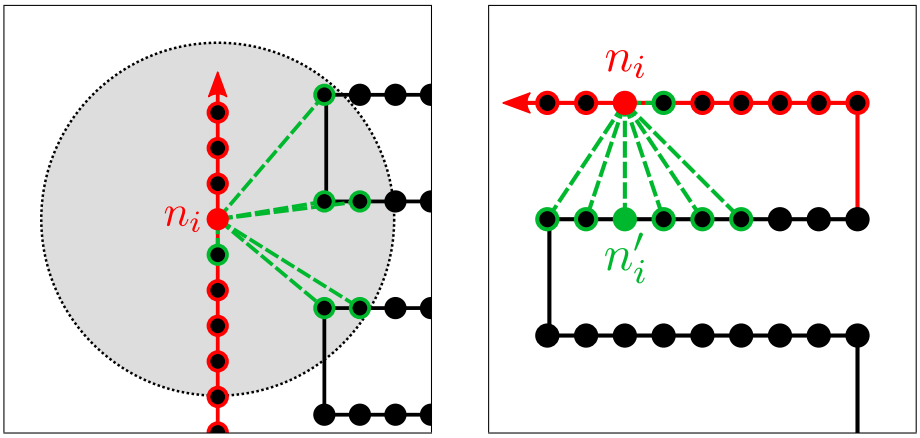
2.5.5. Driving Lanes

We use an automaton to extend the cleaned area by driving the new lanes as planned. This automaton assumes control after the robot has reached the new lane's starting position $\vec{p}_m$ (Section 2.5.4). The behavior of the lane-driving automaton differs between parallel (regular) lanes, piercing lanes, and the very first lane in a cleaning run.

2.5.5.1. Parallel Lanes

First, we will discuss the most common case of driving a parallel lane. The automaton follows a line with a cleaning direction θ and length s, which extends from the starting position $\vec{p}_m$. A PI controller steers the robot towards a point on the line that is $d_l = 60\,\mathrm{cm}$ ahead of the current position $\vec{p}_r$. For this d_l, the robot quickly corrects displacements from the planned lane without making sharp turns. Motor commands are thus based purely on the estimated metric pose of the robot and the planned lane. To estimate its pose while driving the lane, our robot applies the localization method from Section 2.5.2. Recall that this method requires reference images from nearby nodes to correct the pose estimate. Here, the automaton selects nodes along the map border to which the new lane is attached.

While driving the lane, the automaton extends the map by requesting a new node every $10\,\mathrm{cm}$. This continues until the new lane reaches the planned length s. An additional constraint arises if the new lane extends the current part: Here, the lane ends once it passes the first node of the previous, antiparallel lane, even if the length remains below s. In the absence of obstacles, all lanes in a part will thus have the same length. As seen in Figure 2.2, this leads to parts with straight sides. This is a useful property, as it is easy to attach future parts to such straight borders.



(a) Edges between parts **(b)** Edges between lanes

Figure 2.25.: After completing a new lane (red line), we connect its nodes (black dots, red outline) to the existing map (black line, black dots). For each given node n_i within the new lane (red dot), we first add an edge to the previous node along the same lane. In this illustration, new edges are shown as green, dashed lines, while the nodes newly connected to n_i are shown with a green outline. In (**a**), the new lane is the first of a new part. We therefore connect n_i to all nodes from the parent part that lie within a radius of $50\,\mathrm{cm}$ (gray circle). In (**b**), we consider a new lane that is added to an existing part. Here, we first add an edge between n_i and its nearest neighbor n'_i along the previous lane (green dot). We then connect n_i with up to three predecessors and successors of n'_i along that previous lane.

In practice, obstacles may also affect the actual lane length: While driving a new lane, the automaton continuously checks for obstacles in front of the robot. If such an obstacle is less than $10\,\mathrm{cm}$ away, the robot will carefully approach it at a reduced speed. Once the obstacle distance reaches $1\,\mathrm{cm}$, the automaton ends the current lane. Yet if this has not occurred after $5\,\mathrm{s}$, the robot must have passed the obstacle; the automaton therefore continues the lane at a normal speed.

Our robot may also react to obstacles by lengthening the current lane: As discussed, a lane extending an existing part should not pass the end of its immediate predecessor. However, this may leave small uncleaned gaps between the lane's end and a nearby obstacle. To prevent this, the robot continues the lane if it detects an obstacle less than $50\,\mathrm{cm}$ ahead. In this case, the lane ends after it reaches either the obstacle or its planned length s.

Since our map lacks global metric consistency, the robot may occasionally encounter an unexpected previously-cleaned area. After creating a new node, we therefore compare the current camera image to those from the border nodes of the existing parts. This comparison is based on an image-distance heuristic contributed by Horst and Möller [69]. If the automaton detects that the robot has reached such a previous part, it ends the current lane.

Upon reaching the end of the current lane, the automaton requests the creation of a final node. Having thus completed the lane, we now have to connect its new nodes n_i through edges. Here, we follow the procedure shown in Figure 2.25. First, we add an edge between each new node n_i and its immediate successor n_{i+1}. Next, we also connect these nodes to the rest of the map graph in one of two ways: For the first lane in a new part (Figure 2.25a), we add edges between the new nodes and nodes from the parent part. Specifically, we connect nodes with an estimated Euclidean distance $\leq 50\,\mathrm{cm}$. If the new lane extends an existing part (Figure 2.25b), we connect it to the previous, parallel lane. In general, we insert an edge between the new node n_i and its closest neighbor n'_i from the previous lane. Additionally, we connect n_i with up to three nodes that follow and precede

n_i', for up to seven total edges. In this case, we also apply several checks to eliminate edges which the robot may be unable to traverse.

2.5.5.2. Piercing Lanes

In general, the control mechanism for piercing lanes is similar to that used for parallel lanes: As before, our robot follows a planned straight line which extends from a starting position. However, piercing lanes require some modifications: First, the robot cannot rely on nodes from other parts to correct its position estimate. Recall that we only use piercing lanes to pass through narrow obstacles. Since the obstacles are close, moving the robot may cause large changes within the camera image. This increases the error of our visual pose-estimation method, leading to greater pose uncertainty. We therefore correct the estimated pose using nearby nodes within the piercing lane. Relying on such few nearby nodes within the current lane still leads to an increased pose uncertainty. However, recall that the planner restricts piercing lanes to a length of $s \leq 140\,\mathrm{cm}$. This limits the errors in the robot's pose estimate, allowing the control automaton to function normally.

While driving a piercing lane, the robot explores a new area away from the existing map. Yet at the same time, the limited lane length leaves little time to detect new obstacles. Thus, the robot may have little information about free space around the piercing lane. To compensate for this, our robot will rotate on the spot after completing a piercing lane. This turn lets us gather the obstacle data needed to plan subsequent lanes.

2.5.5.3. First Lane

The very first lane for a new cleaning run poses a special challenge, since there is no existing map to which it could be attached. A straight and well-formed first lane is important, since it serves as the foundation on which our robot builds the subsequent map. In practice, an approach similar to the piercing lanes gave good results: From its start-up position, the robot simply drives a straight line. As in Section 2.5.5.2, we use nearby nodes within the lane to correct the pose estimate. We also limit the length of the first lane to $190\,\mathrm{cm}$. We consider this to be a good compromise between the straightness and length of the lane. Since we assume that the robot starts within a fairly open area, we skip the obstacle-detection turn .

2.6. Experiments and Results

To test our cleaning robot, we order it to clean an unknown environment. Experiments on our physical robot are important to judge our framework's real-world performance. However, testing in a simulated environment also offers some advantages: First, we can easily determine the true state of both the environment and the robot within it. This makes it much easier to analyze our robot's behavior. Second, we can experiment with a greater variety of environments, including some not accessible for real-robot experiments. Finally, the simulation runs faster than our real robot, allowing us to perform a large number of experiments within a limited time. Consequently, this section includes results from both the real robot and our simulation.

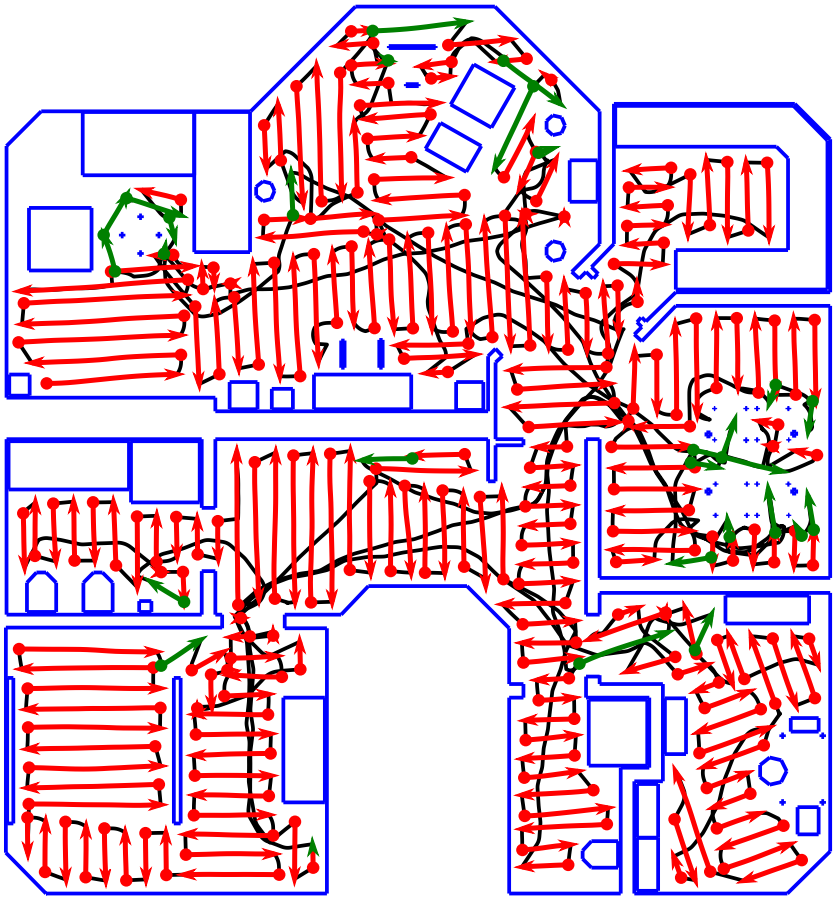
Figure 2.26 shows cleaning runs from two different real-world apartments. These apartments contain different room types, including a bedroom, living room, hallway, kitchen, and bathroom. In each case, our robot manages to cover most of the free space with meandering lanes. However, the robot also uses piercing lanes to clean some of the narrow areas. Afterwards, the robot successfully returned to its starting location.

Figure 2.27 shows similar results for two simulated environments. As in the real-world experiments, the robot also succeeds in cleaning these larger apartments. For these simulated experiments, we made a small change to the self-localization component from Section 2.5.2: Since simulating

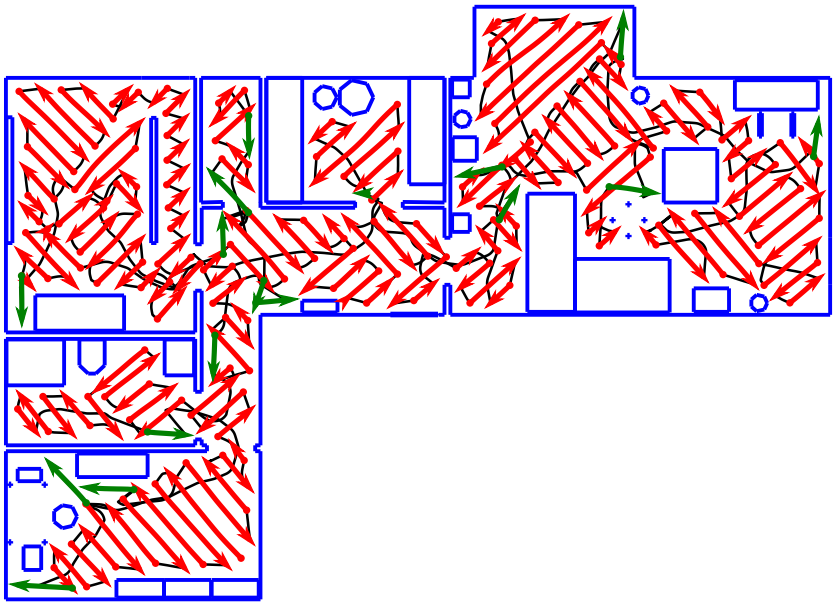
(a) Apartment A (real)

(b) Apartment B (real)

Figure 2.26.: Cleaning runs from real-world apartments, as performed by our robot prototype. We represent lanes as thick arrows, with a dot indicating the starting position of each lane. Regular lanes are colored red, while piercing lanes are shown in green. A thinner, black line represents the trajectory of the robot between the lanes. Small black circles depict the obstacle points detected by the robot. Lacking a ground truth, we base the positions of these elements on the robot's internal estimates. Because the estimates lack global metric consistency, the obstacle map and lanes in these illustrations appear somewhat distorted.

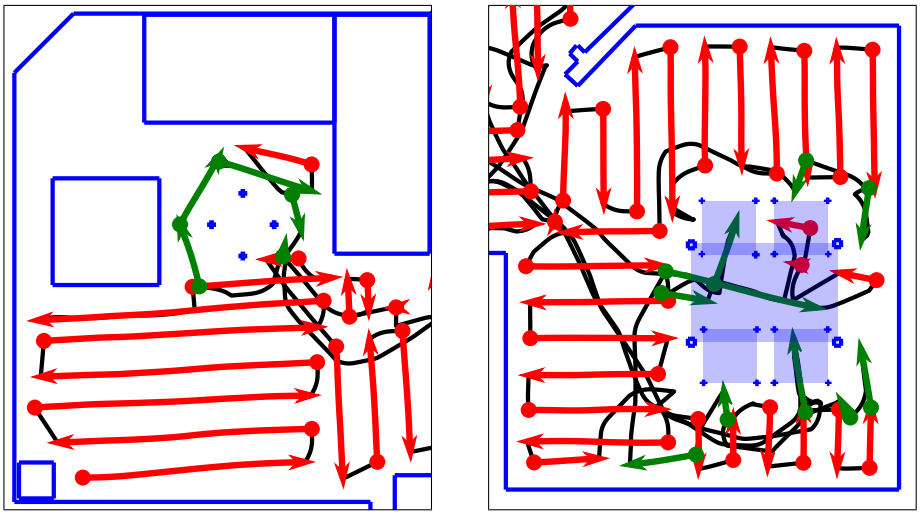


(a) Apartment C (simulated)



(b) Apartment D (simulated)

Figure 2.27.: Simulated cleaning runs from two comparatively large apartments. We visualize the robot's behavior using the same elements as in Figure 2.26. However, we omit the obstacle points, and instead show the actual simulated obstacles as blue lines. Since they are based on a simulation, these illustrations show the robot's true trajectory within the environment.



(a) Cleaning around a small table **(b)** Cleaning under a table and chairs

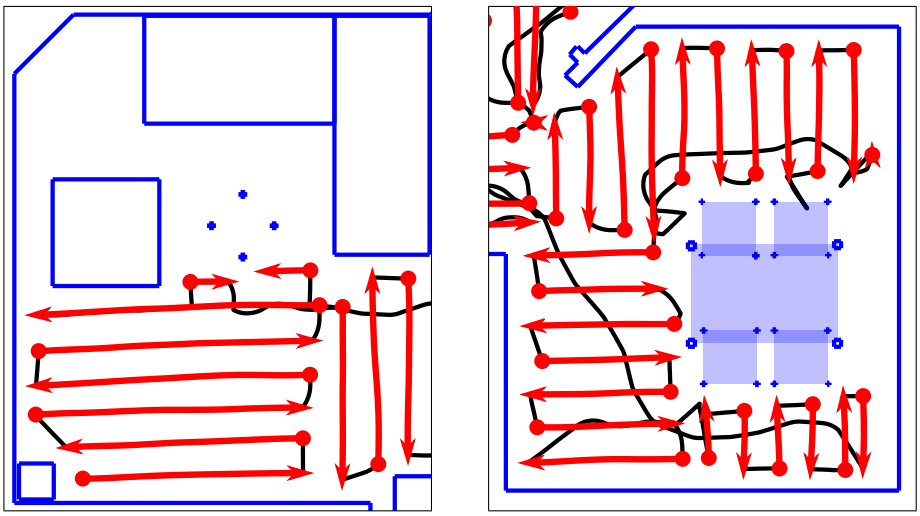
Figure 2.28.: We found that narrow spaces may be difficult to cover with meandering lanes. These figures show two details from the cleaning run in Figure 2.27a: In (**a**), the robot uses a chain of piercing lanes (green arrows) to clean around a small table in the apartment's top-left corner. The detail (**b**) was taken from the center-right room. Here, the robot attempts to clean the narrow space between the legs of a table with four chairs. We have highlighted the area underneath this furniture using blue, transparent rectangles. The black line represents the robot's trajectory while not driving a lane. Note how our robot repeatedly moves around the obstacles to travel between the piercing lanes.

realistic camera images is both difficult and time-consuming, we do not use the visual min-warping method for pose estimation. Instead, we provide the robot with relative-pose estimates by adding noise to the true relative pose between two locations. Note that the rest of the localization scheme remains unchanged, and works as described in Section 2.5.2.

We also highlight some of the difficulties encountered by our cleaning robot, which we will discuss in Section 2.7: Figure 2.28 shows two narrow spaces from Figure 2.27a. These areas are poor fits for the meandering lanes employed by our cleaning robot. Consequently, the robot performs numerous piercing lanes (green arrows) while repeatedly driving around the area (black lines). While this contravenes our meandering-lane strategy, Figure 2.29 shows that our robot cannot cover such narrow spaces with parallel lanes.

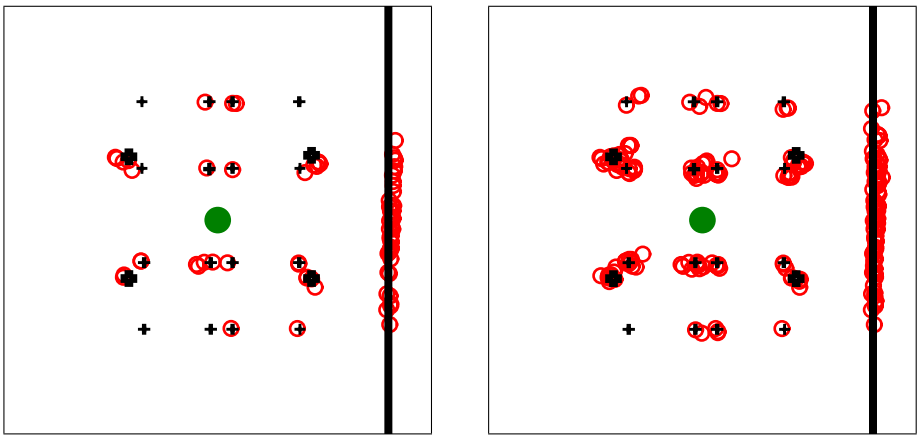
In Section 2.3.2, we discussed how our robot retrieves local obstacle data from our obstacle map. Given a map node k, we constructed a set of local obstacle points O'_k by combining data from multiple range-sensor readings. To compensate for sparse obstacle data, this set also contains points recorded at nearby nodes k'. However, even within this local neighborhood of k our map does not provide perfect metric consistency. For performance reasons, we also do not process the obstacle points in O'_k to ensure their consistency. As shown in Figure 2.30, the combined set O'_k may thus be somewhat inconsistent.

To estimate the efficiency of our robot, we compare the number of map nodes to the distance traveled: While cleaning a new area, our robot inserts one node every $\approx 10\,\mathrm{cm}$. However, we add no nodes when traveling within the previously-mapped area. Thus, creating more nodes within a given travel distance means that more of this distance was spent on cleaning new areas. Figure 2.31 shows such an estimate for the cleaning runs from Figure 2.27. Note that the node-distance curves flatten over time, indicating a decrease in efficiency as the cleaning run progresses.



(a) Small table, no piercing lanes **(b)** Table and chairs, no piercing lanes

Figure 2.29.: This figure shows an attempt to clean the areas from Figure 2.28 using only meandering lanes. Without piercing lanes, our robot failed to clean behind the small table in (**a**) or underneath the table and chairs in (**b**). However, the black trajectory shows that the robot also spent comparatively less time traveling around the obstacle.



(a) Single-node obstacles O_k **(b)** Combined obstacles O'_k

Figure 2.30.: This figure shows the local obstacle points (red circles) for the node k (green disc) in a simulator experiment. Black lines denote the true obstacles in the environment; for clarity, these are drawn above the obstacle points. In (**a**), we only show the point set O_k (Section 2.3.2). These points were recorded within 7 s before or after the robot created or visited the node k. Here, some obstacles are not detected, while others are represented by isolated obstacle points; as per Figure 2.7 these may be eliminated during the DBSCAN filtering step. The point set O'_k in (**b**) also contains obstacle points from other nearby nodes k'. While this increases the number of obstacle points, these points are also less consistent and appear somewhat spread out.

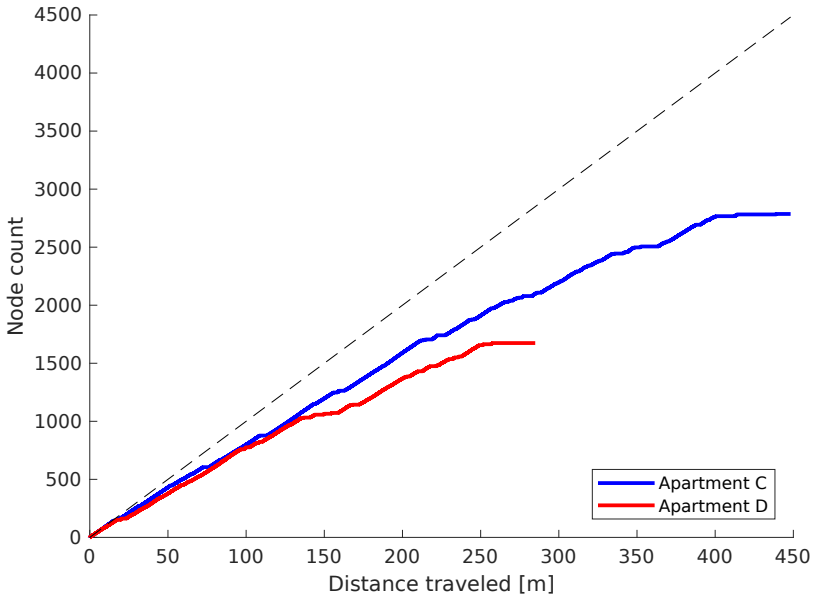


Figure 2.31.: The number of nodes in the map compared to the distance traveled by the robot, based on the experiments from Figure 2.27. A black, dashed line represents the upper bound of one node per $10\,\mathrm{cm}$. Since our robot also travels through the previously-cleaned area without creating new nodes, the actual results remain below this limit.

2.7. Discussion

In general, we found that our cleaning robot can solve its task. It has proven capable of covering a variety of indoor environments, including apartments, offices, and laboratories. As an example, Figure 2.26 and Figure 2.27 show successful cleaning runs in real and simulated environments, respectively. Next, we focus our discussion on specific aspects and problems of our robot's fundamental design.

One problem arises because the robot cleans the floor by driving parts of meandering lanes. As discussed in Section 2.4.2, this leaves no uncleaned gaps between adjacent lanes. The resulting parts also match the rectangular room shapes commonly found in domestic environments. In practice however, rooms usually contain many obstacles of varying shape and size, such as furniture. Thus, the actual shape of the free floor space is often highly irregular. We found that our fairly rigid part-lane structure is not an ideal fit for such areas. This problem is more severe within narrow spaces, for example underneath tables and chairs. Figure 2.28 shows two such cases, which our robot encountered within a simulated apartment. Here, our systematic cleaning strategy is reduced to driving many short piercing lanes. Unfortunately, the straight piercing lanes are sometimes difficult to fit into narrow, complex spaces. The resulting uncleaned gaps lower our robot's effectiveness. In Figure 2.28b, the robot covers a considerable distance while traveling between the short piercing lanes. This is somewhat inefficient, since we spend both time and energy on traversing a previously-cleaned area. However, our robot cannot currently cover these narrow spaces using meandering lanes, as demonstrated in Figure 2.29. In the future, it might be advantageous to clean such areas with curved lanes. Such curved lanes would however increase the complexity of the map and planning components.

In Section 2.4.2, we explained that our robot only plans ahead by a single lane. This allows the robot to operate with a limited map, while also saving computational resources. Without planning

far ahead, the order of the individual parts is sometimes not optimal. For example, the robot may leave small niches uncleaned, instead choosing to clean a large area first. Our robot must then return to this niche later, which increases the overall travel distance. The node-distance graphs from Figure 2.31 demonstrate this problem: Early on, our robot creates new map nodes at a steady rate while covering previously-uncleaned areas. However, the node-creation rate eventually declines as the robot runs out of easily-cleaned space. Towards the end of the cleaning run, the robot travels long distances to add just a few additional nodes before returning to its starting location. As discussed in Chapter 5, it may be more efficient to clean the environment room-by-room. Alternatively, the robot could plan several lanes and parts in advance, with a goal of minimizing the travel distance. However, this likely requires a map that extends far beyond the previously-cleaned area. These changes also require more computational resources, extending the time and power spent on planning.

Our rather sparse obstacle map and relaxed metric consistency also affect our robot's efficiency. Figure 2.30 illustrates this problem by comparing two sets of obstacle points: Figure 2.30a shows the obstacle points recorded shortly before or after the robot created or visited a map node k. While these points are largely consistent with the actual obstacles, they are also rather sparse. As described in Section 2.3.2, we thus also include obstacle points from other nearby map nodes k'. In Figure 2.30b, this increases the density of the obstacle points, but also makes them less precise. Due to these issues, the robot may create a plan which is valid according to the map. While executing the plan, the robot then however encounters an unexpected obstacle. Alternatively, the robot may also falsely detect a nonexistent obstacle. As discussed in Section 2.5, the control automatons try to deal with such problems by modifying the plan. Yet in some cases, our robot is forced to abandon its current action. It must then create a new plan, which takes time and reduces the robot's efficiency. Note that this kind of problem cannot be avoided entirely when dealing with uncertain maps. Nevertheless, a higher-resolution obstacle sensor or a map-correction scheme may provide better obstacle data. We hope that this would reduce the impact that map uncertainties have on the robot's behavior.

A comprehensive, quantitative analysis of our robot's cleaning performance would also be useful. In practice, even minor variations in the robot's starting pose or environment can considerably influence a cleaning run. A meaningful study thus requires repeated cleaning runs, as well as experiments in a number of different environments. To analyze our robot's cleaning performance, we need to track its movement through the environment. While our group has previously developed a suitable tracking system using overhead video cameras, installing and calibrating several of these systems across multiple rooms is a time-consuming procedure. Ideally, the experiments would be conducted in realistic environments, such as inhabited apartments. However, this may also irritate the regular human occupants of these spaces. This makes it harder to find a sufficient number of suitable domestic environments. Consequently, a comprehensive, quantitative analysis of our robot's performance remains an unresolved task.

Overall, we believe that our robot performs its cleaning task reasonably well. The remaining issues seem comparatively minor, and may be unavoidable trade-offs in the face of limited resources. More importantly, our prototype provides a practical example for a domestic floor-cleaning robot. It thus offers a valuable experimental platform for the research questions discussed in the subsequent chapters.

3. Comparing Holistic and Feature-Based Methods for Visual Relative-Pose Estimation

The localization scheme used by our domestic cleaning robot estimates relative poses from pairs of camera images. Feature-based and holistic methods present two fundamentally different approaches to this pose-estimation problem. While our prototype currently uses the holistic min-warping method [102], the literature also contains numerous other solutions. In this chapter, we therefore evaluate a selection of such methods within the context of our cleaning robot. We find that min-warping gives good and fast results, while being fairly robust to strong illumination changes. Some of the feature-based candidates can provide excellent and robust results, but at much slower speeds. Other such methods also achieve high speeds, but at reduced robustness to illumination changes. We also provide novel image databases and auxiliary data for public use. Please note that this chapter is based on an earlier publication by the author [49].

3.1. Introduction

Visual relative-pose estimation is the problem of estimating the relative orientation and movement[1] between two camera postures from the two corresponding camera images. This has at least four applications in mobile robotics: First, pose estimation can be used to solve visual homing problems. These include returning to the point at which an image was taken, and traversing a route formed by a series of images [52, 153, 60, 12, 82, 100]. Second, when image sequences are available, successive pose estimates can be integrated to perform visual odometry, for example [26, 117]; overview: [131, 53]. However, this integration requires information about the (relative) movement distance between the individual camera postures. Third, pose estimation can act as a visual odometry component in visual localization and mapping (SLAM) systems [143, 79]. Visual relative-pose estimation can also serve as one component within a more complex SLAM system, where, for example, it is used as an input for structure-from-motion methods [20]. As described in Section 2.5.2, our cleaning-robot framework uses visual relative-pose estimation to determine its pose relative to the map. Additionally, our research group has previously presented similar navigation systems for cleaning robots [57, 59]. Fourth, at least one work uses relative-pose estimates only indirectly to compute an image similarity measure [16].

There are two fundamentally different approaches to relative pose estimation from two camera images: Feature-based methods, which we discuss in Section 3.1.1, and holistic methods, which we cover in Section 3.1.2. Evaluating these two approaches within the context of our domestic cleaning robot is the main goal of this chapter.

3.1.1. Feature-Based Methods

Methods based on local visual features are clearly the most popular solution for the visual pose-estimation problem. They serve as components in most of the works referenced above.

[1] Without depth information, it is impossible to determine movement distances from just two images. However, the relative bearing can still be derived.

As the name implies, these methods rely on distinct local image elements, such as corners, blobs, or edges. In a first step, a detector locates these keypoints for each image. The SIFT [89] and SURF [7] detectors, or variants thereof [4, 76, 106], represent one common choice. Detectors based on segment tests have also become popular in recent years, in part because they can be computed very quickly [127, 128, 93, 129, 84]. For a comparison of several detectors in the context of robot navigation, refer to the assessment by Schmidt et al. [134].

After the keypoint locations are known, a feature descriptor vector is extracted from the vicinity of each keypoint. These descriptors are often designed for a specific type of detector. As such, the SIFT [89] and SURF [7] descriptors are commonly used with their corresponding detectors. Binary feature descriptors, offering high computational efficiency, are also widely used [129, 2, 3]; comparison: [11, 43]. For an overview over numerous local-feature detectors and extractors, see [56]. Local features often rely on information contained in high spatial frequencies, and thus typically benefit from high-resolution images.

Once features have been extracted, a matcher identifies corresponding features between the two images based on their descriptors. A straightforward solution is matching through a brute-force, exhaustive search. Alternatively, approximative matchers [108, 109] or visual bag-of-words approaches [20] can be used. When working with video sequences, features can also be tracked across multiple frames [116].

Using the intrinsic and extrinsic camera calibration, we can compute a robot-relative bearing vector for each keypoint. The relative pose is then derived from the bearings of matching features in the two images. In general, at least five feature pairs [138, 117] are required, although two pairs suffice if planar robot motion is assumed [17]. If the movement of the camera is further constrained to that of a nonholonomic wheeled vehicle, the pose may be estimated from just a single pair [130]. Where the 3D position of features is known, it can also be used in the pose estimation [63]. The necessary depth information can be acquired through stereo cameras [116, 117], structure-from-motion on image sequences [143], or RGB-D cameras [71].

Incorrect feature matches greatly reduce the quality of the pose estimate, and must thus be rejected. This is commonly accomplished through random sample consensus (RANSAC) [44], a hypothesize-and-test scheme with numerous variants [148, 24, 115, 124]. An alternative approach uses Bayesian methods to determine the likelihood function for all possible planar relative poses, without explicitly rejecting incorrect matches [15]. The relative pose can then be estimated by searching for the maximum likelihood. For a wider view on feature-based relative-pose estimation, we direct the reader towards a tutorial in feature-based visual odometry [131, 53], as well as a general survey of visual SLAM [55].

3.1.2. Holistic Methods

Holistic methods follow a different approach: Instead of focusing on specific points of interest, they use the entire image to solve the pose-estimation problem. These methods typically operate on panoramic, lower-resolution images with only a moderate amount of preprocessing. Holistic methods also use lower spatial frequencies and make few assumptions about the environment. They specifically do not require the presence of certain kinds of local visual features. Originally developed as models of insect visual navigation, these methods have recently found applications in robotics. Examples include visual homing [52, 12], route following [82], and localization and mapping [51, 72, 50]. As we have explained in Section 2.5.2, our cleaning-robot framework also uses a holistic method for relative-pose estimation [103, 105, 102]. For a wider survey of these methods, see [33].

Holistic methods based on pixel-wise image comparisons (overview: [102, 101]) execute either actual or simulated test movements to explore the resulting changes in the panoramic images. In methods based on a spatial descent in translational image distance functions [158, 141] and in multi-

snapshot methods [61, 112], the robot has to execute *actual*, physical test movements. The bearing towards the capture position of image 1 can then be estimated from the image distance between image 1 and the images captured during the test movements in the vicinity of the capture position of image 2. In contrast, other methods only *simulate* test movements by distorting panoramic images accordingly. Some methods calculate short test steps to simulate the physical test movements used for spatial descent in image distance functions [12, 103, 82, 105]. Alternatively, warping methods use simulated long-distance movements to directly determine the relative-pose hypothesis [52, 140, 50, 100, 102, 97]. These simulated methods have the advantage that only a single input image from each location is required. Optical-flow methods are a second group of holistic methods, which have been studied in [151]. These methods estimate the optical flow by matching blocks of pixels or through differential techniques. They require an external orientation estimate, although this can be determined from the input images using a separate visual compass [158].

Like the feature-based candidates, the holistic min-warping method used in these experiments also estimates the pose from the displacement of image patches. However, as a holistic method it selects these patches according to a fixed scheme, without searching for distinct image elements like corners or edges.[2] Instead, min-warping operates on *all* pixel columns in a panoramic image, without deciding on whether a specific column is of interest or not. Similarly, an optical-flow method based on block matching may use blocks of pixels which are arranged in a fixed grid [151].

3.1.3. Our Contributions

As we have mentioned above, we base our comparisons on our cleaning-robot framework from Chapter 2. Consequently, we employed our robot's onboard camera to acquire the image databases for this study (Section 3.2.1). We use these panoramic views as the input for both the holistic method and the feature methods. To make comparisons between different methods possible, we assume that pose estimation must be performed using only a single, arbitrary pair of monocular, panoramic images. We thus do not include techniques that rely on image sequences or video to track features, perform structure-from-motion, or construct maps. This use of image pairs also matches the way in which our cleaning-robot framework employs visual pose estimation, as discussed in Section 2.5.2. Since our robot is equipped with a single monocular camera, we cannot acquire stereo or depth images. Similarly, we do not take into account methods that rely on robot odometry or additional sensors.

For the feature-based methods, we include a variety of feature detectors and descriptors combined with a brute-force matcher. From the resulting feature pairs, we estimate the relative pose using 5-point and planar 2-point algorithms with outlier rejection through standard RANSAC. We also investigate the effects of a final iterative refinement on the pose estimate. Regarding the holistic methods, our research group tested several warping methods in earlier studies [100, 103, 102]. Based on these results, we use min-warping both for this study and in our cleaning-robot framework: Except for one much slower but only marginally better method, it produced the best relative-pose estimates. In an earlier publication [49], we also tested a holistic method based on the block-matching algorithm [151]. Since the performance of this method remained notably below that of the other candidates, we choose to omit it from this chapter.

Figure 3.1 outlines how the feature-based and holistic methods estimate the relative robot pose from the camera images. Details on the selection of methods and their parameters and use are found in Section 3.2. As we have discussed in Section 2.1, small domestic robots such as ours have to operate with limited onboard computational resources. The real-time constraints faced by our robot allow us to formulate meaningful time constraints for our evaluation. Beyond our cleaning

[2] The class of holistic methods is not equivalent to that of visual-appearance methods, as at least some appearance methods use local visual features such as SIFT or SURF [16, 29].

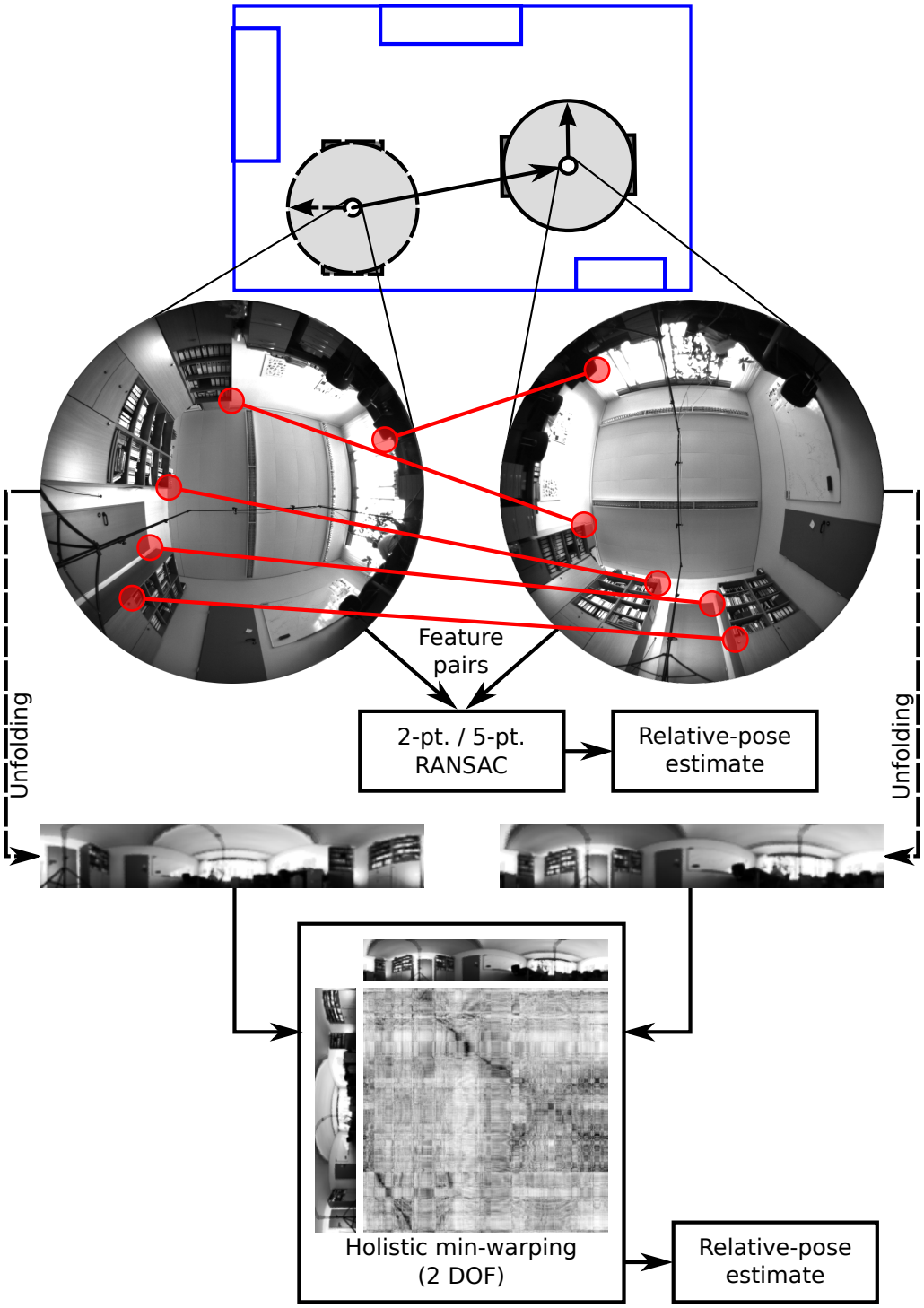


Figure 3.1.: From two camera images to a relative pose estimate: Features are extracted and matched for the two images captured by a robot (**top**). The relative robot pose is then estimated from the feature locations in the two images. Alternatively, a holistic method determines the relative pose from all pixels in the low-resolution unfolded images (**below**). The feature matches shown here are merely an illustration, and the actual number of features is much higher.

robot, we also want to provide insights for a variety of other applications. We therefore analyze our results using general criteria for quality, speed, and robustness.

We have organized the remainder of this chapter as follows: Section 3.2 describes the general design and acquisition of the image databases which we created for this study (Section 3.2.1). Next, we give an overview of holistic (Section 3.2.2) and feature-based (Section 3.2.3) pose-estimation methods, and explain our choice of methods. Finally, we describe our experiments and evaluation criteria in Section 3.2.4. Section 3.3 contains comprehensive results from our experiments, including measurements of quality (Section 3.3.1) and computational speed (Section 3.3.2). We also study the candidates' robustness to strong illumination changes or violations of the planar-motion assumption (Section 3.3.3). We discuss our results in Section 3.4, with an emphasis on the strengths and weaknesses of the individual methods. Lastly, we offer brief conclusions and suggest avenues for further research in Section 3.5.

3.2. Materials and Methods

In this section, we first introduce the image databases which we collected for this work (Section 3.2.1). We then discuss the holistic (Section 3.2.2) and feature-based (Section 3.2.3) pose-estimation methods evaluated in this study. Finally, we present our experiments and evaluation criteria in Section 3.2.4.

3.2.1. Image Databases

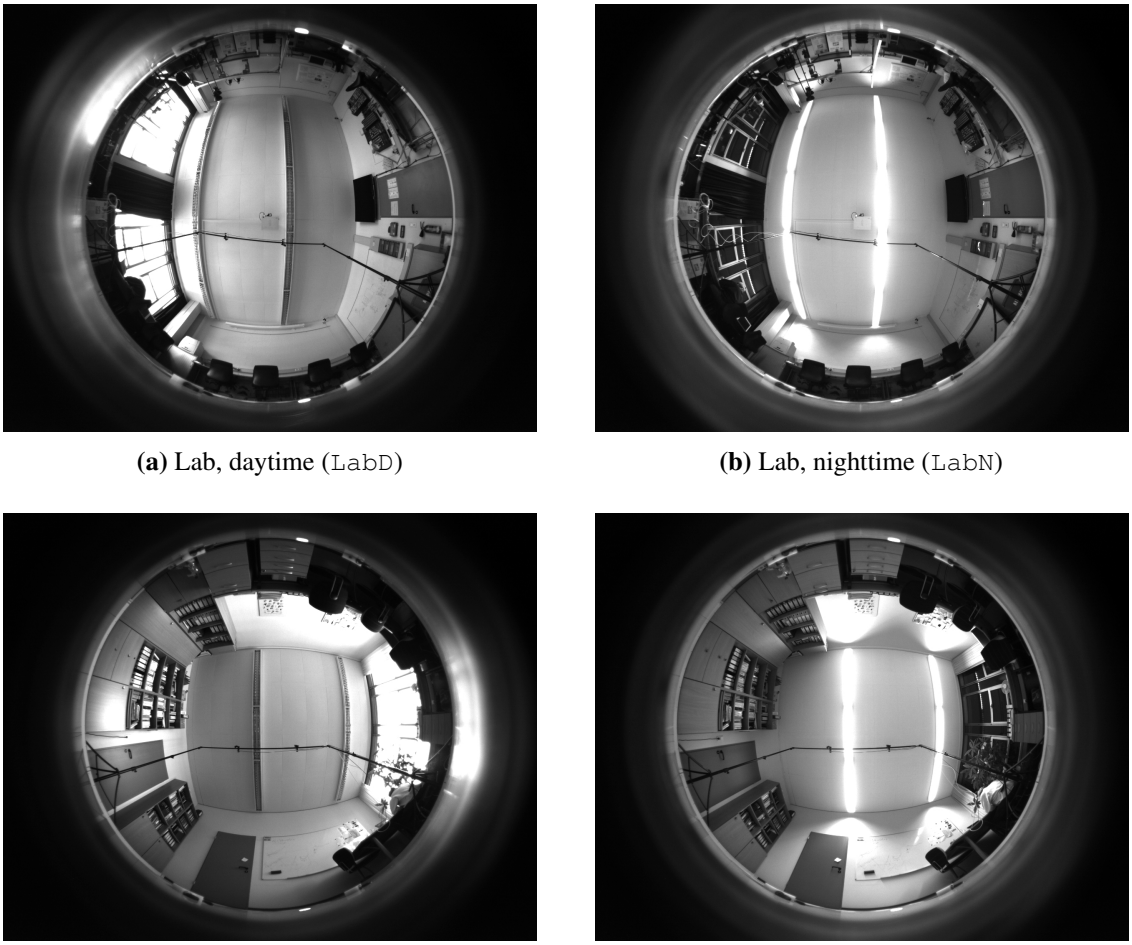
To meet the specific requirements of this comparative study, we recorded novel image databases. We found this necessary as existing databases do not provide the image types, calibration, or ground truth data suitable for our experiments. We acquired these images using our robot and its onboard panoramic camera, which we introduced in Section 2.2. For these experiments, we captured images using the camera's full 1280×1024 resolution. With the fisheye lens, the resulting usable image area is a disc with a diameter of approximately 900 pixels, as shown in Figure 3.2. A controller adjusted the camera exposure time with the goal of maintaining a constant average image brightness.[3] All images and related information are made available for download [47].

Our robot recorded images on a regular grid with $\approx 15\,\mathrm{cm}$ spacing. This grid merely ensures that there is an approximately uniform image coverage of the chosen area. The grid is not used to provide ground truth data,[4] therefore precise positioning of the robot at each location was not required. Using wheel odometry, the robot autonomously captured images on each individual grid column. In between, we cleaned the lens and positioned the robot for the next column. For each grid location, we recorded images with four different robot orientations: $0°$, $90°$, $180°$ and $270°$. These orientations are parallel to the axes of the location grid. Each grid consists of 16×12 locations, covering an area of $225\,\mathrm{cm}$ by $165\,\mathrm{cm}$. With the four different orientations per location, a database contains 768 images in total. To provide a stable platform and to simplify the synchronization with the ground truth data, our robot remained stationary during each image capture. This behavior also avoids any motion blur.

The ground truth was acquired by overhead cameras, which tracked two differently-colored LEDs built into the robot's upper lid. From the LED positions in the overhead camera images, we calculated the robot position and orientation on the 2D floor plane. Specifically, the ground truth position corresponds to the location of the onboard camera. We can therefore directly compare

[3] The topmost $15°$ of azimuth were not included in the average brightness calculation, due to concerns regarding the effect of strong ceiling-mounted light sources in this area.

[4] Ground truth data is captured with an overhead video tracking system.



(a) Lab, daytime (LabD)

(b) Lab, nighttime (LabN)

(c) Office, daytime (OfficeD)

(d) Office, nighttime (OfficeN)

Figure 3.2.: Raw example images from each database, each captured at grid position (5,5).

relative poses estimated by the visual methods with those calculated from the ground truth. The tracking system uses calibrated cameras and achieves a resolution of a few millimeters.

In total, we gathered four image databases in two different environments under two illumination conditions. The Lab databases were recorded in a typical lab environment. LabD was recorded in the early afternoon, with the windows as the only source of light. The lab faces an interior courtyard, and there is no direct sunlight entering the room. For LabN, recording began 30 minutes after sunset. Two rows of overhead fluorescent lights provided the illumination. The Office database was recorded in an office. As before, OfficeD was recorded around noon, while OfficeN was captured at night under fluorescent illumination. The northeast-facing windows of the office did not allow any direct sunlight to enter the room. Figure 3.2 shows a raw example image from each database.

For our actual experiments, we randomly selected 10000 out of 589056 possible image pairs from each database. Experiments with strong illumination changes also use 10000 out of 1179648 image pairs for both the LabD-LabN and OfficeD-OfficeN database pairs, selected from both day-night and night-day combinations. We use these random subsets of image pairs to reduce the computation time taken by the experiments to a reasonable level. For any given database, the same set of image pairs is used for all experiments. To test a pose-estimation method, we compute the relative-pose estimate for each image pair. We then calculate the estimation error by comparing

(a) Lab, daytime (`LabD`)

(b) Lab, nighttime (`LabN`)

(c) Office, daytime (`OfficeD`)

(d) Office, nighttime (`OfficeN`)

Figure 3.3.: Unfolded and preprocessed images from each database, as used by min-warping. All images were taken at the same grid position and correspond to the raw example images in Figure 3.2.

these estimates to the ground truth.

3.2.2. Holistic Method: Min-Warping

Here, we provide a verbal description of the min-warping algorithm; details are available in the literature [100, 102, 101]. To reduce the degrees of freedom (DOF) in the relative-pose problem, min-warping assumes that robot motion is planar. Note that this matches the planar-motion assumption used in our cleaning-robot framework, as per Section 2.5.2. For a scene of unknown scale, this leaves only the relative bearing angle α and orientation ψ as the parameters of the relative pose. To estimate the relative pose, min-warping performs a systematic search for the (α, ψ) pair with the best match score. The match score expresses how well the given image pair agrees with a hypothetical relative pose, as explained further below.

3.2.2.1. The Min-Warping Algorithm

Min-warping operates on "unfolded" images, as shown in Figure 3.3. In this type of panoramic image, each image column with index i corresponds to a particular azimuth in robot coordinates. Min-warping also assumes that all pixels in a column correspond to points with the same ground distance from the camera. Even though this equal-distance assumption is often violated, min-warping produces good estimates of the relative pose in many environments. Under these assumptions, image columns can be used as landmarks. A column at position i can only reappear in another image at certain positions $j \in M_i$, where the search range M_i depends on the pose hypothesis (α, ψ). When the ground distance for a column differs between the images, the vertical magnification at which it appears will also change. If the column i is shifted to position j, the relative vertical scale $\sigma_{i,j}$ is determined by the pose hypothesis.

Min-warping now estimates the shift for each image column by finding its closest match within the search range in the other image. To identify matches, a visual distance $d(i, j, \sigma_{i,j})$ is calculated from the columns' pixels [101]. This pixel-wise comparison is a typical characteristic of holistic methods. In this chapter, we use a distance measure which averages the normalized sum of absolute differences (NSAD) and the normalized sum of absolute differences of absolute values (NSADA) on edge-filtered images, as described in [99]. For each column i, min-warping then searches for the match $j \in M_i$ with the lowest distance $\breve{d}(i) = \min_j d(i, j, \sigma_{i,j})$. The sum $\sum_i \breve{d}(i)$ over all image columns gives the match score for a hypothesis (α, ψ). Finally, the hypothesis $(\breve{\alpha}, \breve{\psi})$ with the lowest score is the relative-pose estimate determined by min-warping. This corresponds to the pose hypothesis for which the best-case disagreement between the images is minimized.

Several optimizations markedly increase the speed of this algorithm: We precompute the search range M_i at which each column i can appear in the other image. For each possible column pair (i, j), we also precompute the relative vertical scale $\sigma_{i,j}$. Binning is used to map these continuous $\sigma_{i,j}$ to a finite set of discrete σ_k. For a given image pair, the distance $d(i, j, \sigma_k)$ is precomputed for all column pairs (i, j) and discrete σ_k. The precomputed $d(i, j, \sigma_k)$ for a given scale σ_k are called a scale plane, and the scale planes for all possible σ_k form the scale-plane stack.

Here, we use a heuristic called compass acceleration to speed up the search for the best pose hypothesis [102]: First, a fast approximate search identifies the most promising orientation estimates $\breve{\psi}$ from the scale-plane stack. The search for the best hypothesis $(\breve{\alpha}, \breve{\psi})$ is then limited to this fraction of ψ. This speeds up the second phase of min-warping, with only a slight impact on the quality of the results. When using min-warping to correct our cleaning robot's pose estimate in Section 2.5.2, we can reduce the execution time even further: Based on the robot's uncorrected pose estimate, we first precompute the expected relative pose between the robot and a map node. We then use this coarse estimate to restrict the search space for both the bearing $\breve{\alpha}$ and orientation $\breve{\psi}$ to one-quarter of their full range (Section 3.2.2.1). This reduces the overall search space by a fraction of $\frac{15}{16}$, thus speeding up min-warping's second phase. In our cleaning-robot framework, this search-space restriction replaces the compass-acceleration heuristic. To allow for a fair comparison between the various methods, we do not use this application-specific optimization in this chapter.

Due to its regular structure, the entire min-warping algorithm can be parallelized efficiently using vector instructions, such as SSE and AVX on x86 CPUs. We employ such an accelerated implementation for the tests in this work [98].[5] To optimally exploit the parallelism of vector instructions, our implementation uses small integer data types for almost all stages of the computation. We provide the specific min-warping parameters used for our experiments in Table A.2.

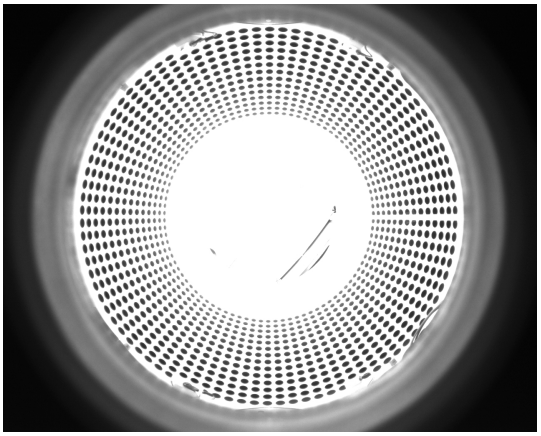
3.2.2.2. Calibration and Preprocessing

The camera mounted on our robot can operate in a 640×512 pixel half-resolution mode. Min-warping does not require high-resolution camera images, and a lower resolution reduces the image-processing time. Consequently, our cleaning-robot framework use this half-resolution mode for all images captured during a regular cleaning run. In these experiments, we simulate the half-resolution mode by reducing the resolution of the database images through binning.

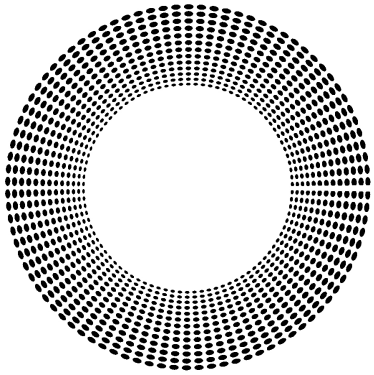
As mentioned, min-warping requires some moderate image preprocessing. This includes unfolding the fisheye image (Figure 3.2) into a low-resolution panoramic image (Figure 3.3). To determine the mapping between the fisheye image and the unfolded panoramic image, we perform an initial calibration step: First, we place and align the robot inside a cylinder covered with a grid of black dots. These dots are visible in a calibration image captured by the robot's camera, which is shown in Figure 3.4a. Next, we use thresholding and segmentation to extract these dots, as seen in Figure 3.4b. The bearing of each dot in robot coordinates can easily be calculated from the known sizes of the grid and cylinder.

We thus know the image coordinate and robot-relative bearing for each dot. From this, we calculate a mapping between the camera image and the bearing vectors through bilinear interpolation. Based on this result, we can now create unfolded panoramic images from the fisheye camera images. In these unfolded images, each column corresponds to a specific azimuth in robot coordinates, while each row corresponds to a certain elevation angle. Figures 3.4c and 3.4d demonstrate the relationship between the camera and unfolded images. Note that elevation angles above $\approx 45°$, corresponding to the central region of the fisheye image, are not covered by the calibration pattern. Even if we would extend the pattern, the dots would appear very close together in the image, making

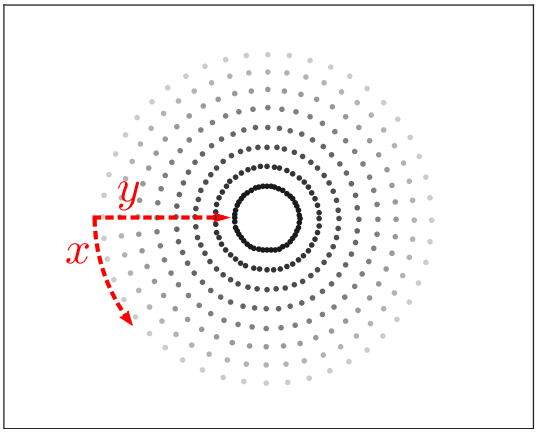
[5] A version of this implementation is available to the public at `http://www.ti.uni-bielefeld.de/html/people/moeller/tsimd_warpingsimd.html`.



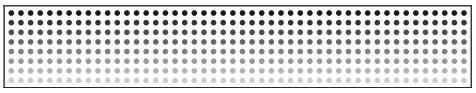
(a) Calibration image



(b) Extracted grid pattern



(c) The final mapping used for image unfolding. Each point corresponds to a pixel in the camera image that is mapped to a pixel in the unfolded image (Figure 3.4d). For the sake of clarity only every sixth point is shown.

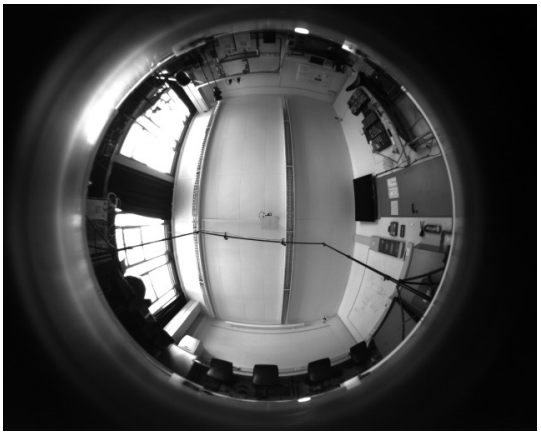


(d) The structure of the unfolded image. Each point represents a pixel, and each row of pixels corresponds to one of the concentric rings in Figure 3.4c. Points of equal brightness represent pixels of equal elevation angle.

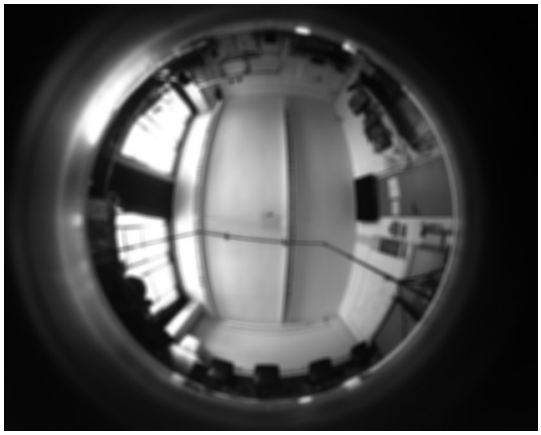
Figure 3.4.: Calibration for image unfolding, as described in Section 3.2.2.2.

calibration difficult. As our fisheye lens uses equidistant projection, we instead approximate the high-elevation mapping through linear extrapolation. The extrapolation coefficient for this step is derived from the calibration toolbox used in Section 3.2.3.2 [132].

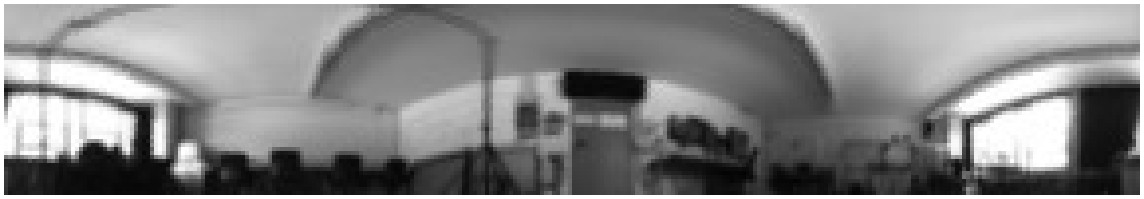
For our experiments with holistic methods, we use unfolded images with a resolution of 288×48 pixels, which encompass elevation angles from $0°$ to $75°$. Note that, as the unfolded image does not cover elevation angles above $75°$, min-warping cannot utilize the innermost region of the fisheye image. To avoid aliasing, we apply an averaging filter with a 7×7 mask to the camera image; this also lowers the effective resolution. Finally, histogram equalization is applied to the unfolded image to improve invariance under different illumination conditions. Here, we use the implementation from the OpenCV library [21] on integer images. Figure 3.5 shows the effects of these preprocessing steps. Further examples of unfolded and preprocessed images are shown in Figure 3.3.



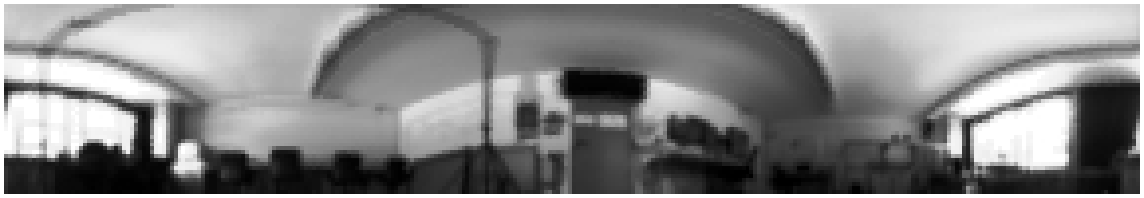
(a) Raw camera image, from the `LabD` dataset.



(b) Figure 3.5a with an averaging filter.



(c) Unfolded image generated with the mapping shown in Figures 3.4c and 3.4d.



(d) Unfolded image after histogram equalization. In practice, this effect can be subtle.

Figure 3.5.: Preprocessing for holistic methods: Prefiltering, unfolding and histogram equalization.

3.2.3. Feature Methods

There are numerous feature-based techniques that can be used for visual relative-pose estimation [11, 43, 131, 134, 53, 55]. For this comparison, we decided to focus our attention on a selection of popular methods. We are less experienced in implementing feature-based algorithms compared to holistic methods. To reduce the risks posed by a bad implementation, we rely on established libraries where possible. Table A.1 lists the most important libraries and programs used, together with their version number. Unless otherwise noted, we also used the default parameters for each method within these libraries. We include the feature-method parameters in Tables A.3a to A.3d.

3.2.3.1. Detection and Extraction

As feature detectors and descriptors, we chose SIFT, SURF, ORB and BRISK. SIFT is a widely used detector and descriptor known for its robustness and the quality of its results [89, 11, 43]. As initial results with SIFT were promising, we include SURF as a potentially faster alternative [7]. In contrast, ORB represents a newer class of algorithms that use fast detectors and binary descriptors [129, 11, 43]. Its high speed, especially compared to SIFT, is a strong argument for including ORB [134]. BRISK also uses binary descriptors and offers robust results, making it an interesting competitor to ORB [84]. In all cases, we use the implementation from the popular OpenCV library [21]. All four methods are invariant under rotation and some illumination changes. The former is especially important, since we extract features directly from the original fisheye image: Given the equidistant projection of our fisheye lens, a change in camera pose may also change the orientation of the features. SIFT, SURF, BRISK and the OpenCV implementation of ORB are also scale-invariant through the use of a scale pyramid.

To keep the number of detected features within a reasonable range, the ORB implementation requires an upper limit on the number of features. We use the OpenCV default of 500 as well as a value of 1500. The latter is close to the average number of features that SIFT detects on these databases (Table 3.3). Throughout this chapter, we call these two variants ORB-500 and ORB-1500. With the OpenCV default parameters, SURF also generates a large number of features — often more than twice as many as SIFT. We therefore adjusted the SURF feature detection threshold to reduce the average number of features to about 1500 (see Table A.3d). Without this adjustment, the matching and pose estimation steps take much longer compared to SIFT or ORB, without any great improvement in quality.

Unlike min-warping, the feature methods do not depend on a specific type of camera projection. This simplifies preprocessing, since the input images in equidistant projection are used without reprojection (see Figure 3.2 for example images). However, these raw camera images contain areas that are not exposed by the fisheye lens or that only show the robot's chassis. To exclude these areas, we crop the image and mask out the pixels below the horizon. This reduces the usable image area to a disc with a diameter of ≈ 880 pixels.

We employ the OpenCV implementation of the brute-force matcher to identify pairs of features across two images. For each feature from one image, the matcher selects the nearest neighbor from another image through linear search. While faster, approximate matchers exist [108, 109], the brute-force matcher it is guaranteed to find the best match for each feature. In general, linear search is the fastest exact solution for this high-dimensional nearest neighbor problem [108]. The matcher also performs mutual consistency checking to reduce the number of false matches: Feature A is only matched with its nearest neighbor B in another image if A is also the nearest neighbor of B. We employ the L2 norm to calculate the distance between the SIFT and SURF feature vectors. For the binary descriptors extracted by ORB and BRISK, we use the much faster Hamming distance instead.

3.2.3.2. Relative-Pose Estimation

To estimate the relative pose of the robot, we need to know each feature's bearing vector in robot coordinates. We use a camera projection model to calculate the bearing vectors $\bar{p}$ and $\bar{q}$ from the feature keypoint locations. $\bar{p}$ represents the bearing in one camera image, while $\bar{q}$ denotes the matching feature in the other image. For this step, we rely on the widely-used calibration toolbox introduced by Scaramuzza, Martinelli, and Siegwart [132]. This calibration technique supports fisheye lenses up to $195°$ and thus meets the demands of our camera. Careful calibration resulted in a reported reprojection error of 0.10 pixels.

To transform the bearing vectors from camera to robot coordinates, we measure the orientation of the camera relative to the robot chassis. Here, we use the calibration cylinder described in Section 3.2.2.2: Given the intrinsic calibration, we first calculate the bearing vectors of the grid dots in the camera coordinate system. As before, the bearings of these dots in the robot coordinate system are already known. We then use the eigensolver described in [78] to find the relative orientation matrix O between the robot and the camera. Finally, the simple multiplications $p = O\bar{p}$ and $q = O\bar{q}$ transform the bearing vectors from camera to robot coordinates.

We compare two different algorithms that estimate the relative pose from these feature bearing vectors: Like min-warping, the two-point method described in [17] utilizes the planar-motion assumption. Its relative-pose estimate is thus limited to two degrees of freedom. We also test the five-point method introduced by Stewenius, Engels, and Nistér [138], which extends a well-known previous work by Nistér, Naroditsky, and Bergen [117]. This method estimates the relative pose in five degrees of freedom, six minus the unknown scale. It is not limited to planar motion and thus robust against violations of this constraint. Here, we use the five-point implementation provided by the OpenGV library [77]. For the two-point method, we add our own implementation based on the description in [17]. In practice, the five-point method actually uses three additional feature pairs for disambiguation and decomposition of the results. The two-point method also requires one additional pair for this reason. Combining SIFT, SURF, BRISK and the two ORB variants with both the two-point and five-point algorithm results in ten distinct candidates.

The two- and five-point methods are highly susceptible to outliers, for example from incorrectly matched features. A standard random sample consensus (RANSAC) scheme is therefore used to find outlier-free sets of feature pairs [44, 131, 55]. Under RANSAC, the methods discussed above generate pose hypotheses from randomly drawn minimal point sets. The algorithm then keeps the hypothesis which is in consensus with the largest number of other point pairs. To decide whether a point pair is in consensus with a pose hypothesis, OpenGV calculates the reprojection error e. If e is greater than a threshold ϵ, the feature pair is rejected. We could estimate ϵ based on the reprojection error of the intrinsic calibration. However, a search revealed that the value of ϵ which minimizes the actual pose-estimation error is not constant. Among others, it depends on the image set, feature method, pose-estimation method and distance between the image-capture locations. We decided to use $\epsilon = 3 \times 10^{-5}$, which gives good results for all the methods we test. RANSAC terminates once the estimated probability of an outlier-free hypothesis or the number of iterations exceed predetermined thresholds. Finally, an optional refinement step may improve the best hypothesis using the information from all feature pairs that are in consensus with it. In some of our experiments, we employ the OpenGV implementation of the Levenberg-Marquardt algorithm to perform this step [85, 95, 77].

As shown in Figure 3.6, the number of RANSAC iterations required for a given success rate grows rapidly as the rate of incorrectly matched features increases [53]. For an application with real-time constraints, the number of iterations thus has to be limited. The OpenGV default of 250 iterations caused high pose-estimation errors in many of the experiments. Unless stated otherwise, we therefore increased the limit to 1000 iterations. This value usually keeps the time taken by RANSAC comparable to the time taken by the other steps. For some methods, we later vary this

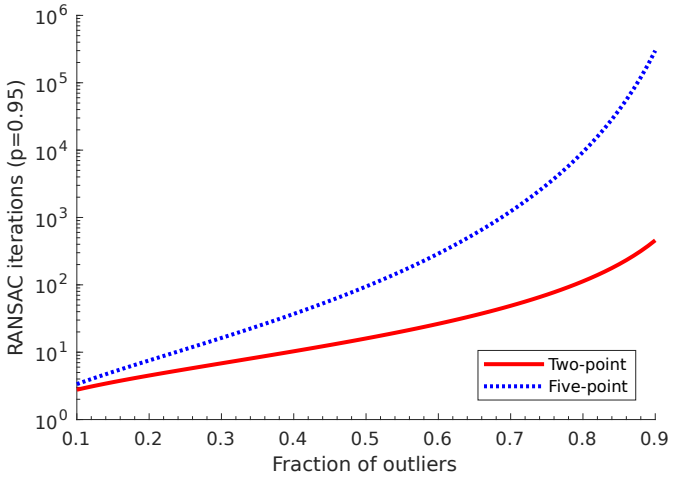


Figure 3.6.: This logarithmic plot shows the number of RANSAC iterations required to identify an outlier-free set with a probability of $p = 0.95$. It is based on the formula from [53], which assumes that the number of feature pairs is much larger than the size of the outlier-free set. The five-point and two-point methods require three or one additional feature pairs, respectively. These are required for disambiguation and decomposition of the pose estimate. The actual required RANSAC iterations are thus likely to be higher than those shown in this figure.

limit to study its effect on the pose-estimation errors.

3.2.4. Evaluation Criteria

We group our evaluation criteria into three categories: The quality criteria in Section 3.2.4.1 focus on the general accuracy of the estimated poses. Section 3.2.4.2 discusses how we measure the computational speed of the candidates. Finally, we evaluate the effects of strong illumination changes or violations of the planar-motion assumption according to Section 3.2.4.3.

3.2.4.1. Quality

We use the relative orientation and bearing errors to judge the quality of the pose estimates. Since our robot moves on a level floor, we only consider errors in the floor plane. For the 5-point method, we project the three-dimensional pose estimates onto this plane. Interpretation of these errors depends on the task at hand: For a simple visual homing task, the acceptable error can be quite large if catastrophic errors are avoided. In SLAM applications low errors are desirable, because the resulting maps and absolute robot pose estimates can be more accurate [40, 134, 55]. For the same reason, we prefer low errors when using visual pose estimation in our cleaning-robot framework.

The magnitude of the relative-pose errors strongly depends on the distance between the two image-capture locations. In turn, the typical distance between capture locations depends on the application at hand. For example, homing tasks or sparse topological maps will usually involve longer distances. When using dense maps, or when images are acquired frequently as with a video sequence, the distances will be shorter. For our cleaning robot, we typically estimate poses between locations which are $25\,\mathrm{cm}$ to $50\,\mathrm{cm}$ apart. To provide estimates of the error magnitude for a given application, we plot the errors over the distance between the image-capture locations.

3.2.4.2. Speed

When running on an actual robot, visual pose estimation may be subject to strict real-time constraints. We use two scenarios to evaluate the time required by the pose-estimation candidates. The first scenario is built around our cleaning-robot prototype. As discussed in Section 2.2, a dual-core Intel Atom N2600 1.6 GHz processor — a typical CPU for embedded applications — controls the entire robot. One core is available for visual pose estimation, while the second core performs other tasks. Ideally, the robot should be able to perform several pose estimates per second for visual homing or as part of a SLAM system. For example, our robot calculates about six estimates per second while extending the map during a cleaning run (Section 2.5.2). The second scenario uses a modern desktop PC with an Intel Core i7-4790K quad-core CPU, which supports vector instructions up to AVX2. Such a high-performance system might be available on larger robotic platforms, or for robots that offload computations to external systems. It also offers an outlook into the future, where similar processors might become feasible for smaller platforms like our domestic cleaning robot. In both scenarios, we measured the elapsed wall-clock time using the system's internal monotonic high-resolution clock. The time required for ancillary operations — like trivial conversions between data structures — was not included. These values are small by comparison and highly implementation-dependent.

The various implementations used in this study do not consistently support parallel computing on multiple CPU cores. Many of them do not support parallelization at all, even where the underlying algorithm would allow this. Using multiple CPU cores would thus distort the performance measurements. To avoid this problem, we limit our test programs to one CPU core each by using Linux's `numactl` command. This also avoids the cost associated with the migration of a single thread between CPU cores.

As we discussed before, the camera on our robot provides a 640×512 half-resolution mode. Since min-warping does not require high-resolution images, it uses this mode to reduce preprocessing costs. For the sake of fairness, we also tested some of the feature-based methods on these lower-resolution images. This should reduce the time needed to detect and extract the features. However, the lower resolution may also reduce the precision of the measured feature bearings. Recalibrating the camera for the low-resolution mode could distort the experiments if the calibration errors change. Instead, we recompute the camera parameters directly from the original, full-resolution calibration results.

3.2.4.3. Robustness

While the planar-motion assumption simplifies the pose-estimation problem, it also makes the estimation process more fragile: Even in planar-floor domestic environments, movement on rough surfaces, accelerating motion, or extrinsic calibration errors can lead to poses outside of the ground plane in the robot's coordinate system. Booij and Zivkovic [17] noted that the two-point algorithm performs poorly if the planarity constraint is not met: In their experiments, the performance of the two-point algorithm on an actual robot remained below the level achieved in simulations. The authors suggested a tilt caused by the robot's movement as a possible explanation. During preliminary cleaning-robot experiments, we observed similar problems if the robot was tilted on rough ground.

To study this effect, we include tilt-induced violations of the planar-motion assumption in our experiments. Additionally, we will examine the tilting problem in depth throughout Chapter 4. For our robot, common indoor causes of rough ground include gaps between floor tiles, carpets or door thresholds. We conducted a survey of these causes in typical domestic environments, which we will fully discuss in Section 4.2.3. Using a statics model of our cleaning robot, we found that the resulting tilts would usually be $\approx 3^\circ$ or less. We therefore perform experiments with tilts up to 3° in

the forward, backward, and sideways directions. For the feature methods, we simulate the tilts by rotating the feature bearing vectors. This is faster than simulating tilts through distortions of the camera images, and a reasonable approximation for small angles. For the min-warping method, we modify the image unfolding process, so that the resulting low-resolution panoramic images already incorporate the tilt.[6]

In this study, we also examine each candidate's robustness to varying illumination conditions: In a domestic environment, the lighting conditions are likely to change over time. For a cleaning robot, these changes may be especially stark if the robot has to recharge its batteries during a cleaning run, or when reusing previously-created maps. The pose-estimation methods must thus be able to cope with such changes when operating over longer timespans. The SIFT, SURF, ORB and BRISK methods already offer some degree of illumination invariance. Min-warping also incorporates illumination invariance into the distance measure used to compare image columns [101]. Consequently, we perform cross-database experiments between daytime and nighttime images to test the effectiveness of these invariances.

3.3. Results

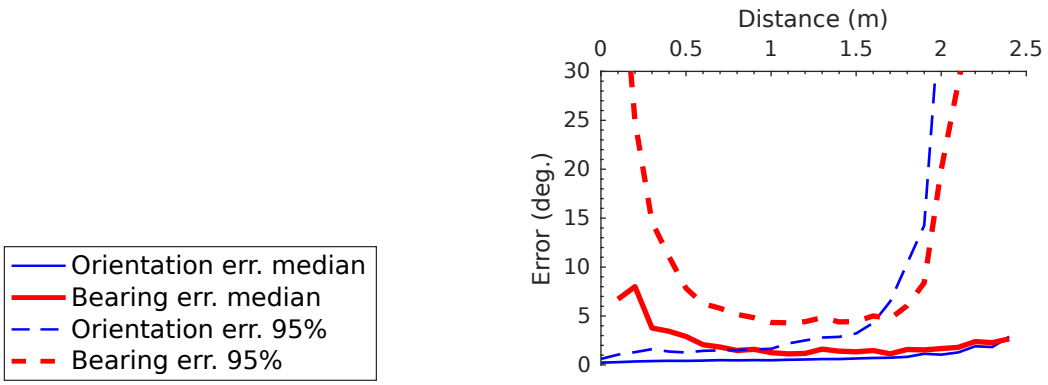
3.3.1. Quality

Figures 3.7 to 3.12 show the pose-estimation errors depending on the distance between the two images. Here, images are only combined with other images from the same database, and thus the environment and illumination are approximately constant. We also generated such plots for the half-resolution camera images. We include such results for ORB in Figures 3.13 and 3.14, as these exhibited the most noteworthy changes. The solid lines in each plot represent the median errors in the estimated orientation and bearing. Besides the median, the plots also contain the 95th error percentile as a dashed line. We choose this measure to indicate the magnitude of larger errors that might still occur quite frequently. Even though larger values do occasionally occur, we fix the vertical axis to a maximum of $30°$. This uniform scale makes it easier to compare the different figures.

To create these figures, we sort the image pairs into bins with a size of $10\,\mathrm{cm}$. The first bin, with the value appearing at $0.1\,\mathrm{m}$ in the plots, thus contains image pairs with a distance of $5\,\mathrm{cm}$ to $15\,\mathrm{cm}$. We leave out long-distance bins that contain less than 100 image pairs, since this low sample size makes the results less conclusive. Additionally, these long-distance image pairs can only occur between a few regions in each database. This makes them less meaningful and prone to large fluctuations in the error magnitudes. We also discard the bearing error for image-capture distances of less than $5\,\mathrm{cm}$: As mentioned, our databases consist of $15\,\mathrm{cm}$ grids, with four different orientations captured per position. Therefore, any close-range image pairs must come from the same grid position, and the bearing cannot be determined.

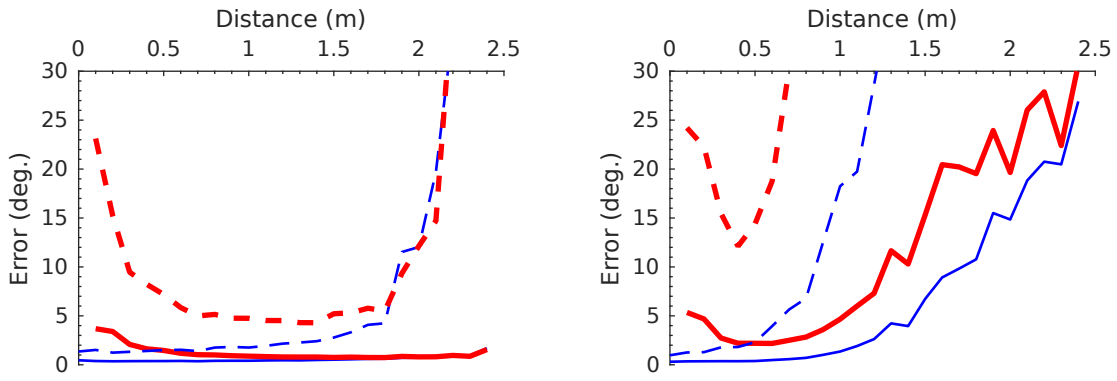
In Table 3.1, we also list the mean, median, and 95th percentile of the errors for all image pairs in the random sample. These offer a closer look at the differences in the pose-estimation quality, making it easier to compare the candidates. Due to the nature of the image databases, the distances between image-capture locations are not uniformly distributed. As seen in Figure 3.15, some distances occur more often than others and thus have a stronger influence on the values in this table. Note that these image-capture distances are often larger than the $25\,\mathrm{cm}$ to $50\,\mathrm{cm}$ typically encountered by our cleaning robot. We include such image pairs to ensure that our results are also

[6] To preserve the effect of small simulated tilt angles, we use bilinear interpolation when unfolding the tilted camera images. This slightly improves results compared to our regular experiments, where a nearest-neighbor interpolation is used to speed up the unfolding. We also noticed that a small simulated tilt ($\approx 0.1°$) in one of the directions actually slightly improves the min-warping results. This effect does not occur for the feature-based methods. We suspect that this is due to a minor residual error in the extrinsic camera calibration used by min-warping.



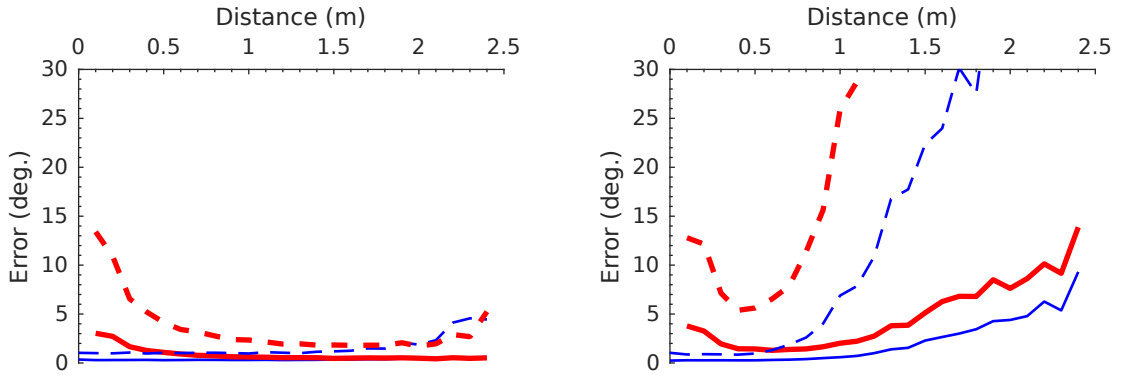
(a) Legend **(b)** Min-warping

Figure 3.7.: Pose-estimation errors depending on the image-capture distance, using **min-warping** and images paired within the **same database**. Pairing images only within the same database keeps illumination and environment approximately constant. Errors are given in degrees, distances in meters. Combined data from all four databases, 10000 random image pairs per database.



(a) ORB-500, two-point **(b)** ORB-500, five-point

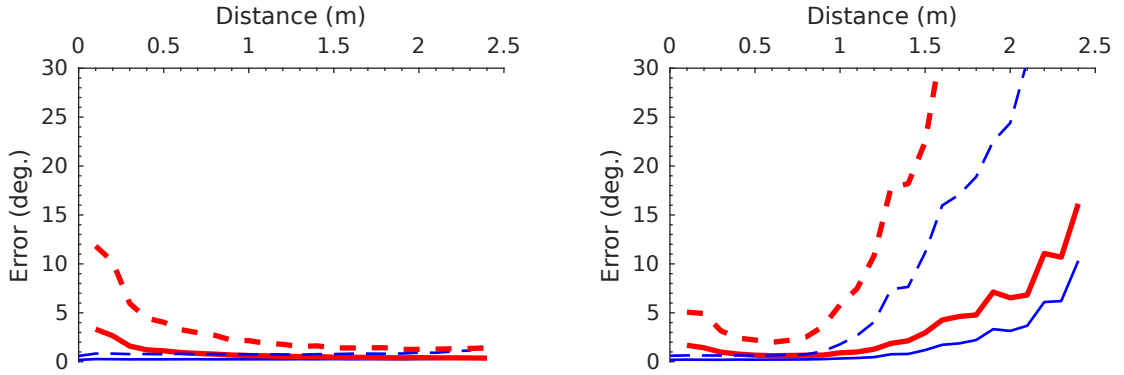
Figure 3.8.: Pose-estimation errors depending on the image-capture distance, using up to $n = 500$ **ORB features** and **full-resolution images** paired within the **same database**. Other settings and legend as in Figure 3.7.



(a) ORB-1500, two-point

(b) ORB-1500, five-point

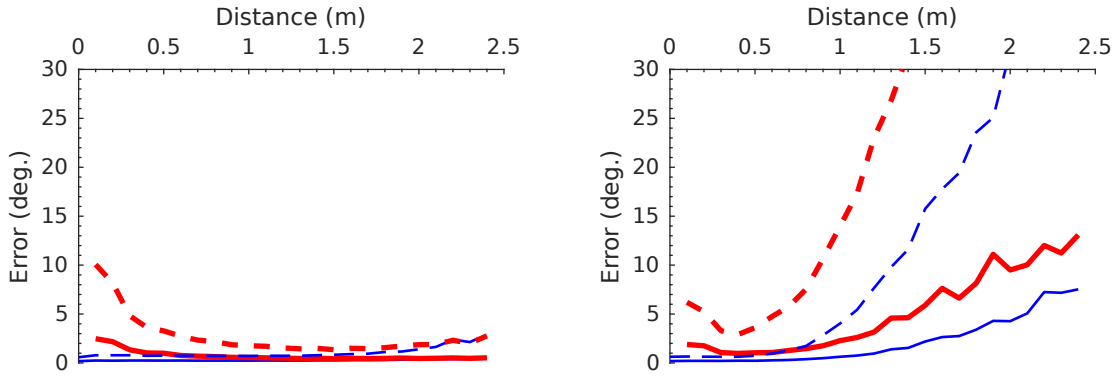
Figure 3.9.: Pose-estimation errors depending on the image-capture distance, using up to $n = 1500$ **ORB features** and **full-resolution images** paired within the **same database**. Other settings and legend as in Figure 3.7.



(a) SIFT, two-point

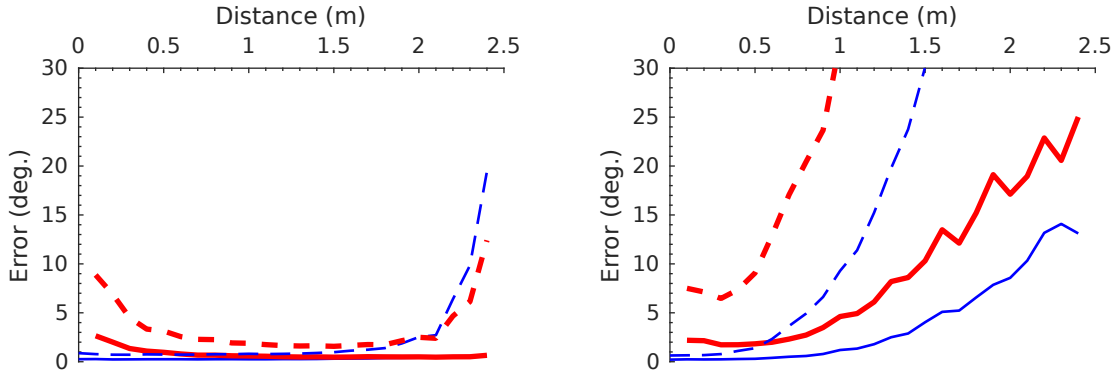
(b) SIFT, five-point

Figure 3.10.: Pose-estimation errors depending on the image-capture distance, using **SIFT features** and **full-resolution images** paired within the **same database**. Other settings and legend as in Figure 3.7.



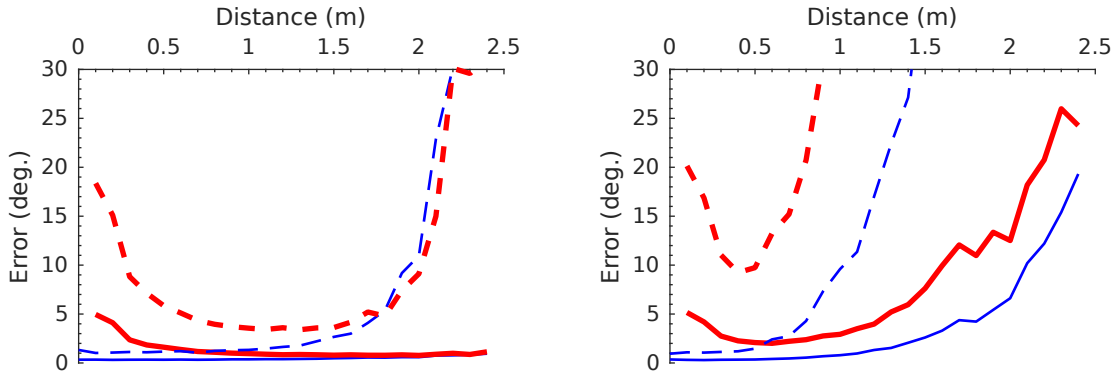
(a) SURF, two-point **(b)** SURF, five-point

Figure 3.11.: Pose-estimation errors depending on the image-capture distance, using **SURF features** and **full-resolution images** paired within the **same database**. Other settings and legend as in Figure 3.7.



(a) BRISK, two-point **(b)** BRISK, five-point

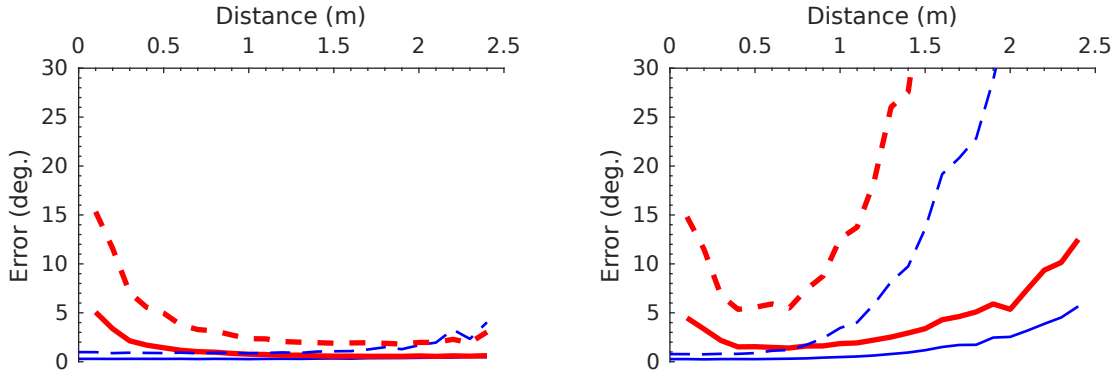
Figure 3.12.: Pose-estimation errors depending on the image-capture distance, using **BRISK features** and **full-resolution images** paired within the **same database**. Other settings and legend as in Figure 3.7.



(a) ORB-500, two-point

(b) ORB-500, five-point

Figure 3.13.: Pose-estimation errors depending on the image-capture distance, using up to $n = 500$ **ORB features** and **half-resolution images** paired within the **same database**. Other settings and legend as in Figure 3.7.



(a) ORB-1500, two-point

(b) ORB-1500, five-point

Figure 3.14.: Pose-estimation errors depending on the image-capture distance, using up to $n = 1500$ **ORB features** and **half-resolution images** paired within the **same database**. Other settings and legend as in Figure 3.7.

Method	Orientation error (°)			Bearing error (°)		
	Mean	**Median**	**95%**	**Mean**	**Median**	**95%**
ORB-500, two-point	1.22	0.46	2.53	2.21	0.97	7.00
ORB-500, five-point	8.85	1.51	43.42	21.01	5.67	112.87
ORB-1500, two-point	0.47	0.33	1.18	1.08	0.66	3.40
ORB-1500, five-point	3.40	0.66	16.10	8.58	2.52	39.76
SIFT, two-point	0.32	0.25	0.80	0.99	0.64	3.07
SIFT, five-point	1.99	0.40	9.67	4.24	1.21	19.13
SURF, two-point	0.33	0.26	0.85	0.88	0.60	2.60
SURF, five-point	2.83	0.63	12.53	6.84	2.41	28.30
BRISK two-point	0.51	0.28	0.94	0.97	0.62	2.68
BRISK five-point	5.84	1.08	25.74	13.39	4.49	57.48
Min-warping	1.89	0.54	3.52	3.02	1.73	8.20

Table 3.1.: Pose-estimation errors for **full-resolution image pairs** within the **same database**. As images are paired within the same database, the illumination and environment are approximately constant. Mean, median, and 95th percentile of errors are given in degrees. Calculated from 10000 random image pairs for each of the four databases. For the feature methods, RANSAC was limited to 1000 iterations per image pair. The min-warping results are identical to Table 3.2 and included for comparison only.

applicable to other applications, which may involve longer distances. As before, image pairs with a distance of less than $5\,\mathrm{cm}$ are not included in the bearing error. We repeat these calculations for the half-resolution camera images in Table 3.2. Furthermore, Table 3.3 lists the number of features that are actually detected and matched by each candidate.

To study the effect of a post-RANSAC Levenberg-Marquardt optimization of the pose estimate, we compared the estimation error and homing time to the values observed without the optimization step. The homing time includes the preprocessing of a single image, combined with pose estimation on one image pair. Section 3.3.2 described this measure in greater detail. The resulting errors and execution times are listed in Table 3.4 and Table 3.12, respectively.

3.3.2. Speed

Tables 3.5, 3.6 and 3.9 list the computation time required for the visual pose-estimation steps, measured according to Section 3.2.4.2. As this time can vary depending on the specific image pair, we list both the median and the 95th percentile. A real-time system must be able to accommodate the 95th percentile execution times, as these can still occur quite frequently. Table 3.5 shows the results for the embedded Atom N2600 platform, Table 3.6 contains the same data for a modern desktop PC. Min-warping data for both platforms is listed in Table 3.9. We measured these results using the same random 4×10000 image-pair sample already described in Section 3.3.1. Each table contains the time required for feature detection and extraction, feature matching, and the pose estimation with RANSAC. The data for the half-resolution images is listed in Tables 3.7 and 3.8.

To make the execution times from Tables 3.5 to 3.9 more accessible, we constructed a "Homing" scenario: Here, a robot captures an image and performs visual pose estimation relative to a previously stored image. The total execution time thus combines the time required for preprocessing the new image and estimating the pose. We assume that the preprocessing or feature extraction result of the previous image has been cached. This means that no second preprocessing step is

Method	Orientation error (°)			Bearing error (°)		
	Mean	**Median**	**95%**	**Mean**	**Median**	**95%**
ORB-500, two-point	1.04	0.40	2.07	2.00	1.05	5.93
ORB-500, five-point	6.59	0.87	31.99	15.60	3.78	88.95
ORB-1500, two-point	0.42	0.31	1.03	1.25	0.81	3.80
ORB-1500, five-point	2.52	0.53	11.03	6.69	2.28	28.13
SIFT, two-point	0.33	0.27	0.84	0.93	0.61	2.81
SIFT, five-point	1.50	0.38	6.47	3.21	1.14	13.37
SURF, two-point	0.43	0.29	0.99	1.01	0.68	2.90
SURF, five-point	2.16	0.56	8.54	5.49	2.28	20.86
BRISK two-point	0.74	0.35	1.28	1.30	0.79	3.49
BRISK five-point	5.27	0.85	23.64	12.03	3.76	56.34
Min-warping	1.89	0.54	3.52	3.02	1.73	8.20

Table 3.2.: Pose-estimation errors for **half-resolution image pairs** within the **same database**. As images are paired within the same database, the illumination and environment are approximately constant. Mean, median, and 95th percentile of errors, in degrees. Other settings are the same as for Table 3.1.

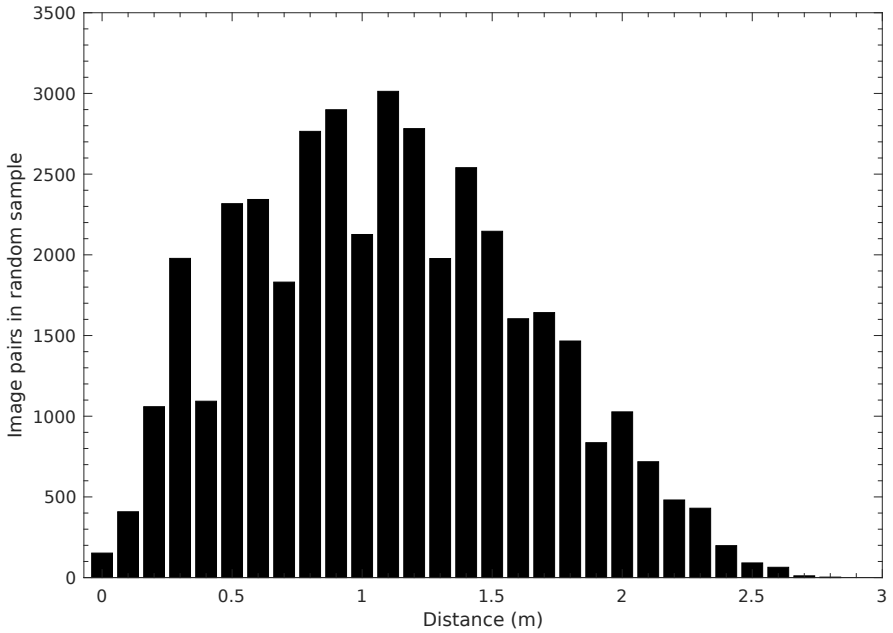


Figure 3.15.: Histogram of the distances between image-capture locations in the database sample. Here, we use the same $10\,\mathrm{cm}$ bins as for the error-distance plots. The grid structure of the image databases causes the prominent spikes in the plot.

Method	No. of Features		No. of Matches	
	Mean	Median	Mean	Median
ORB-500	500	500	262	258
ORB-1500	1500	1500	786	772
SIFT	1579	1586	914	903
SURF	1503	1515	837	828
BRISK	623	591	337	317
ORB-500 (half-res)	494	498	266	262
ORB-1500 (half-res)	1382	1387	745	733
SIFT (half-res)	710	711	422	415
SURF (half-res)	621	620	354	350
BRISK (half-res)	299	289	166	159

Table 3.3.: Number of features per image, and number of matched features per image pair. We use all 3072 images to calculate the number of features per image. The number of matches was determined from the union of the 10000 random image pairs per databases (images paired within the **same database**, 40000 total pairs).

Method	Orientation error (°)		Bearing error (°)	
	Median	95%	Median	95%
ORB-500 two-point	0.38 (0.46)	2.44 (2.53)	0.83 (0.97)	6.20 (7.00)
ORB-500 five-point	1.36 (1.51)	44.87 (43.42)	5.00 (5.67)	112.65 (112.87)
ORB-1500 two-point	0.27 (0.33)	0.98 (1.18)	0.52 (0.66)	2.55 (3.40)
ORB-1500 five-point	0.55 (0.67)	16.41 (16.72)	2.05 (2.53)	40.38 (40.87)
SIFT two-point	0.23 (0.25)	0.69 (0.80)	0.49 (0.64)	2.27 (3.07)
SIFT five-point	0.36 (0.40)	9.99 (9.88)	0.96 (1.22)	19.94 (19.59)
SURF two-point	0.23 (0.26)	0.75 (0.85)	0.49 (0.60)	2.07 (2.60)
SURF five-point	0.57 (0.63)	12.32 (12.53)	2.06 (2.41)	27.41 (28.30)
BRISK two-point	0.24 (0.28)	0.83 (0.94)	0.51 (0.62)	2.11 (2.68)
BRISK five-point	0.96 (1.08)	24.88 (25.74)	3.90 (4.49)	56.86 (57.48)

Table 3.4.: Effect of Levenberg-Marquardt optimization of the RANSAC result for **full-resolution** images, paired within the **same databases** (constant illumination). All errors in degrees. Original values without optimization given in parentheses, as per Table 3.1. Execution-time changes caused by the final optimization are covered by Table 3.12.

Method	Extraction		Matching		RANSAC	
	Median	95%	Median	95%	Median	95%
ORB-500, two-point	192.1	196.7	47.3	47.5	34.9	191.1
ORB-500, five-point	192.1	196.7	47.3	47.4	1567.2	1590.2
ORB-1500, two-point	218.1	221.8	428.0	428.6	74.8	515.1
ORB-1500, five-point	218.1	221.8	427.9	428.5	1929.8	1978.7
SIFT, two-point	2667.8	2802.3	1034.1	1360.1	77.7	576.2
SIFT, five-point	2667.8	2802.3	1033.8	1359.5	1988.2	2099.2
SURF, two-point	3582.1	3927.1	983.3	1214.5	184.0	570.6
SURF, five-point	3582.1	3927.1	983.3	1214.5	1960.0	2052.7
BRISK, two-point	163.0	185.1	125.0	243.0	114.7	291.7
BRISK, five-point	163.0	185.1	125.0	243.0	1623.5	1714.9

Table 3.5.: Feature-method execution times, **embedded system** (Atom N2600 CPU), **full-resolution images**. All times single-core wall-clock times in milliseconds. Extraction time measured across all images. Matching and RANSAC times measured for the union of the 10000 random image pairs per database (images paired within the **same database**, 40000 total pairs).

Method	Extraction		Matching		RANSAC	
	Median	95%	Median	95%	Median	95%
ORB-500, two-point	16.8	17.9	3.9	3.9	3.7	20.2
ORB-500, five-point	16.8	17.9	3.9	3.9	135.4	137.8
ORB-1500, two-point	20.0	20.6	34.2	34.3	8.0	55.1
ORB-1500, five-point	20.0	20.6	34.3	34.3	171.5	176.5
SIFT, two-point	222.7	234.1	55.6	73.5	8.4	61.8
SIFT, five-point	222.7	234.1	56.0	76.4	179.6	191.8
SURF, two-point	386.0	420.1	52.8	65.1	19.7	61.0
SURF, five-point	386.0	420.1	53.0	66.0	175.5	184.7
BRISK, two-point	21.3	23.5	9.8	19.0	12.1	31.2
BRISK, five-point	21.3	23.5	9.8	19.0	140.4	150.7

Table 3.6.: Feature-method execution times, modern **desktop PC** (Core i7-4790K CPU), **full-resolution images**. All times single-core wall-clock times in milliseconds. Extraction time measured across all images. Matching and RANSAC times measured for the union of the 10000 random image pairs per database (images paired within the **same database**, 40000 total pairs).

Method	Extraction		Matching		RANSAC	
	Median	95%	Median	95%	Median	95%
ORB-500, two-point	66.2	67.3	46.7	47.3	38.1	196.3
ORB-500, five-point	66.2	67.3	46.7	47.3	1573.9	1612.8
ORB-1500, two-point	87.6	90.6	365.5	397.7	74.7	497.1
ORB-1500, five-point	87.6	90.6	365.4	397.7	1881.4	1939.1
SIFT, two-point	866.5	930.4	204.0	262.3	30.6	268.2
SIFT, five-point	866.5	930.4	203.6	262.4	1664.3	1715.0
SURF, two-point	987.2	1093.6	159.0	192.6	71.6	253.6
SURF, five-point	987.2	1093.6	158.8	192.5	1644.2	1682.8
BRISK, two-point	59.2	66.7	29.9	51.1	53.7	149.5
BRISK, five-point	59.2	66.7	29.9	51.1	1511.8	1552.4

Table 3.7.: Feature-method execution times, **embedded system** (Atom N2600 CPU), **half-resolution images**. All times single-core wall-clock times in milliseconds. Extraction time measured across all images. Matching and RANSAC times measured for the union of the 10000 random image pairs per database (images paired within the **same database**, 40000 total pairs).

Method	Extraction		Matching		RANSAC	
	Median	95%	Median	95%	Median	95%
ORB-500, two-point	7.0	7.3	3.8	3.9	4.1	21.3
ORB-500, five-point	7.0	7.3	3.8	3.9	136.3	138.7
ORB-1500, two-point	9.5	10.4	29.3	31.8	8.0	53.3
ORB-1500, five-point	9.5	10.4	29.3	31.8	169.2	175.0
SIFT, two-point	72.7	78.2	11.5	14.8	3.3	28.6
SIFT, five-point	72.7	78.2	11.5	14.7	146.3	151.4
SURF, two-point	108.7	119.0	9.0	10.9	7.6	26.9
SURF, five-point	108.7	119.0	9.0	11.0	142.1	146.0
BRISK, two-point	7.5	8.5	2.4	4.0	5.6	15.7
BRISK, five-point	7.5	8.5	2.4	4.0	129.3	133.3

Table 3.8.: Feature-method execution times, modern **desktop PC** (Core i7-4790K CPU), **half-resolution images**. All times single-core wall-clock times in milliseconds. Extraction time measured across all images. Matching and RANSAC times measured for the union of the 10000 random image pairs per database (images paired within the **same database**, 40000 total pairs).

System	Unfolding / preprocessing		Column distances		Search	
	Median	95%	Median	95%	Median	95%
Embedded	11.9	12.0	57.6	58.0	95.5	99.1
Desktop	1.1	1.2	3.5	3.5	3.1	3.3

Table 3.9.: Execution times for min-warping. Wall-clock single-core times in milliseconds. Unfolding time measured across all images. Image column distance calculation and search measured for the union of the 10000 random image pairs per database (images paired within the **same database**, 40000 total pairs). This table includes data for both the embedded Intel Atom N2600 CPU and the modern Intel Core i7-4790K-equipped desktop PC.

Method	Embedded		Desktop	
	Median	95%	Median	95%
ORB-500 two-point	274.8	430.8	24.5	41.0
ORB-500 five-point	1806.9	1829.5	156.1	158.8
ORB-1500 two-point	721.1	1162.2	62.2	109.5
ORB-1500 five-point	2576.0	2626.4	225.8	230.8
SIFT two-point	3844.4	4396.6	292.5	342.9
SIFT five-point	5627.5	6210.4	454.6	496.3
SURF two-point	4804.2	5420.3	465.0	524.2
SURF five-point	6473.3	7073.4	610.6	661.4
BRISK two-point	440.5	692.9	46.1	71.4
BRISK five-point	1909.8	2138.3	171.3	192.9
Min-warping	165.1	168.8	7.8	7.9

Table 3.10.: Total execution time for a "Homing" scenario on **full-resolution images**, in milliseconds. Includes preprocessing / feature extraction for one image, followed by relative pose estimation with another image. Calculated across the standard 10000 image pairs for each of the four databases (images paired within the **same database**, thus constant illumination). Min-warping values are identical to Table 3.11 and included for comparison only.

required. Our cleaning robot also performs these operations while visually correcting its pose estimate (Section 2.5.2). The total execution time for this scenario is listed in Table 3.10 for full-resolution images and Table 3.11 for half-resolution images. Table 3.12 contains the execution time when using Levenberg-Marquardt optimization of the final result.

Figure 3.16 shows the mean bearing error in relation to the mean execution time on the desktop PC platform. This visualizes the trade-off between estimation error and execution time for various methods. The execution time is based on the "Homing" scenario described in Section 3.3.2. To calculate this execution time, the preprocessing time for one image is added to the pose-estimation time of the image pair. This two-dimensional plot can show only one type of estimation error without becoming overcrowded. We choose the bearing error over the orientation error, since it varies more across the different methods. Figure 3.17 shows the results for strong day-night illumination changes.

Method	Embedded		Desktop	
	Median	**95%**	**Median**	**95%**
ORB-500 two-point	150.3	309.5	14.8	32.1
ORB-500 five-point	1685.7	1726.8	147.1	149.5
ORB-1500 two-point	527.6	979.5	46.7	94.3
ORB-1500 five-point	2331.3	2414.6	207.6	215.6
SIFT two-point	1126.2	1322.0	89.4	111.2
SIFT five-point	2712.2	2877.1	229.0	242.0
SURF two-point	1245.7	1456.5	127.8	149.6
SURF five-point	2778.2	2931.5	258.6	272.7
BRISK two-point	148.7	260.1	16.0	27.2
BRISK five-point	1599.9	1668.8	139.1	145.4
Min-warping	165.1	168.8	7.8	7.9

Table 3.11.: Total execution time for a "Homing" scenario for **half-resolution images**, in milliseconds. Calculated across the standard 10000 image pairs for each of the four databases (images paired within the **same database**, thus constant illumination). See Table 3.10 for details.

Method	Desktop	
	Median	**95%**
ORB-500 two-point	26.8 (24.5)	41.9 (41.0)
ORB-500 five-point	157.4 (156.1)	161.2 (158.8)
ORB-1500 two-point	69.1 (62.2)	111.4 (109.5)
ORB-1500 five-point	229.3 (225.8)	239.4 (230.8)
SIFT two-point	300.3 (292.5)	345.9 (342.9)
SIFT five-point	459.2 (454.6)	501.2 (496.3)
SURF two-point	470.2 (465.0)	527.3 (524.2)
SURF five-point	614.0 (610.6)	666.1 (661.4)
BRISK two-point	48.2 (46.1)	72.5 (71.4)
BRISK five-point	172.6 (171.3)	193.8 (192.9)

Table 3.12.: "Homing" execution time for Levenberg-Marquardt optimization with **full-resolution images** paired within the **same database** (constant illumination). Original values without optimization given in parentheses, as per Table 3.10. The impact of post-RANSAC optimization the on pose-estimation errors is listed in Table 3.4. All times measured on the desktop PC and given in milliseconds.

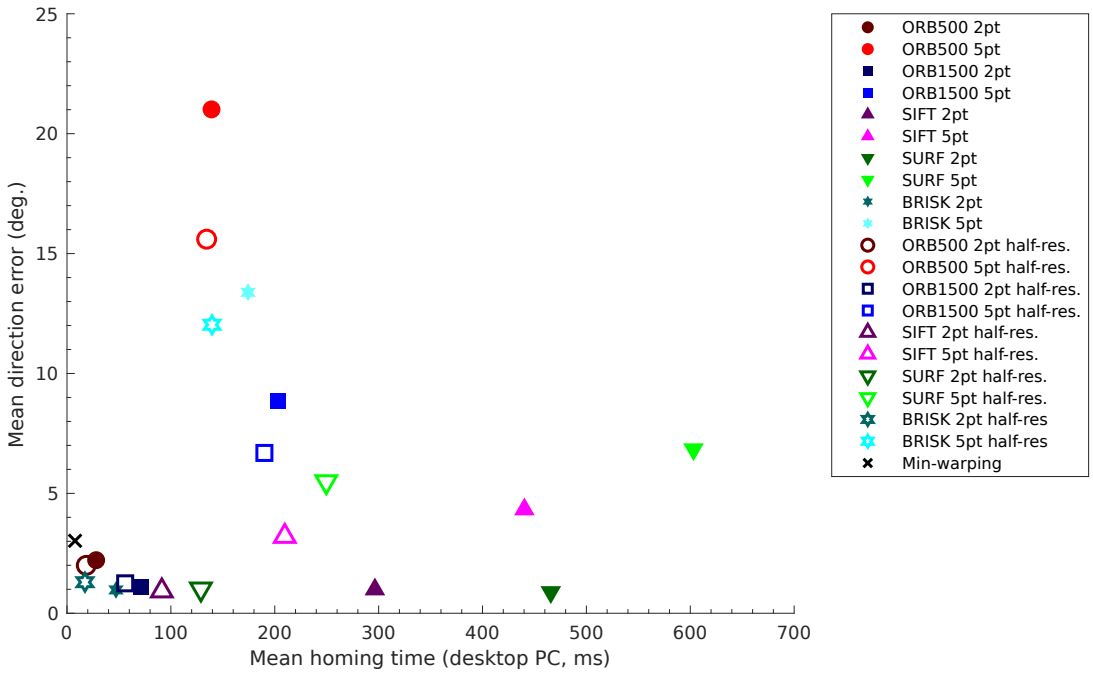


Figure 3.16.: Mean bearing estimation error against the mean desktop PC execution time, for images paired within the **same databases** (constant illumination). The mean estimation error was taken from Tables 3.1 and 3.2. The execution time includes preprocessing or feature extraction on one image, in addition to pose estimation for a single image pair. We use desaturated colors for methods that utilize the planar-motion assumption and high-saturation colors for those that do not. Filled-in markers indicate that a method uses full-resolution images.

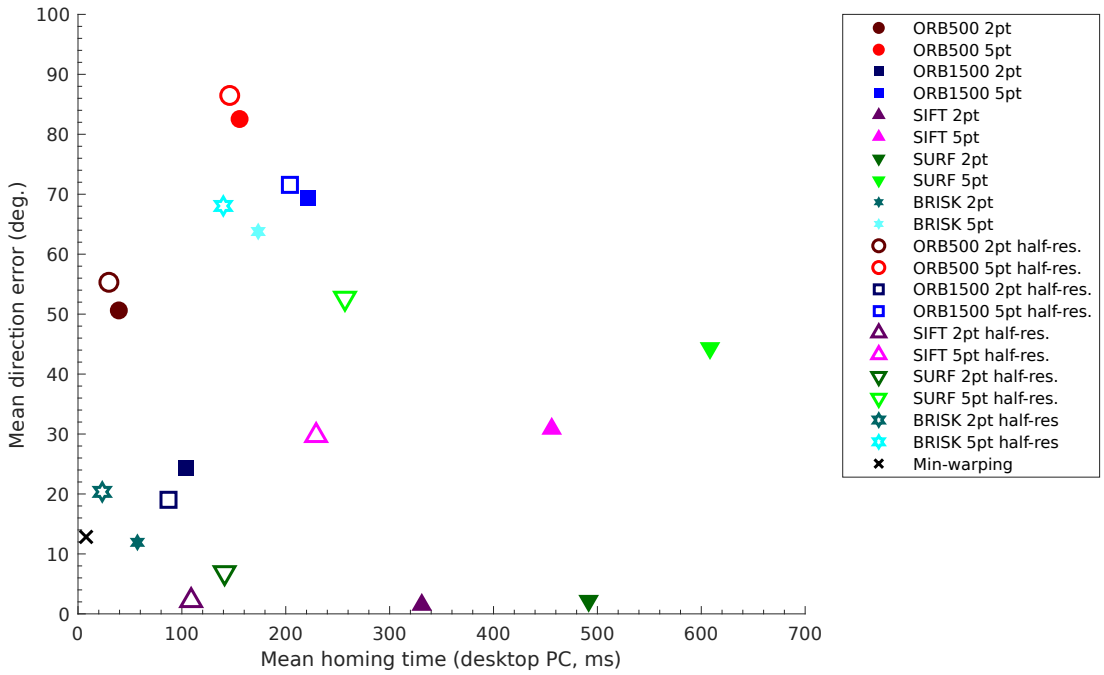


Figure 3.17.: Mean bearing estimation error against the mean desktop PC execution time, for image pairs with strong **day-night** illumination changes. The mean estimation error was taken from Tables 3.13 and 3.14. See Figure 3.16 for details.

3.3.3. Robustness

To study the effect of planar-motion constraint violations, we determined the pose-estimation error for simulated robot tilts from $0.2°$ to $3.0°$. Experiments were conducted at $0.2°$ intervals and in each of the four cardinal robot directions. We then calculated the mean error for each tilt magnitude across all tilt directions and image databases. As we do not expect the results to fundamentally differ across the feature types, we only used ORB with up to $n = 1500$ features. The results are shown in Figure 3.18.

Figures 3.19 to 3.24 show the pose-estimation errors for strong illumination changes. These figures are generated in the same manner as the figures in Section 3.3.1. However, we now combine images from different databases: Images from the daytime `LabD` and `OfficeD` databases are paired with images from the nighttime `LabN` and `OfficeN` databases, respectively (Section 3.2.1). Random sampling from these cross-database pairs gives us 2×10000 image pairs. For these experiments, the pose-estimation errors frequently grow very large. We therefore set the vertical axis to show errors up to the theoretical maximum of $180°$. For these day-night experiments, we only show the half-resolution results in Figure 3.25. Since the other plots do not differ greatly from their full-resolution counterparts, we omit them here. Half-resolution results for all candidates can however be found in Table 3.14. As before in Table 3.1, we include the mean, median, and 95th percentile of the errors for the illumination-variance experiments in Table 3.13. Similarly, Table 3.14 shows the results from the half-resolution images under illumination changes. Because incorrectly-matched features are more common under strong illumination changes, RANSAC might also take longer to identify a mismatch-free set. We include a comparison of the RANSAC execution times for constant and varying illumination conditions in Table 3.15.

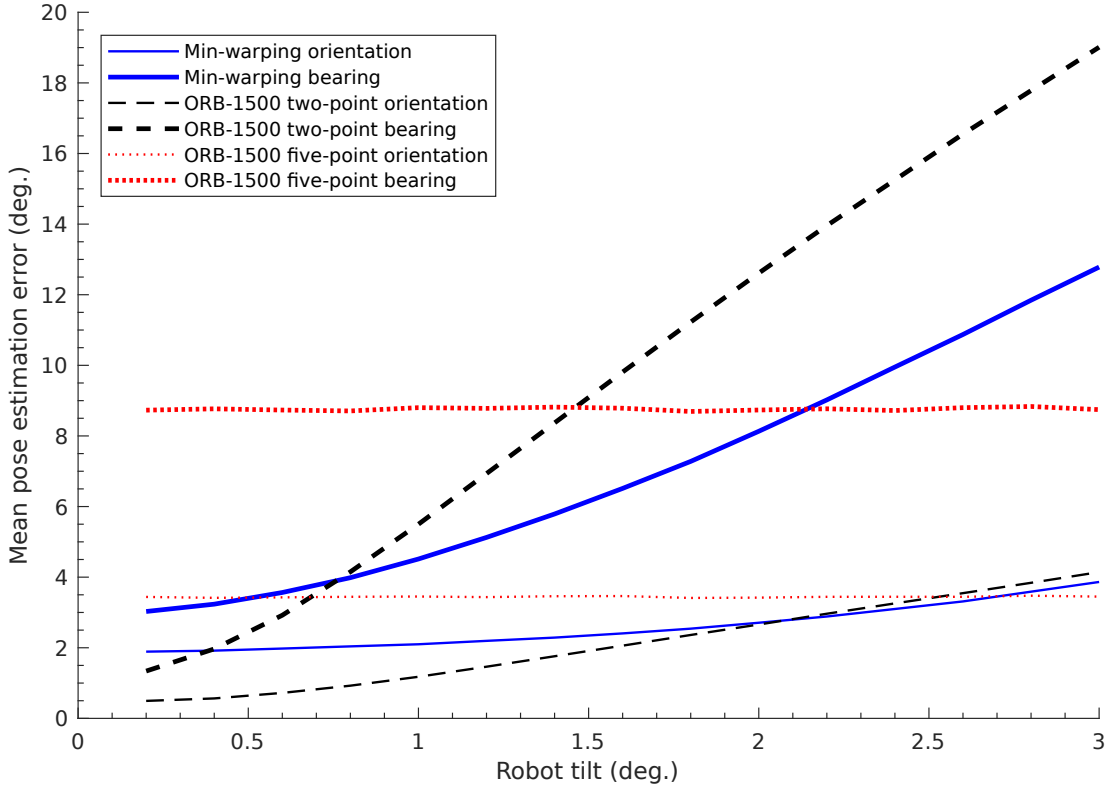
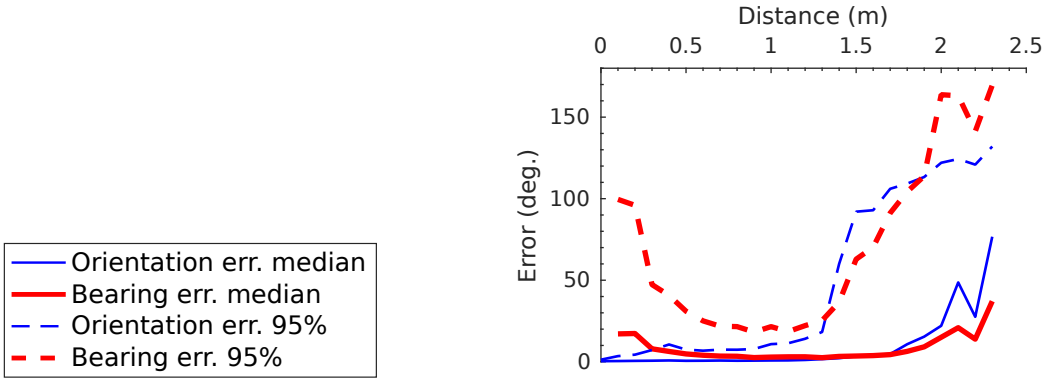


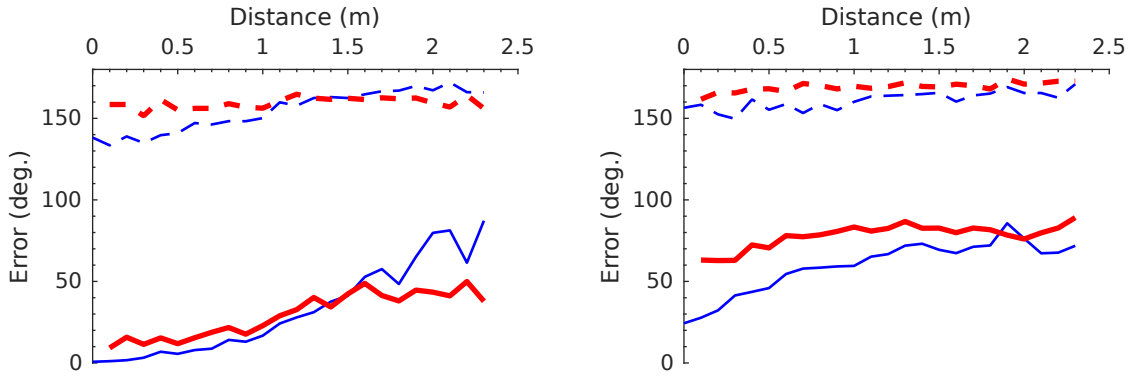
Figure 3.18.: Mean orientation and bearing error for a simulated tilting of the robot. Combined data from tilts in the four cardinal directions. **Full-resolution** images, 10000 random image pairs for each of the four image databases. Images paired within the **same database**, thus illumination is constant.



(a) Legend

(b) Min-warping

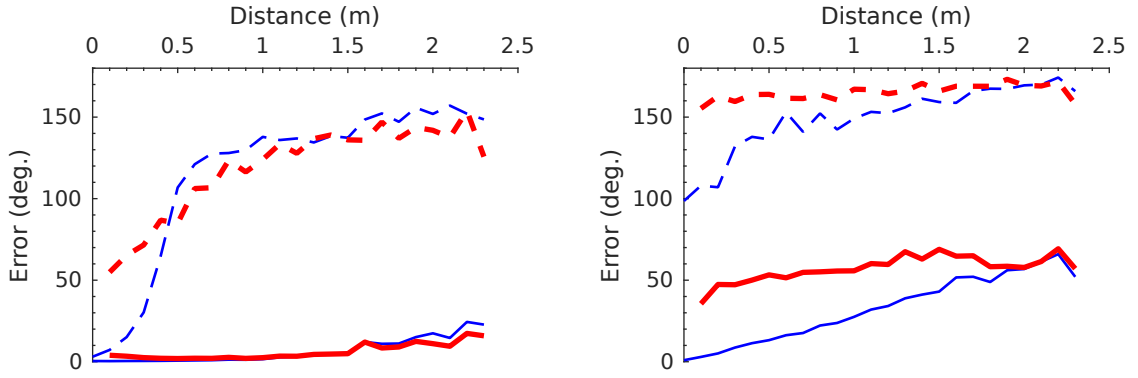
Figure 3.19.: Pose-estimation errors depending on the image-capture distance, using min-warping and images paired across **day-night databases**. Cross-database image pairs contain strong changes in illumination. Errors are given in degrees, distances in meters. Combined data from two cross-databases, 10000 random image pairs per database.



(a) ORB-500, two-point

(b) ORB-500, five-point

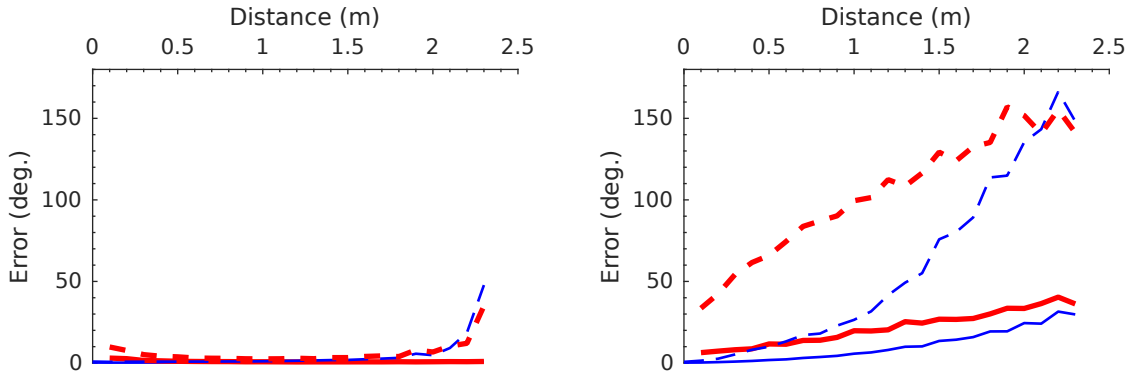
Figure 3.20.: Pose-estimation errors depending on the image-capture distance, using up to $n = 500$ **ORB features** and **full-resolution images** paired across **day-night databases**. Other settings and legend as in Figure 3.19.



(a) ORB-1500, two-point

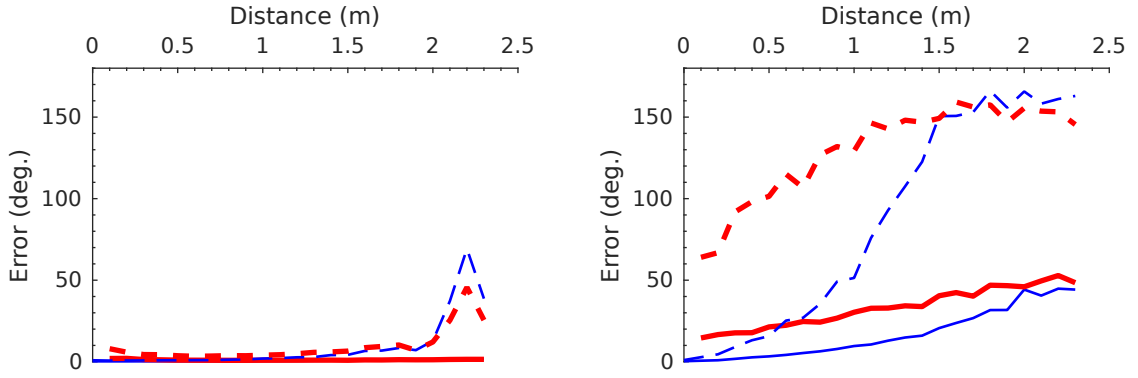
(b) ORB-1500, five-point

Figure 3.21.: Pose-estimation errors depending on the image-capture distance, using up to $n = 1500$ **ORB features** and **full-resolution images** paired across **day-night databases**. Other settings and legend as in Figure 3.19.



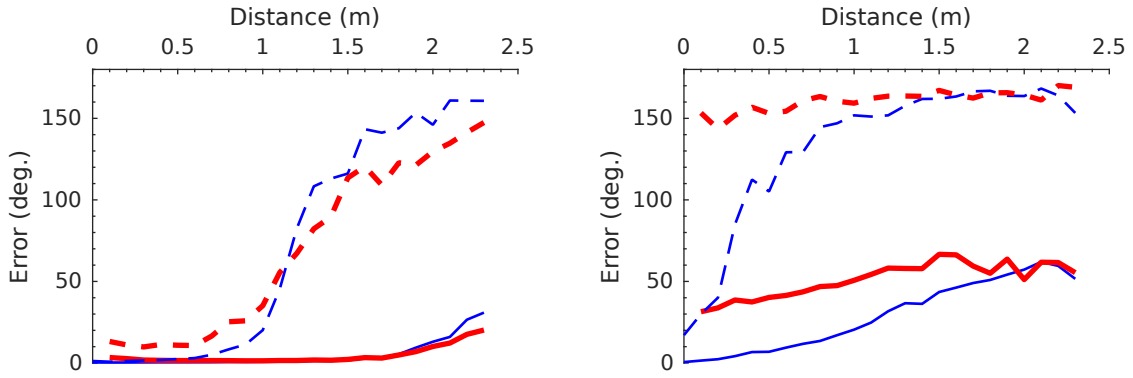
(a) SIFT, two-point **(b)** SIFT, five-point

Figure 3.22.: Pose-estimation errors depending on the image-capture distance, using **SIFT features** and **full-resolution images** paired across **day-night databases**. Other settings and legend as in Figure 3.19.



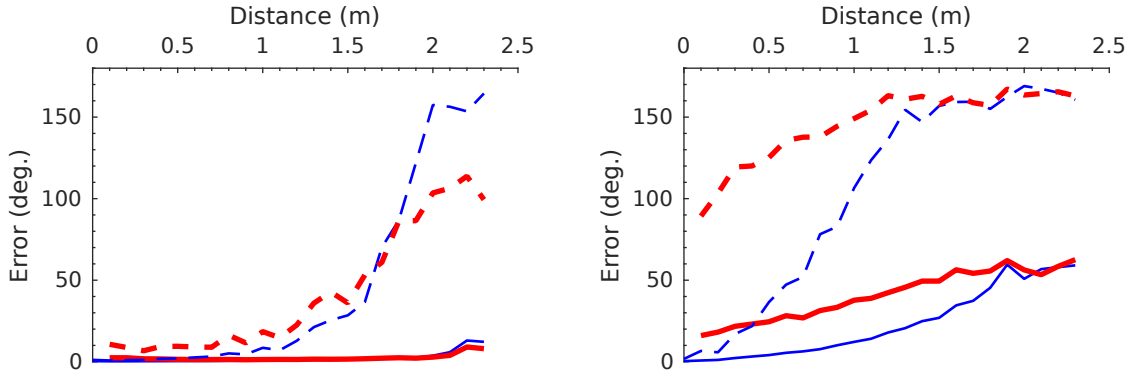
(a) SURF, two-point **(b)** SURF, five-point

Figure 3.23.: Pose-estimation errors depending on the image-capture distance, using **SURF features** and **full-resolution images** paired across **day-night databases**. Other settings and legend as in Figure 3.19.



(a) BRISK, two-point **(b)** BRISK, five-point

Figure 3.24.: Pose-estimation errors depending on the image-capture distance, using **BRISK features** and **full-resolution images** paired across **day-night databases**. Other settings and legend as in Figure 3.19.



(a) SURF, two-point **(b)** SURF, five-point

Figure 3.25.: Pose-estimation errors depending on the image-capture distance, using **SURF features** and **half-resolution images** paired across **day-night databases**. Other settings and legend as in Figure 3.19.

Method	Orientation error (°)			Bearing error (°)		
	Mean	**Median**	**95%**	**Mean**	**Median**	**95%**
ORB-500, two-point	52.68	24.14	158.35	50.62	30.02	159.56
ORB-500, five-point	68.68	61.83	161.70	82.55	78.81	169.35
ORB-1500, two-point	27.15	2.09	135.94	24.38	3.38	126.34
ORB-1500, five-point	48.78	29.04	155.98	69.39	57.53	166.06
SIFT, two-point	0.82	0.33	1.62	1.51	0.81	4.06
SIFT, five-point	13.91	5.52	55.44	30.88	18.28	108.94
SURF, two-point	1.48	0.45	3.54	2.17	1.03	5.62
SURF, five-point	23.90	9.14	115.29	44.33	29.69	140.24
BRISK two-point	12.22	0.80	98.71	11.93	1.86	73.03
BRISK five-point	43.79	21.19	153.06	63.78	50.96	161.72
Min-warping	10.52	1.23	86.91	13.02	4.02	64.85

Table 3.13.: Pose-estimation errors for **full-resolution image pairs** across **day-night databases**. Pairing images across databases causes a strong change in illumination. Mean, median, and 95th percentile of errors are given in degrees. Calculated from 10000 random image pairs for each of the two cross-databases. For the feature methods, RANSAC was limited to 1000 iterations per image pair. The min-warping results are identical to Table 3.14 and included for comparison only.

Method	Orientation error (°)			Bearing error (°)		
	Mean	**Median**	**95%**	**Mean**	**Median**	**95%**
ORB-500, two-point	48.52	17.19	160.26	55.31	34.08	163.97
ORB-500, five-point	69.96	62.35	162.50	86.48	85.41	170.24
ORB-1500, two-point	17.63	1.36	128.23	19.03	3.17	119.50
ORB-1500, five-point	46.54	22.71	155.91	71.56	61.33	166.45
SIFT, two-point	1.56	0.40	2.62	2.21	0.89	5.71
SIFT, five-point	14.79	5.32	64.07	29.74	16.81	109.62
SURF, two-point	6.04	0.75	28.63	6.88	1.61	36.24
SURF, five-point	31.46	12.46	138.87	52.65	37.77	152.83
BRISK two-point	18.82	1.69	125.15	20.35	3.95	113.68
BRISK five-point	44.76	22.39	153.84	68.04	56.27	164.99
Min-warping	10.52	1.23	86.91	13.02	4.02	64.85

Table 3.14.: Pose-estimation errors for **half-resolution image pairs** across **day-night databases**. Pairing images across databases causes a strong change in illumination. Mean, median, and 95th percentile of errors are given in degrees. Other settings are the same as for Table 3.13.

Method	Two-point		Five-point	
	Median	95%	Median	95%
ORB-500	18.8 (3.7)	20.4 (20.2)	134.4 (135.4)	136.1 (137.8)
ORB-1500	51.1 (8.0)	55.0 (55.1)	168.0 (171.5)	172.0 (176.5)
SIFT	57.3 (8.4)	65.5 (61.8)	176.8 (179.6)	182.9 (191.8)
SURF	53.4 (19.7)	60.2 (61.0)	169.3 (175.5)	176.0 (184.7)
BRISK	24.9 (12.1)	28.3 (31.2)	142.0 (140.4)	146.2 (150.7)

Table 3.15.: RANSAC wall-clock execution times for **day-night databases** with strong illumination changes. These values were measured on the desktop system and are given in milliseconds. Times for constant-illumination image pairs given in parenthesis, as in Table 3.6. All times calculated from 10000 random **full-resolution image pairs** per database.

3.4. Discussion

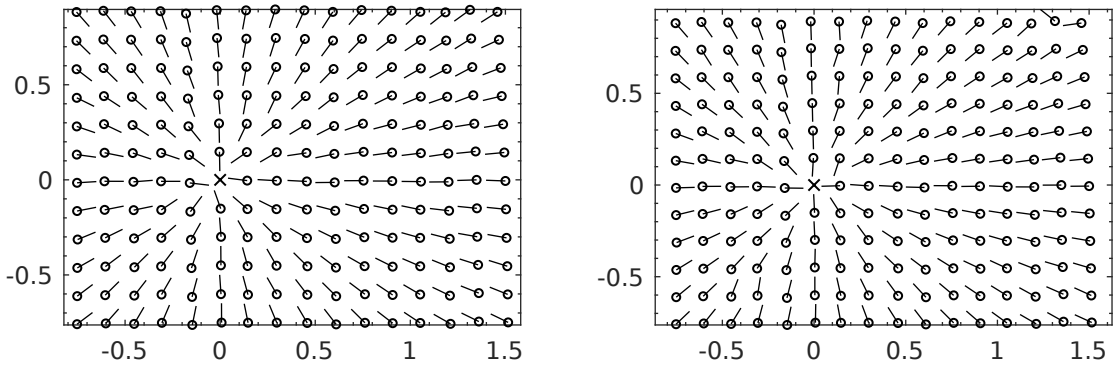
To discuss the results of our experiments, we once again consider the three aspects of quality (Section 3.4.1), speed (Section 3.4.2), and robustness (Section 3.4.3).

3.4.1. Quality

Figures 3.7 to 3.12 and Table 3.1 show that, on average, most methods can successfully solve the relative-pose estimation problem when changes in illumination are small. All methods except for the ORB-500 five-point combination show median estimation errors below 5° across most of the image-capture distances. Furthermore, even this method gives good results for intermediate ranges (Figure 3.8b). To present an example of the pose estimation performance, we include several bearing vector-field plots in Figure 3.26. These plots show the estimated bearing to a specific reference image, as calculated for all other images. They offer a quick overview of the pose-estimation quality, although only for a single reference image.

One major difference between the candidates lies in how well close- or long-range image pairs are handled. ORB-1500, SIFT or SURF features with two-point RANSAC provide good results across the entire distance range, as seen in Figures 3.9a, 3.10a and 3.11a. Some methods show a sharp increase in the severity of the outliers at short or long ranges. For ORB-500 and BRISK with two-point RANSAC, as well as for min-warping, this is shown by their 95th percentile curves. For the five-point methods, the increase in outliers occurs for shorter distances. It is also accompanied by a marked increase in the median errors. Recall that our cleaning-robot framework usually performs pose estimation at ranges below $\approx 50\,\mathrm{cm}$. Thus, such large errors at long ranges are mostly relevant for other applications.

Figure 3.6 indicates that the five-point candidates require more RANSAC iterations than their two-point counterparts. We suspect that the limit of 1000 RANSAC iterations degrades the five-point results if mismatched features are common. Figure 3.27 gives an example of five-point RANSAC with a limit of 100000 iterations. Greatly increasing the iteration limit does indeed improve the quality of the pose estimate. In return, the execution time is also greatly increased: On our modern desktop PC, the median homing time rises from $225.8\,\mathrm{ms}$ (Table 3.10) to $6.03\,\mathrm{s}$. The 95th percentile increases even further, from $230.8\,\mathrm{ms}$ to $17.34\,\mathrm{s}$. We also used these results to plot the median RANSAC inlier rate in Figure 3.28. The rate is calculated by finding the number of features that are in agreement with the RANSAC solution, and then dividing it by the total number of matched features for that image pair. The RANSAC inlier rate is only a coarse approximation



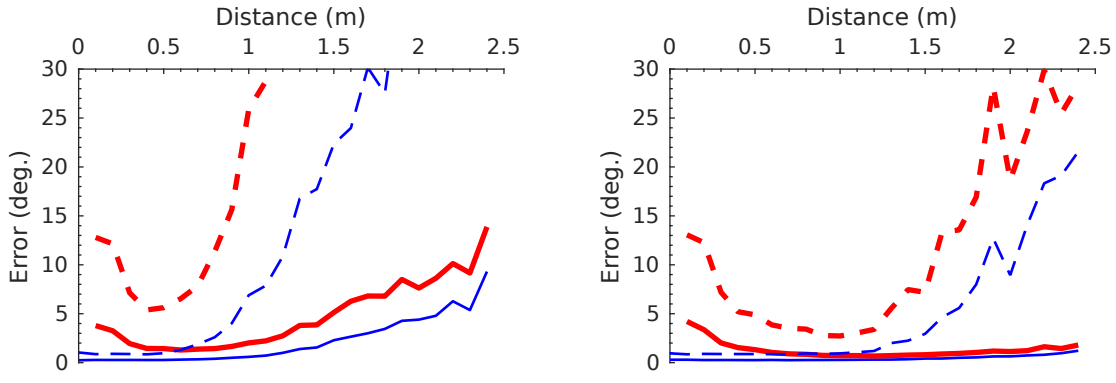
(a) Min-warping

(b) ORB-500, two-point

(c) SIFT, two-point

(d) SIFT, five-point

Figure 3.26.: Bearing vector fields for the **full-resolution** (min-warping: half-resolution) `OfficeD` database. The black lines indicate the estimated bearing from the image position (black circle) to the reference image (black cross at the origin). Ideally, all black lines would point directly from the circle at their base to the black cross. The reference image is the same one that is shown in Figure 3.2c. Axis labels are distances in meters.



(a) ORB-1500, five-point (1000)

(b) ORB-1500, five-point (100000)

Figure 3.27.: Pose-estimation errors depending on the image-capture distance for varying RANSAC iteration limits. Comparison between the five-point method with a limit of 1000 (as in Figure 3.9b) and 100000 RANSAC iterations. Up to $n = 1500$ **ORB features** and **full-resolution images** paired within the **same database**. Other settings and legend as in Figure 3.7.

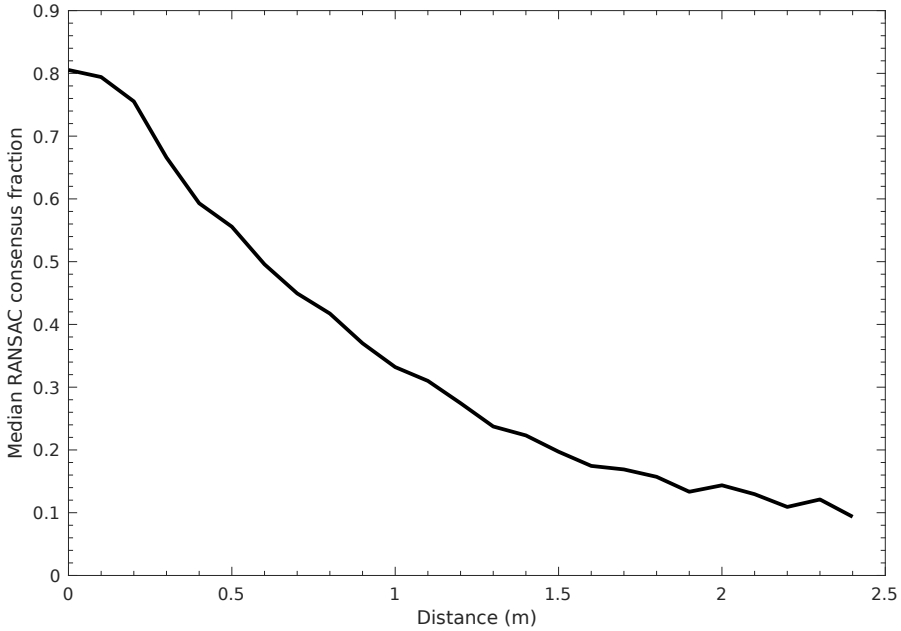


Figure 3.28.: Median fraction of inliers identified by RANSAC. Determined from ORB-1500 features and the five-point method with a 100000 RANSAC iteration limit. Here, we use the same 10 cm bins as for the error-distance plots. Based on 10000 **full-resolution image pairs** for each of the four constant-illumination databases (images paired within the **same database**).

of the true fraction of correctly matched features. However, it does show a steady decline as the distance between the images increases.

A high outlier rate at long distances would explain why some feature-based methods struggle at these ranges: An increasing outlier rate causes a rapid growth in the number of RANSAC iterations required to achieve a given success rate (Figure 3.6) [53]. However, we limit the number of iterations to prevent very long RANSAC execution times, as discussed in Section 3.2.3.2. As image-capture distances and thus outlier rates increase, it is more likely that RANSAC will fail to find an outlier-free feature set within this limit. These failures decrease the quality of the pose-estimation results. Since the number of required iterations increases more rapidly for the five-point method, this effect should be more pronounced for these candidates. This is also in agreement with our results.

For constant-illumination conditions, there is no marked improvement from using the high-resolution images. Indeed, Tables 3.1 and 3.2 and Figure 3.16 show that half-resolution images actually result in lower errors for ORB-500, ORB-1500 and BRISK features with the five-point method. Extracting local visual features from half-resolution images is also faster. For SIFT and SURF features especially, using half-resolution images greatly reduces the overall execution time (compare Tables 3.10 and 3.11).

Using Levenberg-Marquardt to improve the RANSAC result can reduce the pose-estimation error, as seen in Table 3.4. Table 3.12 shows that this step also takes little time. However, refinement does not work if RANSAC fails completely, and can even increase the error. This occurs for some of the 95th percentile entries in Table 3.4. Where computational resources are limited, moderately increasing the number of RANSAC iterations might thus be a better choice.

3.4.2. Speed

The methods we investigated vary widely in execution time, as shown in Section 3.3.2. Considering Tables 3.10 and 3.11, min-warping and two-point ORB-500 or BRISK offer by far the lowest total execution time. On our cleaning-robot prototype, only these three can reliably perform multiple pose estimations per second. On the desktop PC however, this is possible for almost all methods. SIFT, with its very low pose-estimation error, routinely runs in under $100\,\mathrm{ms}$ when used with the two-point method. As more powerful embedded CPUs become available, we expect that further candidates will become feasible for onboard use. In addition, we have not exhausted the range of optimizations that could improve the speed of these methods, as discussed in Section 3.6.

In our experiments, we found that the detection and extraction of the SURF features was unexpectedly slow. SURF was often slower than SIFT, which disagrees with the literature [7]. We repeated our experiments on additional systems and with the newest OpenCV 2.4 version (2.4.13), but observed similar results. We assume that this behavior is caused by the OpenCV implementation. Using the OpenSURF implementation of the SURF algorithm [41], we achieved execution times closer in line with the literature. However, the other implementations we evaluated did not support masking out part of the input image. This is required when working directly with the fisheye camera images. Some implementations were either written in a different programming language, such as Java, or did not provide the source code required to compile versions optimized for our test architectures. This makes them unsuitable for a direct comparison. We therefore decided to include our OpenCV-based results as they are.

Min-warping always executes the same number of steps, and its execution time is approximately constant. Note that this property is advantageous when operating under real-time constraints, such as on a domestic robot: With a near-constant execution time, we will rarely encounter delays caused by unexpectedly lengthy pose estimations. For the feature methods we use in this chapter, the execution time varies according to the number of features found and RANSAC iterations required. The difference between the median and 95th percentile times can thus be greater than 100% for a "Homing" scenario. Limiting the number of features and RANSAC iterations makes it possible to limit the execution time. Unfortunately, this will also negatively affect the quality of the pose estimates, as seen with the ORB-500 and ORB-1500 results.

3.4.3. Robustness

For our experiments, methods that use the planar-motion assumption outperform those that do not. This is unsurprising, given that our experiments fulfill this constraint. The planar-motion assumption then greatly reduces the complexity of the pose-estimation problem. However, if the planar-motion assumption is violated, this situation can be reversed: Figure 3.18 shows that this occurs even at small tilt angles. Beyond $\approx 1.5^\circ$ to 2.1°, the min-warping and ORB-1500 two-point mean bearing errors surpass that of the five-point method. The planar-motion methods are thus less robust whenever the planar-motion assumption is likely to be violated. We will discuss this problem and possible solutions in Chapter 4.

For the day-night cross-database tests, the various candidates respond very differently to strong illumination changes: The holistic min-warping method gives mixed results, as seen in Figure 3.19. Median pose-estimation errors remain low, except at long distances. However, we see pose-estimation failures — with bearing errors of more than 90° — for very short or long image-capture distances. Table 3.13 also reflects this, showing elevated mean errors for min-warping. We point to the bearing-vector plot in Figure 3.29a for an example.

When the illumination varies strongly, ORB-based methods frequently fail for almost all image-capture distances. This can be seen in Figures 3.20, 3.21 and 3.29b. Here, ORB with $n = 500$ features gives the largest errors out of all candidates. ORB-1500 performs somewhat better: The

median error for the two-point variant remains low, and is similar to min-warping (Tables 3.13 and 3.14). However, common pose-estimation failures cause high mean errors. Comparing Tables 3.13 and 3.14, we notice a somewhat lower error for the half-resolution images. With the five-point method, ORB performs badly, with median bearing errors of more than 50° in all experiments. This is to be expected: For strong illumination changes, the fraction of correctly-matched ORB features decreases. Because the five-point method requires RANSAC to find a larger set of correct matches, it is affected more severely than the two-point method; we discussed this connection in Section 3.4.1.

SIFT and SURF perform very well in our day-night experiments when used with two-point pose estimation. In this case, both methods give excellent results for all image-capture distances. The bearing vector plot in Figure 3.29c also shows these very good results. As seen in Figures 3.22 and 3.23, SIFT performs somewhat better than SURF. Outliers remain low, and we observe only a moderate increase at very long distances. When used with the five-point method, SIFT still gives acceptable results at short distances. We give an example for this in the bearing vector plot from Figure 3.29d.

According to Tables 3.13 and 3.14 and Figure 3.25, SURF performs worse on low-resolution day-night image pairs. This effect was not observed for the constant-illumination image pairs in Tables 3.1 and 3.2. The choice of image resolution can thus affect the robustness under illumination changes. It is possible that changes in the feature-method parameters would alter this behavior. Consequently, it may be worthwhile to vary these parameters during future experiments.

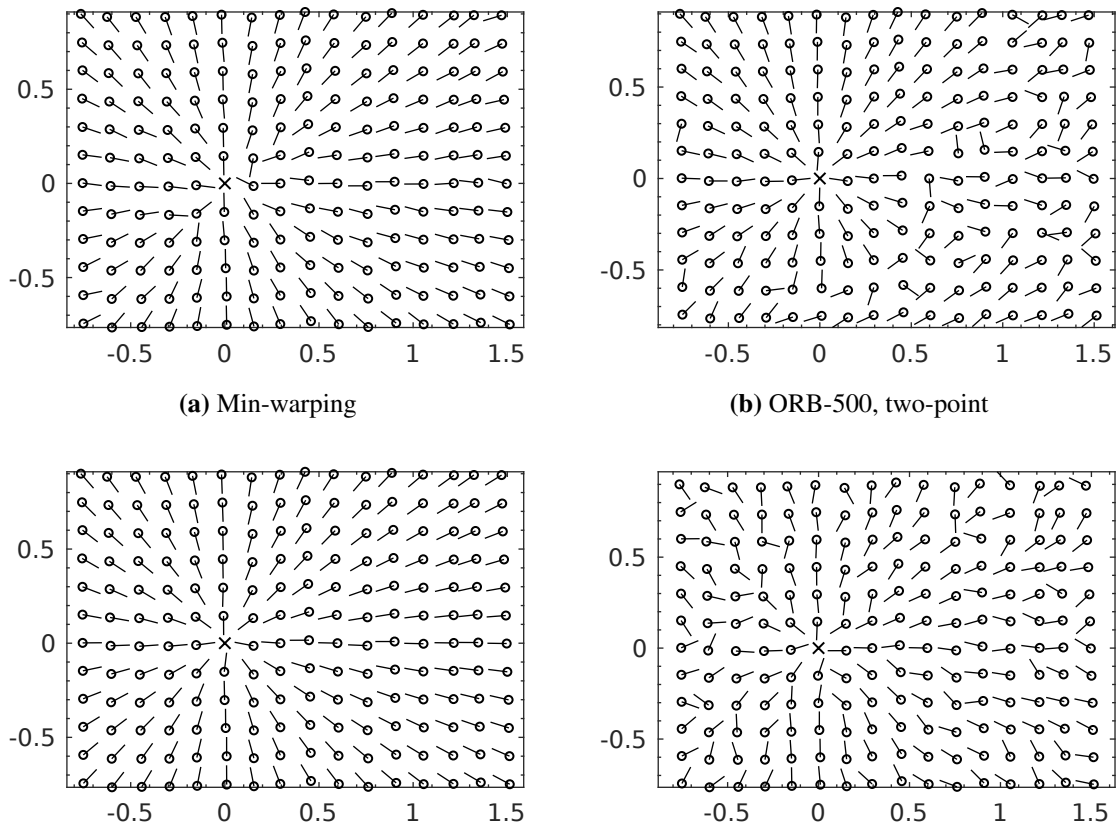
BRISK, when paired with the two-point method, also performs well under strong illumination variations. As per Table 3.13, it provides the best results among the fast, binary-feature methods when operating on full-resolution images. On half-resolution images, BRISK performs somewhat worse and is similar to the ORB-1500 variant, as shown in Table 3.14. However, according to Tables 3.10 and 3.15 BRISK still requires less time than ORB-1500. When using the five-point method in these day-night experiments, BRISK displays the same problems as the ORB variants: Pose-estimation errors are high, and pose-estimation failures occur at all ranges, as seen in Figure 3.24.

Since incorrectly-matched features occur more commonly under strong illumination changes, the RANSAC step might take longer to complete. Table 3.15 clearly shows this effect, but only for the median execution time when using the two-point method. For the 95th percentile values and the five-point method, there is no obvious effect: Here, RANSAC will often reach the limit of 1000 iterations, and this also limits the execution time. Increasing the rate of incorrect feature-matches will therefore not increase the execution time any further. As RANSAC is only one step in the pose-estimation process, the effect is rather small compared to the total execution time. This can be seen when comparing the total execution times in Figures 3.16 and 3.17.

3.5. Conclusions

All methods had at least some success with solving the visual relative-pose estimation problem in domestic environments (Table 3.1). We found that each method has specific advantages and disadvantages. No method is clearly superior under all circumstances. Selecting an appropriate method then depends on the specifics of the task.

The holistic min-warping method gives good results and is very fast, with a near-constant execution time (Figures 3.7 and 3.26a). It is also fairly robust under strong illumination changes (Figures 3.19 and 3.29a). For these reasons, min-warping remains the method of choice for our cleaning robot. However, its pose estimates are not as good as those from some of the feature methods (Table 3.1). Min-warping is also currently limited to panoramic cameras and planar motion.



(a) Min-warping

(b) ORB-500, two-point

(c) SIFT, two-point

(d) SIFT, five-point

Figure 3.29.: Bearing vector fields for **full-resolution** (min-warping: half-resolution) `Office` **day-night database** tests. The reference image was taken under daylight conditions (`OfficeD`), all other images were taken at night (`OfficeN`). Refer to Figure 3.26 for further details.

ORB is one of the fastest feature methods we studied (Table 3.10). With 500 features on half-resolution images and two-point pose estimation, its median execution time is similar to that of min-warping. ORB also gives good results under steady illumination (Figures 3.8, 3.9 and 3.26b). Unfortunately, these candidates do not perform as well for day-night illumination changes (Figures 3.20, 3.21 and 3.29b).

SIFT features give excellent results, even under strong illumination changes (Figures 3.10, 3.22, 3.26 and 3.29). Unfortunately, the implementation we used is not fast enough for real-time use on our cleaning-robot prototype (Table 3.10). Nevertheless, SIFT is the strongest of all our candidates whenever its computational requirements are not an issue. SURF offers similar performance, and should be a good contender with a faster implementation (Figures 3.11 and 3.23).

Under constant-illumination conditions, two-point BRISK gives good, high-speed pose estimates (Figure 3.16). Using half-resolution images, it is one of the fastest feature-based methods (Table 3.11). BRISK also achieves some of the lowest pose-estimation errors on full-resolution images (Table 3.1). However, outliers can occur at long distances (Figure 3.12), and the five-point results are not as good. Where illumination varies, BRISK provides better pose estimates at greater speeds than ORB (Figure 3.17). Unfortunately, it is still less robust than the slower SIFT or SURF candidates.

In our experiments, planar-motion methods are superior if the constraint is actually fulfilled (Figures 3.16 and 3.17). If the constraint is violated, the full five-point method can give better results, although it is also slower (Figure 3.18). The effect of using Levenberg-Marquardt optimization or half-resolution images for the feature-based methods also depends on the circumstances (Tables 3.4 and 3.12).

3.6. Outlook

This comparison of holistic and feature-based visual pose-estimation methods revealed several open questions: For this chapter, all images were taken from just two domestic environments. The behavior of the methods across many different environments is thus open to further study. Furthermore, outdoor environments were not considered at all. We might also include additional changes in the illumination conditions, for example due to weather. A robot reusing older images may also encounter extensive scene changes, for example from rearranged furniture.

A variety of promising local visual-feature algorithms have been presented in the literature[2, 11, 43], and could be used to extend our results. Additionally, the feature candidates in this study have several user-selectable parameters, as shown in Tables A.3a to A.3d. They might perform better if the parameters are carefully optimized. The large number of parameters makes this a very extensive task, and we therefore use the default values where possible. This is also the case for the parameters of the holistic min-warping method shown in Table A.2. We also made no attempt to fine-tune the available implementations for our specific problem. We expect that many of the algorithms can be sped up by employing SIMD instructions and multi-core parallelism. This could make additional methods feasible for real-time use on embedded systems, such as a domestic robot.

For this initial study, we relied on robust but simple algorithms for feature matching and RANSAC. However, there are several more sophisticated variants for approximate matching [108, 109] and RANSAC [148, 24, 115, 124]. Other visual pose-estimation methods are based on entirely different paradigms, such as 3D-reconstruction of features or stereo cameras [116, 117, 131, 53, 55]. These variants should be able to improve on our results, or at least offer additional trade-offs between quality and speed. Due to the large number of possible candidates and the lack of compatible implementations, we did not yet tackle this task.

All our image databases are available for download [47], including information such as the ground truth and camera calibrations. We hope that this will allow others to extend our results.

4. Visually Estimating Camera Tilts from Panoramic Images in Domestic Environments

Visual methods have applications in many mobile robotics problems, such as localization, navigation, and mapping. Some of these methods assume that the robot moves in a plane without tilting. This planar-motion assumption simplifies the problem, and may lead to improved results. For example, in Section 3.4 we found that pose-estimation techniques benefit from exploiting this assumption. However, tilting the robot violates the assumption, and may cause planar-motion methods to fail. In Section 3.3.3, we demonstrated that this problem occurs for the pose-estimation scheme used by our cleaning robot. To correct for such a tilt, we first have to measure its direction and magnitude. In this chapter, we thus estimate our robot's tilt relative to the ground plane from individual panoramic images. This estimate is based on the vanishing point of vertical elements, which commonly occur in domestic environments. Here, we propose two different methods: An image-space method exploits several approximations to detect the vanishing point in a panoramic fisheye image. The vector-consensus method uses a calibrated camera model to solve the tilt-estimation problem in 3D space. We evaluate the accuracy of these methods using images recorded by our cleaning robot. Furthermore, we measure the time required on both a desktop PC and our robot's embedded CPU. We also consider our results in the context of our cleaning-robot framework and its use of visual relative-pose estimation. Overall, we find the methods to be accurate and fast enough for real-time use on domestic robots, such as our prototype. However, the tilt-estimation error increases markedly in environments that contain relatively few vertical edges. Note that we base this chapter on an earlier publication by the author [48].

4.1. Introduction

Visual methods operating on images from a robot's onboard camera have many applications in mobile robotics. These include relative-pose estimation (Chapter 3), visual odometry [131, 53], place recognition [90], and simultaneous localization and mapping (SLAM) [55]. Some of these visual methods are based on a planar-motion assumption, where the robot travels in a flat plane without pitching or rolling. As seen in Figure 4.1, the robot thus moves with only three instead of six degrees of freedom (DOF). This simplification is used, for example, in visual relative-pose estimation [52, 140, 50, 17, 102, 15] or place recognition [58, 69]. We also utilize the planar-motion assumption throughout our cleaning-robot framework, as for example in Section 2.5.2. In Chapter 3, we compared visual relative-pose estimation methods in the context of our cleaning robot. As discussed in Section 3.4, we found that planar-motion methods can be more accurate and faster than their nonplanar counterparts.

During preliminary experiments, we discovered that uneven ground may cause our robot to pitch or roll even when in a benign indoor environment. Tilting the robot in such a manner violates the planar-motion assumption. For our pose-estimation experiments, this introduced large errors in the results, as seen in Figure 3.18. Here, even a small tilt of $\approx 2^\circ$ eliminated the quality advantage of the planar-motion methods. Larger tilts then increased these errors beyond those of the nonplanar alternatives. Such a small tilt angle can be caused even by a slight roughness of the movement

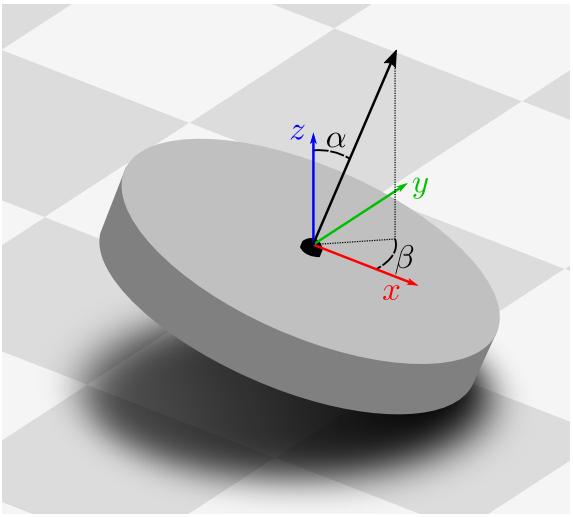


Figure 4.1.: A robot tilted relative to the ground plane (gray tiles). The robot is an abstraction of our cleaning-robot prototype. Colored arrows illustrate the coordinate system of an untilted robot. Under the planar-motion assumption, movement is restricted to the x–y plane. Furthermore, rotations may only occur around the blue z-axis, which is orthogonal to the ground plane. This reduces the degrees of freedom from six to three. Here, the robot has been tilted by an angle α in the direction β, as shown by the robot's tilted z-axis (black arrow). For reasons of legibility, this illustration shows an exaggerated α.

surface (see Section 4.2.3). Booij, Kröse, and Zivkovic [15] encountered a similar effect while testing planar-motion pose-estimation methods. The authors suspected tilts from an accelerating robot as the cause.

To tackle this problem, we estimate the tilt angle α and tilt direction β (Figure 4.1) with which an image was captured. A planar-motion method can then use this information to correct for tilts in the input image. We could also measure the tilts using an additional sensor, such as an inertial measurement unit (IMU). However, this increases the cost and complexity of the robot, more so because the sensor needs to be calibrated [88] and synchronized relative to the camera. Instead, we estimate the tilt parameters (α, β) from a single panoramic camera image by following the algorithm outlined in Figure 4.2.

Here, we utilize vertical elements — such as vertical parts of shelves, windows or doors — commonly found in domestic environments. These elements are orthogonal to the floor, and thus orthogonal to the robot's movement plane. Some of the elements will appear as edges in the camera image, which we can then detect. From this, we determine the robot's tilt by locating the vanishing point of these edges. We evaluate our results in the context of the visual pose-estimation methods discussed in Chapter 3: The tilt-estimation accuracy should allow the planar-motion methods to remain competitive with their nonplanar counterparts. Furthermore, tilt estimation should add only a small overhead to the time required for planar-motion pose estimation.

4.1.1. Related Works

Visually estimating a camera's orientation relative to the world is a prominent research problem. As in our work, vanishing points are widely used, and offer several advantages [10]: because they lie at infinity, their position is not affected by camera translation. Furthermore, the vanishing points are determined from an image's edge pixels. Thus, many non-edge pixels can be disregarded, which speeds up processing time.

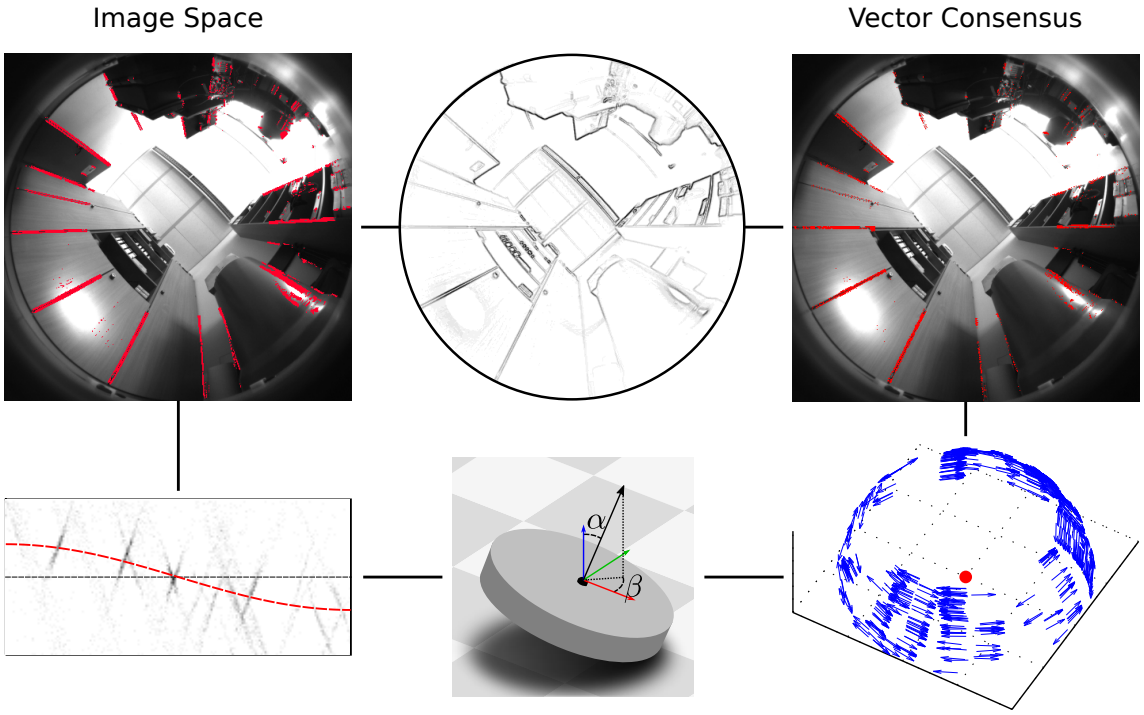


Figure 4.2.: An overview of the tilt-estimation pipeline used in this chapter. We propose two different methods, which are shown on the left and right, respectively. First, a single panoramic camera image is edge-filtered (**top**). Next, edge pixels corresponding to vertical elements are identified (**top left, top right**, shown in red). For the image-space method (Section 4.2.1), we approximate the tilt's effect as a shifting of the edge pixels within the image. We estimate the shift direction and shift magnitude by fitting a function (**bottom left**, red line) to two parameters derived from the edge pixels (black dots). These parameters are based solely on the edge pixels' positions and gradient directions in the image space. The vector-consensus method (Section 4.2.2) determines a 3D normal vector for each edge pixel. Each of these normals is orthogonal to the direction of the vanishing point. We then estimate this direction from a consensus of the normal vectors (**bottom right**, blue normals are orthogonal to tilt direction). Finally, we compute the tilt parameters (α, β) from the edge-pixel shift or the vanishing-point direction (**bottom**).

A popular class of methods assumes a *Manhattan world*, as described by Coughlan and Yuille [28]: Here, image edges are assumed to belong to world elements that are either parallel or orthogonal to each other. These three orthogonal sets of parallel elements serve as the axes of a world coordinate system. One example for such a world is an environment consisting only of parallel cuboids.

Košecká and Zhang [80] present one early work based on the Manhattan world. The authors group edge pixels from nonpanoramic images into distinct straight lines. They then determine the Manhattan-world vanishing points from these lines using expectation maximization [31]. Denis, Elder, and Estrada [32] compare several early Manhattan-world orientation-estimation methods. This work also introduces additional methods based on straight lines in nonpanoramic images. Tardif [142] uses the J-linkage algorithm [146] to identify distinct vanishing points from straight image edges. This method then selects those three vanishing points that best correspond to the Manhattan directions. All of these works require straight image edges. For this reason, they are not directly applicable to our fisheye images, which contain strong radial distortions.

Bazin et al. [10] search panoramic images for edges belonging to straight elements in the world [9]. They then perform a coarse-fine search over the space of possible camera orientations. For a Manhattan world, each hypothetical orientation predicts a distinct set of vanishing points. This method selects the orientation under which the highest number of images edges are consistent with these predicted vanishing points. Another work by Bazin et al. [10] follows the same Manhattan-world approach. As before, orientations are scored by comparing their predicted vanishing points to the image edges. To find the best orientation, a branch-and-bound algorithm [65] divides the orientation space into intervals. Those intervals that contain only inferior solutions are discarded using interval analysis. The orientation estimate is then narrowed down by subdividing the remaining intervals.

In the Manhattan world, there are six (three orthogonal, plus their antipodals) vanishing points: one vertical pair, and two additional orthogonal pairs that lie on the horizon. Schindler and Dellaert [133] generalize this to an *Atlanta world*. In contrast to the Manhattan case, this world may contain additional vanishing point pairs located on the horizon. The authors estimate these vanishing points for nonpanoramic images using expectation maximization [31]. Tretyak et al. [149] also use an Atlanta-like world. They propose a scene model that encompasses lines, their vanishing points, and the resulting horizon and zenith location. The model parameters are then jointly optimized based on the edge pixels in a nonpanoramic image. Thus, the detection of lines, vanishing points, and the horizon and zenith are performed in a single step. However, these two methods assume images without radial distortion; thus, they cannot directly be used for our panoramic fisheye images.

Antone and Teller [5] also estimate camera orientations using vanishing points, but do not assume a specific world. Vanishing-point candidates are found by a Hough transform (survey: [73]), and refined through expectation maximization [31]. The authors then identify a set of global vanishing points that appear across an entire collection of images. Camera orientations relative to these global vanishing points are then jointly estimated for all images. Lee and Yoon [83] also do not require a Manhattan or Atlanta world. Using an extended Kalman filter, they jointly estimate the vanishing points and camera orientations over a sequence. These two methods place no prior restrictions on the locations of the vanishing points. Consequently, the alignment between the vanishing points and the robot's movement plane is not known. To use them for tilt correction, this alignment must first be determined. Thus, these methods are not directly suitable to estimate the tilt from a single image.

The works discussed above estimate camera orientations relative to global vanishing points. Many other works determine the relative camera poses between two images [131, 53]: One popular approach works by matching local visual features, such as those from the scale-invariant feature transform (SIFT) [89], between two images. The relative pose can then estimated from epipolar

geometry [138]. However, these relative poses provide no information about the ground plane. Subsequently, they cannot be used to estimate tilts for planar-motion methods.

Some of the methods discussed here may also solve our tilt-estimation problem. For example, Bazin et al. [10] determine the camera orientation relative to a Manhattan world from a single panoramic image. However, we seek a solution that is optimized specifically towards tilt-estimation for planar-motion methods. Within this problem, we need only the tilt angle and direction, and can ignore the robot's yaw. We can estimate these angles from vertical elements alone, without requiring a full Manhattan or Atlanta world. Thus, we do not have to restrict our cleaning robot to environments that correspond to one of these two world types. In our application, the tilt angle α is also bound to be small, which allows further simplifications. By exploiting these properties, we can achieve good and fast tilt estimates using considerably simpler methods.

4.1.2. Our Contributions

In this chapter, we introduce two different visual methods to determine a robot's tilt relative to a ground plane. We evaluate and discuss these methods within the context of our cleaning robot: In Section 2.5.2, we examined how our robot performs localization by estimating the relative camera pose between two images. For our prototype, we employ the planar-motion min-warping algorithm for this task (Section 3.2.2, [102]). As discussed in Section 3.4, this provides accurate relative-pose estimates while requiring very few computational resources.

However, the robot may tilt when driving over uneven ground, such as carpets or door thresholds. Such a violation of the planar-motion assumption degrades the pose-estimation quality, as seen in Figure 3.18. For this chapter, we use this problem as a guide for designing experiments (Sections 4.2.3 and 4.2.4) and evaluating results (Section 4.4). As per Section 4.2.3, we also use our cleaning-robot prototype to record a plausible database of tilted images. However, the methods presented here are not limited to our specific cleaning robot; they may be used with other robots or planar-motion methods.

Both methods in this work operate on panoramic fisheye images, as captured by our robot's onboard camera (Figure 4.2). Recall that these images show the hemisphere above the robot, but exclude everything below the horizon. Thus, the camera cannot see the ground plane, relative to which the tilt should be measured. Instead, we use vertical elements in the environment, which are orthogonal to the movement plane. Examples include room corners, door- and window frames, as well as the edges of furniture such as shelves. Some of these parallel elements appear as visually distinct edges in the robot's camera images.

We now estimate the tilt parameters from the vanishing point of these edges: First, we apply an edge filter to the camera image and extract pixels with a strong edge response. Second, we identify the set of edge pixels belonging to vertical elements in the world, while rejecting those from non-vertical elements. Here, we assume that the tilt angle is small and that vertical elements far outnumber the near-vertical ones. Third, we determine the tilt angle α and direction β from the remaining edge pixels. Step one is identical for both methods, but steps two and three differ. The *image-space* method uses several approximations to simplify vanishing-point detection in the fisheye images. We then apply a correction factor to reduce the tilt-estimation errors introduced by these approximations. Operating directly on the panoramic fisheye image, this method is fast and simple to implement. In contrast, the *vector-consensus* method solves the tilt-estimation problem in 3D world coordinates. This method makes fewer approximations, but requires a calibrated camera model. It is also more similar to the existing vanishing-point methods discussed in Section 4.1.1.

We test the two methods on images recorded by the onboard camera of our cleaning-robot prototype. Here, we placed our prototype at 43 locations spread across six different environments. For each location, we recorded images for the untilted and six different tilted configurations. The tilts estimated from these images were then compared to the ground truth, giving a tilt-estimation

error. We also measure the execution time required for tilt estimation on both a modern desktop PC and our robot's onboard computer.

The rest of this work is structured as follows: Sections 4.2.1 and 4.2.2 describe the two methods used in this work. We introduce the image database in Section 4.2.3, and the experiments in Section 4.2.4. Section 4.3 contains the results of these experiments, which we discuss in Section 4.4. Finally, we summarize our results and give an outlook to possible future developments in Section 4.5.

4.2. Materials and Methods

Our specific goal is to estimate the robot's tilt angle α and tilt direction β, as shown in Figure 4.1. β is given relative to the robot's heading, with $\beta = 0°$ and $\beta = 90°$ describing a forward and leftward tilt, respectively. α is measured relative to the robot's untilted pose on a planar surface; thus, we have $\alpha = 0°$ for an untilted robot. We choose our world coordinate system so that $\vec{n} = (0, 0, 1)^T$ is the surface normal of the movement plane. If $\vec{n}_R = R_{\alpha,\beta}\vec{n} = (n_{x,R}, n_{y,R}, n_{z,R})^T$ is the normal vector in the coordinate system of a tilted robot, we then find that

$$\begin{aligned} \alpha &= \arccos(n_{z,R}), \\ \beta &= \mathrm{atan2}(-n_{y,R}, -n_{x,R}). \end{aligned} \tag{4.1}$$

Here, $R_{\alpha,\beta}$ is the rotation matrix for the tilt (α, β), and $\mathrm{atan2}$ is the quadrant-aware arctangent function. We can therefore determine (α, β) by first determining $\vec{n}_R$. Both methods perform this step using visually distinct environment elements which are parallel to $\vec{n}$.

4.2.1. Image-Space Method

This method is based on the apparent location $\vec{p}_n$ of the vertical elements' vanishing point. $\vec{p}_n$ is the point in the camera image where the vertical elements would appear to meet, if they were extended to infinity. We estimate $\vec{p}_n$ using several problem-specific approximations, which simplify and speed up the method. We then determine the tilt parameters (α, β) using the location of $\vec{p}_n$.

4.2.1.1. Preliminary Calculations

We first determine the vanishing point for an untilted robot, which we call $\vec{p}_c$. For convenience, we find $\vec{p}_c$ using an existing camera calibration based on [132]. However, this point could also be determined separately, without requiring a fully calibrated camera. The calibrated camera model provides a projection function P. P maps the camera-relative bearing vector $\vec{v}$ to the image point $\vec{p} = P(\vec{v})$. Multiplying with the extrinsic rotation matrix R gives the bearing vector $\vec{v}_R = R\vec{v}$ in robot coordinates. By definition, the surface normal is $\vec{n}_{R,\alpha=0} = \vec{n}$ for an untilted robot. $\vec{n}_{R,\alpha=0}$ is parallel to the environment's vertical elements, and consequently shares their vanishing point. Transforming $\vec{n}_{R,\alpha=0}$ into camera coordinates and projecting it gives us

$$\vec{p}_c = P(R^{-1}\vec{n}_{R,\alpha=0}). \tag{4.2}$$

If the robot is tilted, the apparent vanishing point $\vec{p}_n$ will be shifted from $\vec{p}_c$. We will determine the tilt parameters (α, β) from this shift. Figure 4.3 illustrates $\vec{p}_n$ and $\vec{p}_c$ for an untilted example image.

Note that it is irrelevant whether or not the camera image actually shows the environment at or around $\vec{p}_c$. Thus, this method would still work for robots where this part of the field of view is obscured. For a robot with an upward-facing fisheye camera, $\vec{p}_c$ lies close to the geometric center of the image. However, this may not apply for an upward-facing camera that captures panoramic

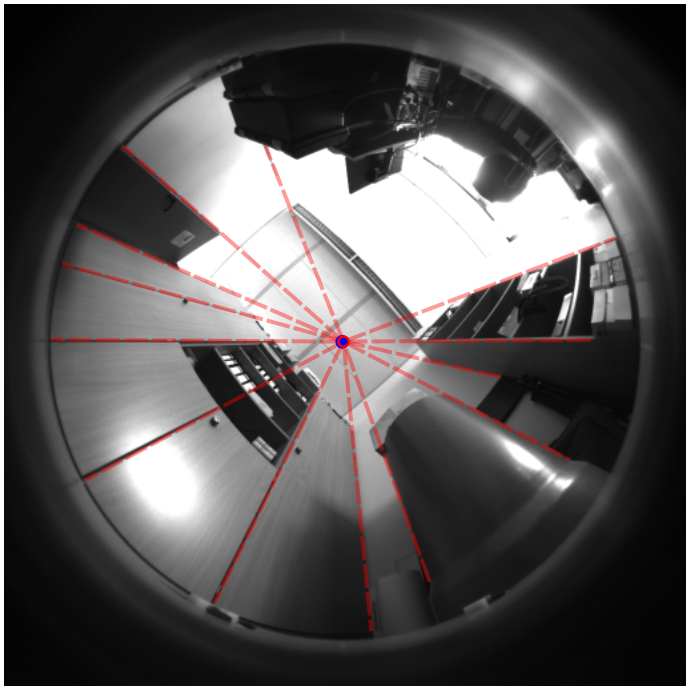


Figure 4.3.: An illustration of the vertical element's vanishing point. This untilted image from our database shows a typical office environment. A blue dot represents the predicted vanishing point for an untilted robot $\vec{p}_c$. The red, dashed lines represent vertical elements, which we manually extended to their vanishing point $\vec{p}_n$ (red ring, overlapping the blue dot). As expected, the predicted and actual vanishing point of the vertical elements are nearly identical.

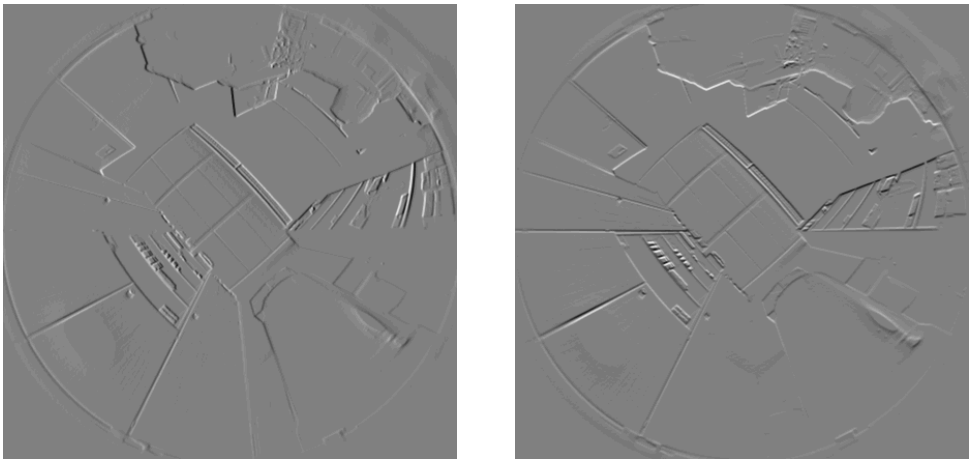
images using a mirror. Here, the image center corresponds to a direction below instead of above the robot. In this case, we can simply use the antipodal vanishing point of the vertical elements. This vanishing point lies below the robot, and thus we calculate $\vec{p}_c$ from $\vec{n}_{R,\alpha=0} = -\vec{n}$ instead. Subsequent steps in the method then use this antipodal vanishing point.

4.2.1.2. Edge Pixel Extraction

As a first step towards tilt estimation, we detect edges in the camera image. We apply the Scharr operator to compute the horizontal and vertical edge gradients g_x and g_y for each pixel. This edge detector is somewhat similar to the popular Sobel operator, but was optimized to be invariant to edge orientation [74, 154]. Since vertical elements can appear at any orientation within the image, this is a highly desirable property. In this work, we use the implementation provided by the OpenCV library [21]. Figure 4.4 contains an example of the resulting edge gradients.

We ignore pixels below the camera's horizon, since they mostly show the robot's chassis. We also reject pixels with a bearing of more than $45°$ above the camera horizon. Such pixels mainly show the ceiling, which contains horizontal elements that may be mistaken for vertical ones. To speed up computations, we only apply the Scharr operator to a bounding box around the remaining pixels. Finally, we discard pixels with a low gradient intensity $g_x^2 + g_y^2 < I_{\text{min}}^2$. Such pixels are of limited use to us, since camera noise may strongly disturb their edge gradients. Eliminating these pixels also speeds up subsequent processing steps.

In this study, we do not try to identify straight lines of edge pixels, and instead consider each pixel individually. This simplifies and speeds up our method, and lets us extract information even from very short edges. In contrast, combining connected pixels into long edges (as in [9]) may reduce the number of incorrect edge pixels, and may make the edge-gradient estimate more accurate



(a) Horizontal **(b)** Vertical

Figure 4.4.: The result of the Scharr operator applied to the example image from Figure 4.3. The operator was only applied to a bounding box containing the pixels above the camera horizon. Black and white correspond to a strong dark-bright or bright-dark edge, respectively; gray indicates a weak edge response. We artificially increased the contrast of these images to make the gradients more noticeable.

[32]. The trade-off between using individual edge pixels or long edges thus poses a possible subject for future experiments.

4.2.1.3. Edge Pixel Processing

Next, we estimate the tilt from the edge pixels identified in Section 4.2.1.2. For a camera with linear projection, a linear element in the environment would appear as a straight edge in the image. We can derive the edge direction for each pixel from the gradient $\vec{g} = (g_x, g_y)^T$. Extending a line from each edge pixel along its edge direction result in a vanishing point, similar to Figure 4.3.

However, the projection of a panoramic camera is typically not linear. Here, we assume that the robot's camera fits the panoramic, nonlinear model proposed by Scaramuzza, Martinelli, and Siegwart [132]. Due to radial distortion, straight environment elements commonly appear as curves in the camera image. One example is given by the horizontal elements in Figure 4.3. Nevertheless, we can use two simplifying approximations for this tilt-estimation problem: First, we assume that vertical elements appear as straight edges in the panoramic image. This is approximately true for small tilt angles, as for example in Figure 4.5. Second, since α is small, we assume that a tilt appears as a shift in the image. The edges and their vanishing point thus appear to move by a uniform amount (Figure 4.6c). In reality, tilting changes the orientation of these edges in the image. Due to the radial distortions of our fisheye lens, the vertical elements may also appear as curves (Figure 4.6b). Under our approximations, we ignore these effects and determine only an approximated vanishing point $\vec{p}_a$ (Figure 4.5) instead of the actual vanishing point $\vec{p}_n$. This greatly simplifies our method, but also introduces tilt-estimation errors. We therefore apply a correction factor after estimating the tilt from the shift between $\vec{p}_a$ and $\vec{p}_c$.

Panoramic Projections of Vertical Elements

We now use the camera model to justify our previous assumption that vertical elements are projected linearly. According to the projection P of the camera model [132], a bearing vector $\vec{v} = (x, y, z)^T$

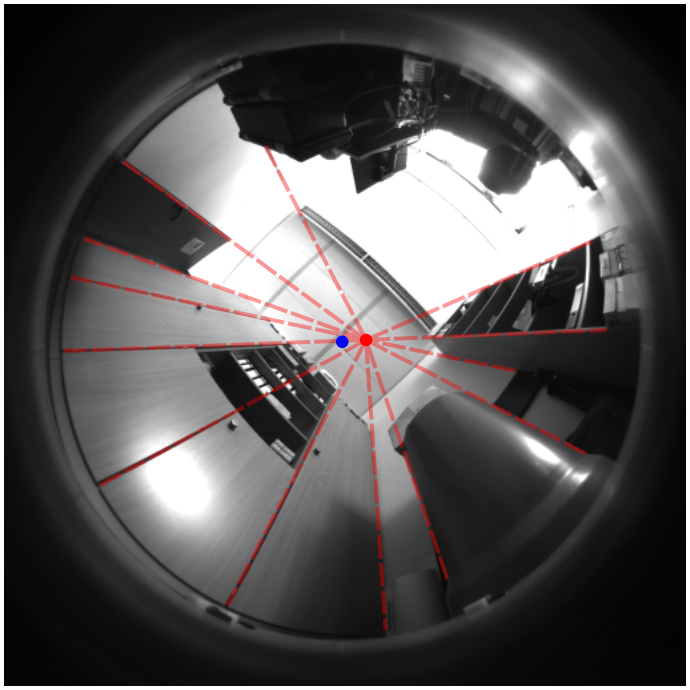
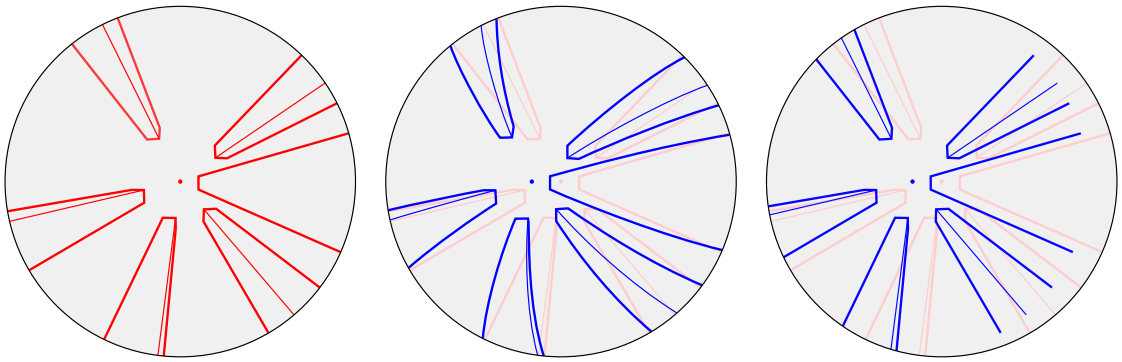
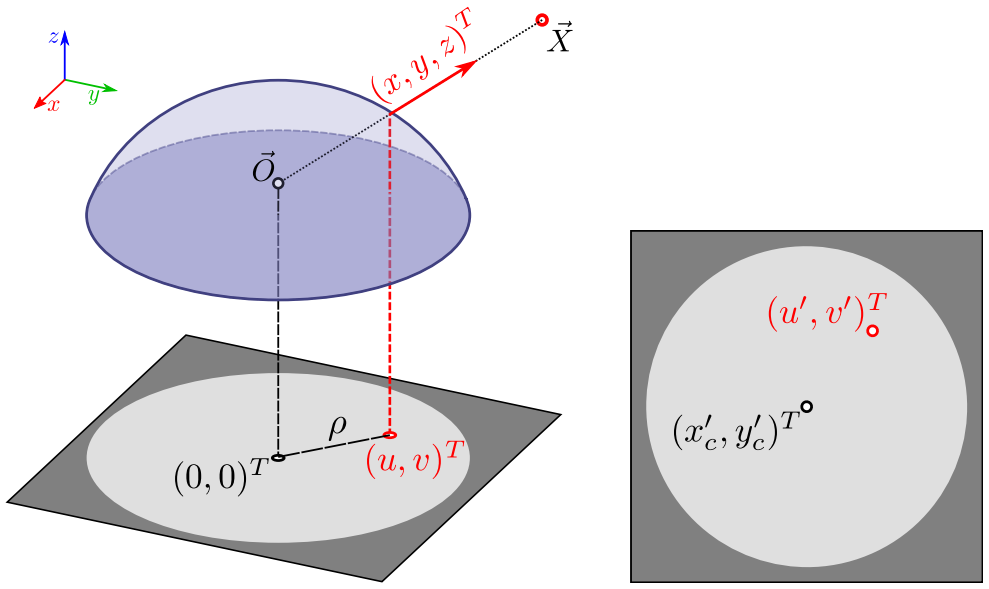


Figure 4.5.: The same as Figure 4.3, but for a forward-tilted robot ($\alpha_T = 4.15°, \beta_T = 0°$). Due to the tilt, the approximated vanishing point $\vec{p}_a$ (red dot) is shifted from $\vec{p}_c$ (blue dot). Close inspection reveals that the tilt caused a slight curvature in the vertical lines. Thus, the dashed red lines and the resulting $\vec{p}_a$ are merely an approximation of the true edges and vanishing point, respectively.



(a) Untilted ($\alpha = \beta = 0°$) **(b)** Tilted ($\alpha = 15°, \beta = 180°$) **(c)** Shifted ($\alpha = 15°, \beta = 180°$)

Figure 4.6.: The effect of tilts on image edges. This illustration shows the outlines of several cuboids, as seen by a robot's panoramic fisheye camera. In (**a**), the robot is not tilted. The vertical elements appear as straight lines oriented towards a vanishing point (red dot). (**b**) shows the effect of an exaggerated tilt, where vertical elements appear as curves. These curves no longer point straight towards the expected vanishing point (blue dot). This point has been shifted from its untilted position (red dot). This figure also contains the untilted outlines in a light shade of red. In the image-space method, we assume that small tilts do not cause the distortions in (**b**). Instead, we model the effect of such tilts as a mere shift in the image. This approximation is shown in (**c**), where the vanishing point and outlines are shifted without distortion.



(a) Projection to the idealized sensor plane **(b)** Projection to the image plane

Figure 4.7.: The camera model used in this work, which was first introduced by Scaramuzza, Martinelli, and Siegwart [132]. In (**a**), the point $\vec{X}$ (red circle) lies at a bearing of $(x, y, z)^T$ (red arrow) relative to the camera center $\vec{O}$ (black circle). A fisheye lens (light blue shape) projects $(x, y, z)^T$ onto the point $(u, v)^T$ (red circle) in an idealized sensor plane (gray square). This nonlinear projection is described by Equation (4.3) and the camera parameters a_k from Equation (4.5). The distance between $(u, v)^T$ and the image center at $(0, 0)^T$ is specified by ρ. Applying Equation (4.4) to $(u, v)^T$ gives us the corresponding pixel coordinates $(u', v')^T$ in the actual digital image, as shown in (**b**). Here, the center of this image has the pixel coordinates $(x'_c, y'_c)^T$.

and corresponding image point $\vec{p} = (u', v')^T = P(\vec{v})$ are related by

$$\vec{v} = (x, y, z)^T = (u, v, f(\rho))^T, \tag{4.3}$$

$$\begin{pmatrix} u' \\ v' \end{pmatrix} = \begin{pmatrix} c & d \\ e & 1 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} + \begin{pmatrix} x'_c \\ y'_c \end{pmatrix}. \tag{4.4}$$

Here, c, d, e, x'_c and y'_c are camera parameters, while $\rho = \sqrt{u^2 + v^2}$. While $(u', v')^T$ refers to the actual pixel coordinates in the image, $(u, v)^T$ is the corresponding point in an idealized sensor plane. This idealized sensor plane is orthogonal to the optical axis, which corresponds to the camera's z-axis. Figure 4.7 illustrates this camera model.

The $n + 1$ coefficients a_k of the polynomial function

$$f(\rho) = \sum_{k=0}^{n} a_k \rho^k \tag{4.5}$$

are also camera parameters determined during calibration; in our experiments, we use $n = 5$. We now consider a straight line in space that is parallel to the camera's optical axis. In camera coordinates, such a line consists of all points $\vec{X}$, which satisfy

$$\vec{X} = (X, Y, Z)^T = (X_0, Y_0, Z_0 + \mu)^T, \tag{4.6}$$

where $(X_0, Y_0, Z_0)^T$ is a point on the line and $\mu \in \mathbb{R}$. The bearing vector from the projection center to $\vec{X}$ is simply $\vec{v} = \lambda^{-1}\vec{X}$; here, $\lambda^{-1} \geq 0$ is the unknown distance factor. From Equation (4.3), we

see that the sensor-plane projection of this line must fulfill

$$(u_\mu, v_\mu, f(\rho_\mu))^T = \lambda^{-1}(X, Y, Z)^T = \lambda^{-1}(X_0, Y_0, Z_0 + \mu)^T. \tag{4.7}$$

We now express this projection in polar coordinates

$$(u_\mu, v_\mu)^T = \rho_\mu(\cos(\gamma), \sin(\gamma))^T \tag{4.8}$$

and, from Equation (4.7), find that

$$\gamma = \operatorname{atan2}(v_\mu, u_\mu) = \operatorname{atan2}(\lambda^{-1}Y_0, \lambda^{-1}X_0) = \operatorname{atan2}(Y_0, X_0) \tag{4.9}$$

is constant. Since the orientation γ for the projection of $\vec{X}$ is constant, the vertical element $\vec{X}$ appears as a straight line in the sensor plane. Because Equation (4.4) is linear, this also corresponds to a line in the final image.

From Equation (4.8), we note that the sensor-plane projections of all lines $\vec{X}$ intersect at the origin point $(u_\mu, v_\mu)^T = \vec{0}$ for $\rho_\mu = 0$. At this origin point, we have $u_\mu = v_\mu = 0$, and thus the distance factor from Equation (4.7) must be $\lambda^{-1} = 0$ (except for lines with $X_0 = Y_0 = 0$). We now wish to find the specific point along a line $\vec{X}$ that is projected to the sensor-plane origin. At this origin $\rho_\mu = 0$, and we thus evaluate

$$\lim_{\rho_\mu \to 0} Z = \lim_{\rho_\mu \to 0} \frac{\overbrace{f(\rho_\mu)}^{\to a_0}}{\underbrace{\lambda^{-1}}_{\to 0^+}} = \infty \operatorname{sgn}(a_0). \tag{4.10}$$

We can assume that $a_0 \neq 0$, since Equation (4.3) would otherwise give a malformed bearing of $\vec{v} = \vec{0}$ for the origin $(u, v)^T = \vec{0}$.

We have now shown that lines parallel to the optical axis are projected linearly. Additionally, we found that these projections all meet at the vanishing point $(u, v)^T = \vec{0}$ as Z approaches infinity. As per Equation (4.4), this point corresponds to $(u', v')^T = (x'_c, y'_c)^T$ in image coordinates. However, even for an untilted robot, the optical axis is usually not perfectly parallel to the vertical elements. Here, we assume that this misalignment is small for an upward-facing panoramic camera ($\approx 0.16^\circ$ in our cleaning-robot prototype, as calculated from the extrinsic calibration matrix R). We therefore choose to ignore this effect.

Approximating Tilts through Image Shifts

Next, we determine how tilting the robot affects the vanishing point of the vertical elements. A tilt is a rotation around an axis $\vec{w} = (w_x, w_y, 0)^T$ parallel to the movement plane, with $\|\vec{w}\| = 1$. We derive the rotation matrix S' for such a tilt from by Rodrigues' formula [111], with

$$S' = \begin{pmatrix} \cos(\alpha) + w_x^2(1 - \cos(\alpha)) & w_x w_y(1 - \cos(\alpha)) & w_y \sin(\alpha) \\ w_y w_x(1 - \cos(\alpha)) & \cos(\alpha) + w_y^2(1 - \cos(\alpha)) & -w_x \sin(\alpha) \\ -w_y \sin(\alpha) & w_x \sin(\alpha) & \cos(\alpha) \end{pmatrix}. \tag{4.11}$$

For a small tilt angle α, we approximate S' using $\cos(\alpha) \approx 1$ and $\sin(\alpha) \approx \alpha$, which gives us

$$S = \begin{pmatrix} 1 & 0 & w_y\alpha \\ 0 & 1 & -w_x\alpha \\ -w_y\alpha & w_x\alpha & 1 \end{pmatrix}. \tag{4.12}$$

As discussed above, we assume that the optical axis is parallel to the vertical elements for an untilted robot. Thus, these elements are parallel to $\vec{n}_C = (0, 0, a_0)^T$ in camera coordinates. After

rotating the camera around $\vec{w}$ by a small angle α, this is $\vec{n}_{C,t} \approx S^{-1}\vec{n}_C = a_0(-w_y\alpha, w_x\alpha, 1)^T$. According to Equation (4.3), the sensor-plane projection (u_t, v_t) of $\vec{n}_{C,t}$ obeys

$$(u_t, v_t, f(\rho_t))^T = a_0(-w_y\alpha, w_x\alpha, 1)^T. \tag{4.13}$$

Here, (u_t, v_t) is sensor-plane location of the vanishing point for the tilted camera.

Since α is small, this vanishing point shifts only a small distance from its untilted location of $(u, v)^T = \vec{0}$. In that case, $\rho_t = \sqrt{u_t^2 + v_t^2} \approx 0$, and we can approximate Equation (4.5) as $f(\rho_t) \approx f(0) = a_0$. This approximation is favorable because $a_1 = 0$ for cameras with fisheye lenses, or parabolic or hyperbolic mirrors [132]. A tilt around the axis $\vec{w}$ therefore shifts the vanishing point in proportion to the tilt angle α:

$$(u_t, v_t)^T = a_0\alpha(-w_y, w_x)^T, \tag{4.14}$$

$$\begin{pmatrix} u_t' \\ v_t' \end{pmatrix} = a_0\alpha \begin{pmatrix} -cw_y + dw_x \\ -ew_y + w_x \end{pmatrix} + \begin{pmatrix} x_c' \\ y_c' \end{pmatrix}. \tag{4.15}$$

Here, $(u_t', v_t')^T$ is the vanishing-point location in the actual image according to Equation (4.4). The direction of this shift depends on $\vec{w}$, and thus on the tilt direction β.

Next, we estimate the shift of $\vec{p}_a$ from the edge pixels extracted in Section 4.2.1.2. For each edge pixel k, we know the position $\vec{p}_k = (u_k', v_k')^T$, as well as the horizontal and vertical edge gradients $g_{k,x}$ and $g_{k,y}$. From this, we calculate the gradient direction angle

$$\varphi_k = \begin{cases} \operatorname{atan2}(-g_{k,y}, -g_{k,x}), & \text{if } g_{k,y} < 0 \vee (g_{k,y} = 0 \wedge g_{k,x} < 0), \\ \operatorname{atan2}(g_y, g_x), & \text{otherwise.} \end{cases} \tag{4.16}$$

This two-part definition ensures that $\varphi_k \in [0, \pi)$ is the same for both light-dark and dark-light edges. We also calculate the edge offset

$$s_k = (\cos(\varphi_k), \sin(\varphi_k))^T (\vec{p}_k - \vec{p}_c). \tag{4.17}$$

s_k is the distance between $\vec{p}_c$ and a line orthogonal to φ_k that passes through $\vec{p}_k$. Figure 4.8 illustrates this geometry for a single edge pixel. If we treat k as part of an infinite, straight edge, s_k would be the distance between that edge and $\vec{p}_c$. A similar parameterization for edge pixels was previously suggested by Davies [30]. For edge pixels k from vertical elements, we expect $s_k = 0$ for an untilted robot. If the robot is tilted, s_k will change based on the tilt parameters (α, β).

Tilt Parameters from Vanishing Point Shifts

In Figure 4.8, a tilt has shifted the vanishing point's location from the expected point $\vec{p}_c$. The new vanishing point at $\vec{p}_a$ is specified by the angle β' and the distance l from $\vec{p}_c$. Here, β' is the image-space tilt direction, from which we will later calculate the tilt direction β of the robot. Due to this shift, the blue line extended from $\vec{p}_k$ no longer passes through $\vec{p}_c$. According to Figure 4.8, the edge offset s_k between the blue line and $\vec{p}_c$ is

$$s_k = l\sin(\epsilon_k) = l\sin\left(\frac{\pi}{2} + \varphi_k - \beta'\right) = l\cos\left(\beta' - \varphi_k\right). \tag{4.18}$$

Using the trigonometric theorem of addition, we can rewrite this as

$$\begin{aligned} s_k &= \left[l\cos(\beta')\right]\cos(\varphi_k) + \left[l\sin(\beta')\right]\sin(\varphi_k) \\ &= A\cos(\varphi_k) + B\sin(\varphi_k). \end{aligned} \tag{4.19}$$

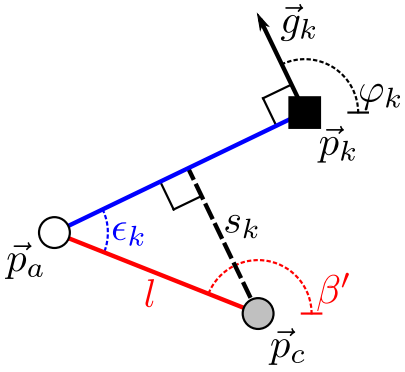


Figure 4.8.: The parameters associated with an edge pixel k. In this illustration, the edge pixel is shown as a black square with position $\vec{p}_k$. This edge pixel has a gradient $\vec{g}_k$, represented by a black arrow. From this gradient, we calculate the gradient direction angle φ_k. The blue line indicates a hypothetical straight edge that is orthogonal to φ_k and passes through $\vec{p}_k$. This line also passes through the approximated vanishing point at $\vec{p}_a$ (white disk). A tilt has shifted $\vec{p}_a$ from the untilted vanishing point $\vec{p}_c$ (gray disk). This shift is represented by a red line, and described by the angle β' and distance l. The distance between the blue line and $\vec{p}_c$ is the edge offset s_k from Equation (4.17). Finally, the angle between the blue and red lines is $\epsilon_k = \frac{\pi}{2} + \varphi_k - \beta'$. Note that we model the effects of a tilt as a simple shift (Figure 4.6).

Each edge pixel k from a vertical element provides us with one instance of Equation (4.19). For N such edge pixels, we then have a system with two unknowns A and B and N linear equations. We can solve this overdetermined system for (A, B) using a linear least squares approach. This lets us determine

$$l = \sqrt{A^2 + B^2} \text{ and} \tag{4.20}$$

$$\beta' = \operatorname{atan2}(B, A). \tag{4.21}$$

Knowing β' and l gives us the position $\vec{p}_a$ of the approximate vanishing point.

Using $\vec{p}_a$ as a substitute for the true location $(u'_t, v'_t)^T$ in Equation (4.15), we get

$$a_0\alpha \begin{pmatrix} -cw_y + dw_x \\ -ew_y + w_x \end{pmatrix} + \begin{pmatrix} x'_c \\ y'_c \end{pmatrix} \approx l \begin{pmatrix} \cos(\beta') \\ \sin(\beta') \end{pmatrix} + \vec{p}_c. \tag{4.22}$$

Recall that for an untilted robot, we assume the optical axis to be orthogonal to the movement plane. In this case, Equation (4.2) simplifies to $\vec{p}_c \approx P((0, 0, 1)^T)$, and from this we can show that $\vec{p}_c \approx (x'_c, y'_c)^T$ using Equations (4.3), (4.4) and (4.7). With this result, we can write Equation (4.22) as

$$a_0\alpha\eta \left[\eta^{-1} \begin{pmatrix} -cw_y + dw_x \\ -ew_y + w_x \end{pmatrix}\right] \approx l \left[\begin{pmatrix} \cos(\beta') \\ \sin(\beta') \end{pmatrix}\right] \tag{4.23}$$

with the magnitude $\eta = \sqrt{(-cw_y + dw_x)^2 + (-ew_y + w_x)^2}$. We note that the two vectors in square brackets both have a length of 1, and can therefore write

$$\alpha = \frac{l}{a_0\eta}, \tag{4.24}$$

$$\eta^{-1} \begin{pmatrix} -cw_y + dw_x \\ -ew_y + w_x \end{pmatrix} = \begin{pmatrix} \cos(\beta') \\ \sin(\beta') \end{pmatrix}. \tag{4.25}$$

For a conventional camera with square pixels, we can approximate $c = 1$ and $d = e = 0$, resulting in

$$\alpha = \frac{l}{a_0}, \tag{4.26}$$

$$(w_x, w_y)^T = \big(\sin(\beta'), -\cos(\beta')\big)^T. \tag{4.27}$$

Note that the tilt axis $\vec{w}$ is given in the camera reference frame. Since we assume that the camera is mounted facing upwards, the z-axis of the robot and the camera should be parallel. However, the x and y axes of the robot and the camera may be rotated relative to each other. This rotation is described by the extrinsic calibration matrix R (Section 4.2.1.1). To estimate the tilt direction β in the robot reference frame, we first determine the robot tilt axis

$$\vec{w}_R = \begin{pmatrix} w_{R,x} \\ w_{R,y} \\ w_{R,z} \end{pmatrix} = R \begin{pmatrix} w_x \\ w_y \\ 0 \end{pmatrix} = \begin{pmatrix} R_{1,1}\sin(\beta') - R_{1,2}\cos(\beta') \\ R_{2,1}\sin(\beta') - R_{2,2}\cos(\beta') \\ R_{3,1}\sin(\beta') - R_{3,2}\cos(\beta') \end{pmatrix}. \tag{4.28}$$

From $\vec{w}_R$, we can finally calculate the tilt direction

$$\begin{aligned} \beta &= \operatorname{atan2}(w_{R,y}, w_{R,x}) - \frac{\pi}{2} = \operatorname{atan2}(-w_{R,x}, w_{R,y}) \\ &= \operatorname{atan2}\left(-R_{1,1}\sin(\beta') + R_{1,2}\cos(\beta'), R_{2,1}\sin(\beta') - R_{2,2}\cos(\beta')\right). \end{aligned} \tag{4.29}$$

Note that Equation (4.29) includes the term $-\frac{\pi}{2}$ because the tilt direction is at a right angle to the tilt axis.

We have made a number of approximations during this derivation. Most noticeably, we assume that a tilt causes a shift in the camera image, disregarding the effects shown in Figure 4.6. During preliminary experiments, we found that this method exhibits systematic errors in the estimated tilt angle α. As per Equation (4.26), α should be proportional to the shift distance l with a coefficient of a_0^{-1}. Since this gives poor results in practice, we replace the camera parameter a_0 with a correction factor a, so that

$$\hat{\alpha} = \frac{l}{a}. \tag{4.30}$$

a is the proportionality constant relating l and α, which we estimate from a set of training images using

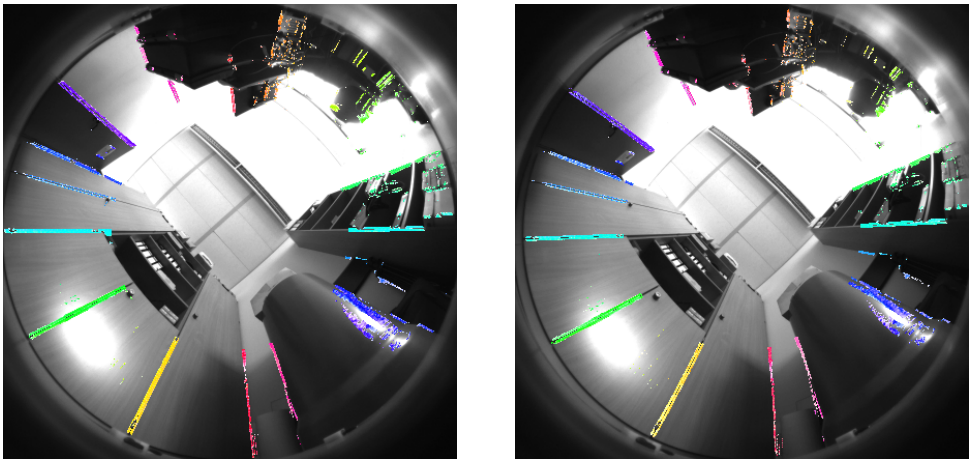
$$a = \frac{1}{m}\sum_{i=1}^{m}\frac{l_i}{\alpha_{T,i}}. \tag{4.31}$$

Here, l_i is the shift distance l measured for the image with index i. $\alpha_{T,i} > 0°$ is the ground truth for α in the ith image, determined according to Section 4.2.3. Thus, a equals the mean ratio between the shift l_i and true tilt angle $\alpha_{T,i}$ over a training set of m tilted images. Using this heuristic, we can improve the accuracy of our estimates while still benefiting from the numerous simplifications made above.

4.2.1.4. Rejecting Incorrect Pixels

So far, we have assumed that all edge pixels correspond to vertical elements. This is not the case in a real environment, as shown by the many non-vertical elements in Figure 4.4. We therefore have to reject incorrect edge pixels caused by these non-vertical elements. Edge pixels disturbed by image artifacts and camera noise should also be filtered out.

As a first step, we apply a prefilter that discards all pixels with a large edge offset s_k: Here, we once again assume that the tilt angle α is small, and thus $\alpha \leq \alpha_{\text{max}}$. According to Equation (4.30),



(a) Untilted, $\alpha_T = \beta_T = 0°$ **(b)** Tilted forward, $\alpha_T = 4.15°, \beta_T = 0°$

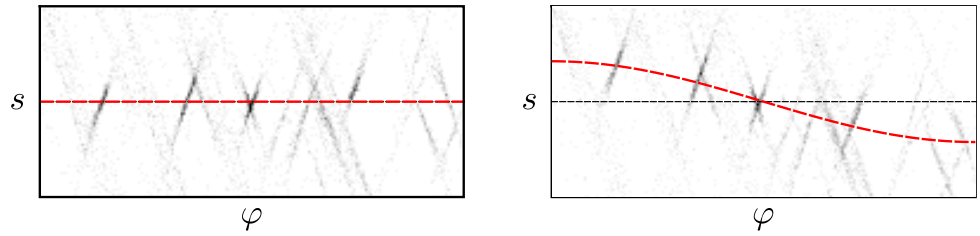
Figure 4.9.: The edge pixels from the prefiltered set F, which we extracted from the Scharr-filtered images in Figure 4.4. Valid edge pixels are shown in color, and are superimposed on the camera image. (**a**) shows the untilted image from Figure 4.4, while (**b**) shows the same location with a forward tilt. We reject pixels with bearings of more than $45°$ above the horizon, or with a gradient intensity of $I_k < 200$. Pixels with an edge offset $|s_k| > s_{\text{max}}$ were also rejected. In this visualization, the pixel's hue indicates its gradient direction angle φ_k. The saturation represents the edge offset s_k, with full saturation and desaturation corresponding to $s_k = 0$ and $|s_k| = s_{\text{max}}$, respectively. (α_T, β_T) are the ground-truth tilt parameters calculated from the measured wheel heights (Section 4.2.3).

an upper bound for α also limits l. From Equation (4.18), we see that $|s_k| \leq l$, and thus a limit on l implies a limit $|s_k| \leq s_{\text{max}}$. We therefore discard edge pixels k with large $|s_k|$, as they are not consistent with $\alpha \leq \alpha_{\text{max}}$. After this prefiltering, the remaining edge pixels form the set F. In this work, we chose $s_{\text{max}} = 40$, which corresponds to $\alpha_{\text{max}} \approx 10°$ for our camera; the precise value of α_{max} depends on the choice of a in Equation (4.30). Figure 4.9 shows the result of this prefiltering step.

We now estimate the tilt parameters (α, β) by fitting a cosine (Equation (4.18)) to the (φ_k, s_k) of the edge pixels in the prefiltered set F. This step is described in Section 4.2.1.3 and illustrated in Figure 4.10. Here, incorrect edge pixels may cause large errors, as demonstrated by Figure 4.11a. Such pixels can arise from non-vertical elements or noise, as seen in Figure 4.12. One solution uses the popular random sample consensus (RANSAC) hypothesize-and-test scheme [44]. RANSAC has previously been used to identify vanishing points from straight lines, for example by Aguilera, Lahoz, and Codes [1] and Wildenauer and Vincze [155]. Under RANSAC, we generate tilt hypotheses from randomly-chosen edge pixels in the prefiltered set F. We then select the hypothesis (α, β) in consensus with the highest number of pixels. Alternatively, we try a reject-refit scheme based on repeated least-squares estimation. Here, we reject edge pixels that disagree with our most recent (α, β) estimate. (α, β) are then reestimated from the remaining pixels. This repeats until the estimate converges, or an iteration limit is reached.

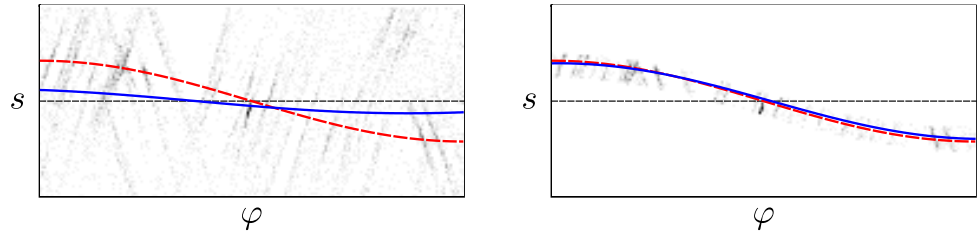
RANSAC Variant

For the RANSAC approach, we randomly select two pixels i and $j \neq i$ from the prefiltered set F. Solving a system of Equation (4.19) for the pixels i, j gives us the hypothesis $(A_{i,j}, B_{i,j})$. We use



(a) Untilted, $\alpha_T = \beta_T = 0°$ **(b)** Tilted forward, $\alpha_T = 4.15°, \beta_T = 0°$

Figure 4.10.: The gradient direction angle φ_k and edge offset s_k of the edge pixels shown in Figure 4.9. Edge pixels are shown as a 2D (φ, s) histogram, with darker bins containing more edge pixels. The red, dashed line shows the (φ, s) cosine predicted from the ground-truth tilt according to Section 4.2.1.3. A thin, dashed black line corresponds to $s = 0$. Due to noise and non-vertical elements in the environment, some edge pixels noticeably deviate from this ideal curve.



(a) No rejection **(b)** After RANSAC

Figure 4.11.: The edge pixels from the image in Figure 4.12, visualized as in Figure 4.10. The blue line represents the (φ, s) cosine fitted to the edge pixels using least squares. In (**a**), incorrect pixels were not rejected. There is thus a large error between the estimate (blue) and ground truth (red). Applying RANSAC with a threshold $\delta_{s,\text{max}} = 5$ pixels gives a much better result, shown in (**b**). Here, the histogram contains only the edge pixels remaining after RANSAC.

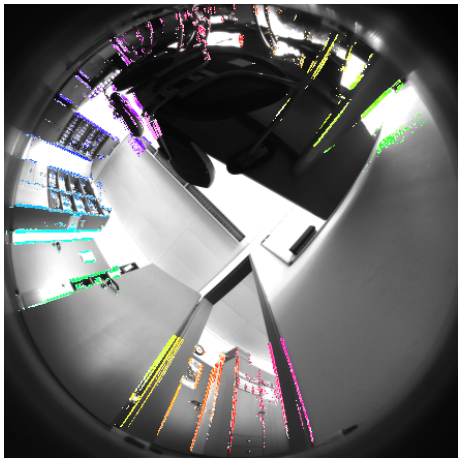


Figure 4.12.: Edge pixels after prefiltering, visualized as in Figure 4.9. This image includes numerous incorrect edge pixels. These are caused by near-vertical elements, such as parts of the curved chairs. If they are not rejected, these pixels cause errors in the least-squares tilt estimate. The image was captured with a forward tilt of $\alpha_T = 4.15°, \beta_T = 0°$.

Equation (4.19) to identify the set of edge pixels in consensus with this hypothesis as

$$C_{i,j} = \bigcup \{k | k \in F \wedge |(A_{i,j}\cos(\varphi_k) + B_{i,j}\sin(\varphi_k) - s_k| < \delta_{s,\max}\}\,. \tag{4.32}$$

We repeat this up to $N_R = 1000$ times, and select the edge pixels $\breve{i}, \breve{j}$, which maximize $\|C_{\breve{i},\breve{j}}\|$. In our experiments, we use a systematic search to select the threshold $\delta_{s,\max}$ (Section 4.2.4). If RANSAC is successful, $C_{\breve{i},\breve{j}}$ contains no incorrect edge pixels. We now construct a linear system of Equation (4.19) using the $(\varphi_{k'}, s_{k'})$ of the edge pixels $k' \in C_{\breve{i},\breve{j}}$. Solving this system through linear least-squares gives us $(\breve{A}, \breve{B})$; here, we use the QR decomposition implemented by the Eigen library [62]. From these $(\breve{A}, \breve{B})$, we can finally estimate (α, β) according to Section 4.2.1.3.

To speed up this process, RANSAC may terminate before reaching the iteration limit N_R [44]: Let p be the fraction of correct edge pixels in the prefiltered set F, with $\|F\| \gg 2$. The probability of drawing at least one pair of correct edge pixels in n attempts is $q \approx 1 - (1 - p^2)^n$. Since we do not know p, we estimate its lower bound as $p_n = \|C_n\|/\|F\|$; C_n is the largest set $C_{i,j}$ encountered after n iterations [147]. The probability of encountering at least one correct pixel pair after n iterations is thus estimated as $q_n \approx 1 - (1 - p_n^2)^n$. We then terminate the RANSAC process once $q_n > q_{\min}$, here using a value of $q_{\min} = 0.9999$.

Reject-Refit-Variant

In our reject-refit scheme, we alternate between estimating (A, B) from Equation (4.19) and rejecting incorrect pixels. We use least squares to estimate (A_n, B_n) from the edge pixels in F_n, beginning with $F_0 = F$. This estimate is affected by incorrect pixels in F, as shown in Figure 4.11a. To reject these pixels, we calculate the residual error

$$\delta_{k,n} = (A_n\cos(\varphi_k) + B_n\sin(\varphi_k)) - s_k \tag{4.33}$$

for each pixel in F_n. Next, we form F_{n+1} by rejecting a fraction Q of pixels in F_n that have the largest absolute error $|\delta_{k,n}|$. Identifying the actual set of pixels with the largest $|\delta_{k,n}|$ requires sorting F_n. This is computationally expensive, and thus we use another heuristic: We assume that the $\delta_{k,n}$ of the pixels in F_n are normally distributed, with mean $\mu_{\delta,n} = 0$ and standard deviation $\sigma_{\delta,n}$. In an idealized case, only a fraction $1 - Q = 2\Phi(z) - 1$ of the edge pixels k in F_n satisfies $\delta_{k,n} \in [-z\sigma_{\delta,n}, z\sigma_{\delta,n}]$ [64]; here, Φ is the cumulative distribution function of the standard normal distribution. For the remaining fraction Q with $\delta_{k,n}$ outside of this interval, we therefore expect

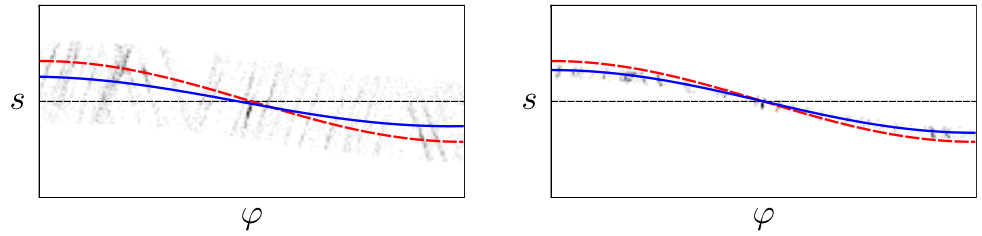
$$|\delta_{k,n}| > \sigma_{\delta,n}\Phi^{-1}\left(1 - \frac{Q}{2}\right). \tag{4.34}$$

We can thus simply construct the next, smaller set

$$F_{n+1} = \left\{k | k \in F_n \wedge |\delta_{k,n}| \leq \sigma_{\delta,n}\Phi^{-1}\left(1 - \frac{Q}{2}\right)\right\} \tag{4.35}$$

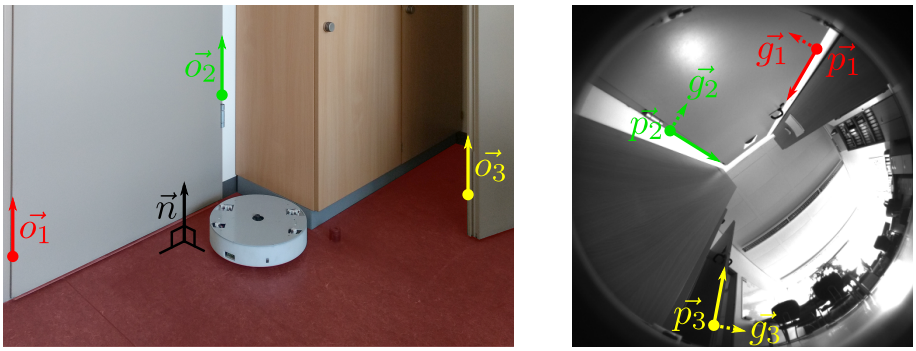
without sorting F_n.

While the $\delta_{k,n}$ in F_n are not actually normally distributed, we found that this fast heuristic nonetheless gives good results. After these two fitting and rejection steps, we increment n and repeat the procedure. This continues until the tilt estimate converges, or until n reaches the limit $N_E = 15$. We assume that the estimate has converged once $|l_n - l_{n-1}| < 0.1$, as calculated from Equation (4.20). Figure 4.13 demonstrates the effect of this reject-refit heuristic.



(a) Two iterations, $Q = 0.2$ **(b)** Eight iterations, $Q = 0.2$

Figure 4.13.: Using the reject-refit scheme to reject incorrect edge pixels from Figure 4.12. Edge pixels are visualized through (φ, s) histograms, as in Figure 4.11. (**a**) shows the remaining edge pixels after the first two iterations. The error between the ground truth (red) and the curve fitted to the remaining pixels F_2 (blue) is reduced, compared to Figure 4.11a. Once the reject-refit scheme converges for $n = 8$, the error shown in (**b**) is even lower.



(a) Vertical elements **(b)** Robot camera image

Figure 4.14.: Vertical elements in the environment and their appearance in the camera image. In (**a**), the orientations $\vec{o}_k$ for some vertical elements are indicated by arrows. The robot's movement plane is highlighted in red, with a black arrow representing the plane's surface normal $\vec{n}$. Note that the $\vec{o}_k$ and $\vec{n}$ are parallel. (**b**) shows the robot's camera view in this location. The vertical elements from (**a**) appear as straight edges. Colored circles represent the edge pixels that correspond to the vertical elements in (**a**). Each such pixel has a position $\vec{p}_k$ and gradient vector $\vec{g}_k$; the latter are shown as dashed lines. Since the camera image contains only two-dimensional projections of the three-dimensional $\vec{o}_k$, we cannot determine these orientations directly.

4.2.2. Vector-Consensus Method

The vector-consensus method is the second tilt-estimation scheme used in this work. It operates on the same edge pixels as the image-space method described in Section 4.2.1.2. As before, we assume that many of these pixels correspond to vertical elements k in the environment. Since they are vertical, their orientations $\vec{o}_k$ are parallel to the movement-plane normal $\vec{n}$. We illustrate this relationship in Figure 4.14a. Knowing their orientation $\vec{o}_{k,R} \parallel \vec{n}_R$ in robot coordinates lets us calculate (α, β) using Equation (4.1). However, as seen in Figure 4.14b, our camera image shows only two-dimensional projections of these three-dimensional elements. We therefore cannot determine $\vec{o}_{k,R}$ from any single edge pixel k, and instead use several pixels from different vertical elements. To identify a set of such edge pixels, we combine a problem-specific prefilter with RANSAC. This approach is highly similar to RANSAC-based vanishing-point detection, as for example in [1].

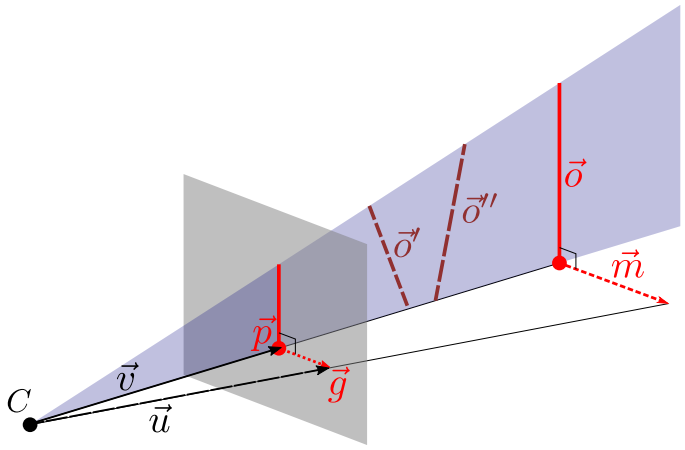


Figure 4.15.: An environment element with orientation $\vec{o}$ is projected onto an image plane (gray). This produces an edge pixel with position $\vec{p}$ and gradient $\vec{g}$. $\vec{v} = P^{-1}(\vec{p})$ is the 3D bearing vector associated with the image position $\vec{p}$, and similarly $\vec{u} = P^{-1}(\vec{p} + \Delta\vec{g})$. The normal $\vec{m}$ specifies the orientation of the blue plane that contains both $\vec{v}$ and $\vec{o}$, this plane is only partially drawn as a triangle. We cannot fully determine $\vec{o}$ from the projection $\vec{p}$ and $\vec{g}$ alone. Any element in the blue plane — such as $\vec{o}'$ or $\vec{o}''$ — would result in the same edge pixel. All vectors in this illustration are relative to the camera reference frame. Note that we use a linear camera as an approximation of the actual camera model from Figure 4.7. This is plausible because the projection is approximately linear within a small radius Δ around the pixel $\vec{p}$: within this radius, $\rho \approx$ const., and thus Equation (4.3) is a linear projection.

4.2.2.1. Orientation Estimation

We first estimate the orientations $\vec{o}_{i,C} \parallel \vec{o}_{j,C}$ of two separate parallel elements from the edge pixels i, j; here, $\vec{o}_{i,C}$ is $\vec{o}_i$ in the camera reference frame. From Section 4.2.1.2, we know the image position $\vec{p}_k$ and gradient vector $\vec{g}_k = (g_{k,x}, g_{k,y})^T$ for both i and j. The camera's inverse projection function P^{-1} provides the bearing $\vec{v}_i = P^{-1}(\vec{p}_i)$ from the projection center towards the edge pixel i. As illustrated in Figure 4.15, $\vec{o}_{i,C}$ and $\vec{v}_i$ define a plane in space. This plane has the normal vector $\vec{m}_i = \vec{o}_{i,C} \times \vec{v}_i$. We repeat this step for a second edge pixel j, giving us the normal $\vec{m}_j$ for a plane containing $\vec{o}_{j,C}$ and $\vec{v}_j$. Since the orientations $\vec{o}_{i,C} \parallel \vec{o}_{j,C}$ are orthogonal to both $\vec{m}_i$ and $\vec{m}_j$, we have

$$\vec{o}_{i,C} \parallel \vec{o}_{j,C} \parallel (\vec{m}_i \times \vec{m}_j). \tag{4.36}$$

This plane-based formalism [92] is commonly used in vanishing-point estimation, for example in [5, 8, 83].

While we do not know $\vec{m}_i$, we can estimate it using the edge gradient $\vec{g}_i$. Shifting the pixel position $\vec{p}_i$ by a small distance Δ along the gradient $\vec{g}_i$, we calculate $\vec{u}_i = P^{-1}(\vec{p}_i + \Delta\vec{g}_i)$; in this work we use $\Delta = 0.01$. As illustrated in Figure 4.15, $\vec{g}_i$ is a projection of $\vec{m}_i$, and $\vec{m}_i$, $\vec{v}_i$, and $\vec{u}_i$ lie in the same plane. $\vec{u}_i \times \vec{v}_i$ is a normal vector of this plane, and therefore orthogonal to $\vec{m}_i$. By definition, $\vec{m}_i$ is also orthogonal to $\vec{v}_i$, leading us to

$$\vec{m}_i \parallel (\vec{v}_i \times (\vec{u}_i \times \vec{v}_i)) . \tag{4.37}$$

Note that we have assumed a continuous P^{-1} when calculating $\vec{u}_i$. Therefore, shifting $\vec{p}$ by a small distance Δ leads to only a small change in $P^{-1}(\vec{p})$. In addition, the camera should use central projection. The camera model used in this work [132] generally fulfills these requirements. From Equation (4.36) and Equation (4.37), we thus estimate the orientation $\vec{o}_{i,C} \parallel \vec{o}_{j,C}$ from the pixel

positions $\vec{p}_i, \vec{p}_j$ and gradients $\vec{g}_i, \vec{g}_j$. If $\vec{o}_{i,C} \parallel \vec{o}_{j,C}$ are the orientations of vertical elements, we can use them to estimate the robot's tilt.

In practice, estimating $\vec{o}_{i,C}$ from just two pixels will give poor results due to noise. We therefore use a set E of two or more edge pixels with $\vec{o}_{i,C} \parallel \vec{o}_{j,C}\ \forall i,j \in E$. By applying Equation (4.37), we determine $\vec{m}_k$ for each pixel $k \in E$. Each $\vec{m}_k$ is orthogonal to $\vec{o}_{k,C}$, with $\vec{m}_k^T \vec{o}_{k,C} = 0$. Since all $\vec{o}_{k,C}$ in E are parallel, we replace them with a general orientation $\vec{o}_C \parallel \vec{o}_{k,C}\ \forall k \in E$, giving us the constraint

$$\vec{m}_k^T \vec{o}_C = 0. \tag{4.38}$$

We now construct a system of equations that contains one instance of Equation (4.38) for each pixel $k \in E$. To avoid the trivial solution of $\vec{o}_C = \vec{0}$, we add the additional equation of $(1,1,1)^T \vec{o}_C = 1$ to the system. Solving this overdetermined system using linear least squares gives us an estimate for $\vec{o}_C$. In practice, we use the singular value decomposition (SVD) implemented in the Eigen library [62].

4.2.2.2. Vertical Edge Selection

Next, we identify a set of edge pixels that corresponds to vertical elements in the environment. Using a prefilter, we discard any edge pixels that indicate an implausibly large tilt angle α. Note that $\vec{m}_k$ is given in the camera reference frame, which may be rotated relative to the robot reference frame. We therefore transform $\vec{m}_k$ to the robot frame by multiplying with the extrinsic calibration matrix R. If $\vec{o}_k$ is actually vertical, then $R\vec{m}_k$ is parallel to the robot's movement plane, and

$$\hat{\alpha}_k = \left| \arcsin\left(\vec{n}^T \frac{R\vec{m}_k}{\|\vec{m}_k\|} \right) \right| \leq \alpha. \tag{4.39}$$

If $\hat{\alpha}_k > \alpha_{\max}$, we therefore reject the edge pixel k. The remaining edge pixels form the set E. Note that $\hat{\alpha}_k$ is always 0 if the tilt axis is parallel to $R\vec{m}_k$.

Similar to Section 4.2.1.3, we apply RANSAC to identify edge pixels with vertical $\vec{o}_k$ [44]. We randomly select two edge pixels $i, j \neq i$ from E, and use Equation (4.36) to generate a hypothesis $\vec{o}_{i,j,C}$. The set $C_{i,j}$ of edge pixels in consensus with this hypothesis is

$$C_{i,j} = \bigcup \left\{ k | k \in E \wedge \left| \vec{m}_k^T \vec{o}_{i,j,C} \right| < \delta_{o,\max} \right\}. \tag{4.40}$$

As in Section 4.2.1.3, we repeat this step until termination ($N_R = 1000$ and $q_{\min} = 0.9999$). If RANSAC is successful, the largest consensus set $C_{\breve{i},\breve{j}}$ contains only vertical edge pixels with $\vec{n} \parallel \vec{o}_k\ \forall k \in C_{\breve{i},\breve{j}}$. Figure 4.16 gives an example for the $\vec{m}_k$ of the pixels in $C_{\breve{i},\breve{j}}$, while Figure 4.17 shows the corresponding edge pixels in the camera image.

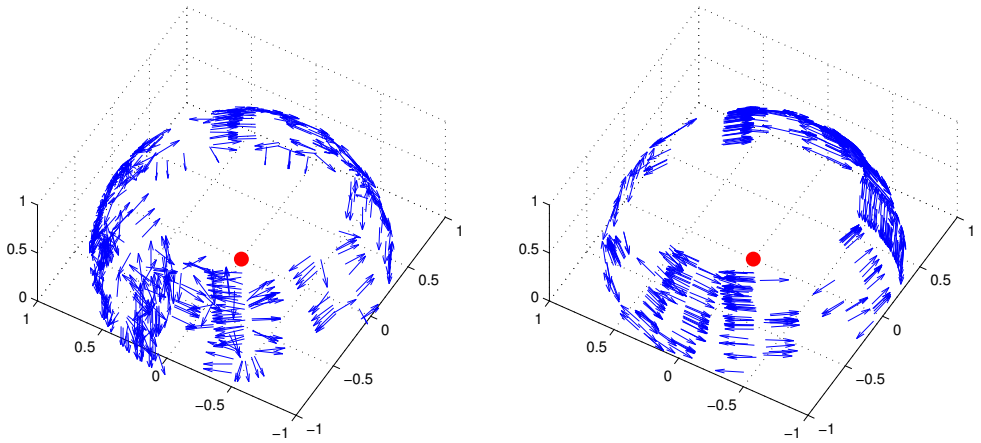
We now use the least-squares approach from Section 4.2.2.1 to determine the common orientation $\vec{o}_C$ from all the pixels in $C_{\breve{i},\breve{j}}$. Since $\vec{o}_C$ should be vertical, we estimate the movement plane normal $\vec{n}_R$ as

$$\vec{n}_{R,\text{est}} = R \frac{\vec{o}_C}{\|\vec{o}_C\|}. \tag{4.41}$$

Finally, we calculate the tilt parameters (α, β) from $\vec{n}_{R,\text{est}}$ using Equation (4.1). For some R, the constraint $(1,1,1)^T \vec{o}_C = 1$ in Section 4.2.2.1 may result in a $\vec{n}_{R,\text{est}}$ that is antiparallel to $\vec{n}_R$. Since we know that $\alpha \ll 90°$, we simply flip $\vec{n}_{R,\text{est}}$ for results of $\alpha > 90°$.

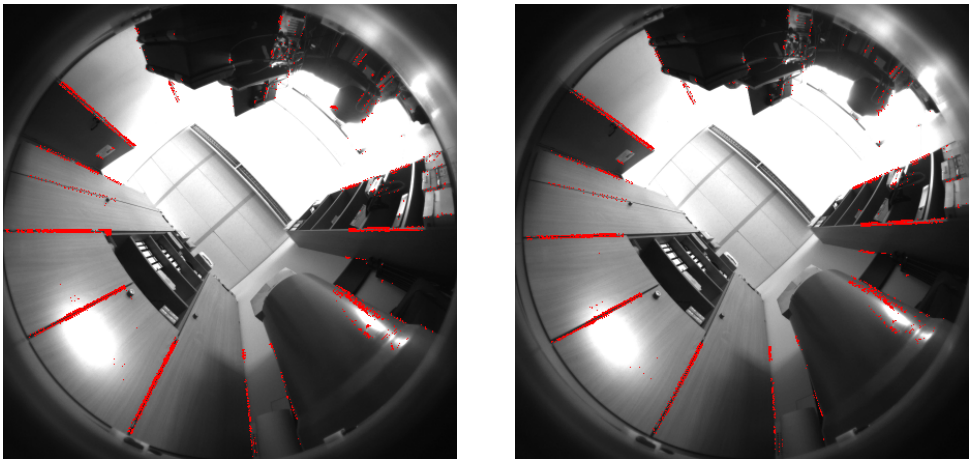
4.2.3. Image Database

To evaluate the methods from Sections 4.2.1 and 4.2.2, we recorded an image database using our cleaning-robot prototype. We capture the images in our camera's 640×480 pixel half-resolution



(a) Unfiltered **(b)** After RANSAC

Figure 4.16.: The normal vectors $\vec{m}_k$ of the edge pixels for an untilted robot. For the sake of clarity, only 500 randomly selected edge pixels are shown. Each edge pixel k is shown as an arrow, with the orientation representing the normal vector $\vec{m}_k$. The base of each arrow lies along the bearing $\vec{v}_k$ belonging to the pixel k. (**a**) is based on the edge pixels extracted from the camera image in Figure 4.17a. As in Figure 4.17, we do not include edge pixels with a gradient intensity below $I_{\text{min}} = 200$. (**b**) shows the result of prefiltering ($\alpha_{\text{max}} = 7°$) and RANSAC ($\delta_{o,\text{max}} = 3.5°$). As expected for an untilted robot, the remaining $\vec{m}_k$ are approximately orthogonal to the vertical axis.



(a) Untilted, $\alpha_T = \beta_T = 0°$ **(b)** Tilted forward, $\alpha_T = 4.15°, \beta_T = 0°$

Figure 4.17.: Edge pixels corresponding to vertical elements, as identified through RANSAC. Edge pixels within the largest set $C_{\tilde{i},\tilde{j}}$ are marked in red. The camera images are the same as used in Figure 4.9, cropped to the area above the horizon. Edge pixels with a gradient intensity below $I_{\text{min}} = 200$ were discarded and are not shown. After prefiltering with $\alpha_{\text{max}} = 7°$, we applied RANSAC with $\delta_{o,\text{max}} = 3.5°$.

(a) **(b)** **(c)**

Figure 4.18.: The robot at three different locations from our image database. Besides open areas (**a**), we also captured images in narrow spaces (**b**) and underneath furniture (**c**).

Object	h	α_T **(Main wheel)**	α_T **(Caster)**
Spacer (thin)	$5.0\,\mathrm{mm}$	$1.38°$	$2.06°$
Spacer (thick)	$10.1\,\mathrm{mm}$	$2.80°$	$4.15°$
Carpets (various)	$4.4\,\mathrm{mm}$ to $7.4\,\mathrm{mm}$	$1.22°$ to $2.05°$	$1.81°$ to $3.04°$
Door threshold	$5.3\,\mathrm{mm}$	$1.47°$	$2.18°$

Table 4.1.: The ground-truth tilt angle α_T when raising one wheel by a distance h, as calculated from Equation (4.42). Raising one of the main wheels results in $\beta_T = \mp 137°$ for the left and right wheel, respectively (Equation (4.43)). $\beta_T = 0°$ when raising the caster wheel, which holds up the rear of the robot. *Spacer* refers to the metal spacers which we use to capture tilted images. We also include common objects from domestic environments that may cause the robot to tilt. The α_T values in this table were calculated for a stationary robot. For a moving robot, driving-related forces or torques may affect the true tilt angle or direction.

mode. This lower resolution speeds up image processing, and is also used by our robot when capturing images for visual relative-pose estimation. As per Section 4.2.1.2, we mask out pixels showing areas below the horizon. The remaining pixels form a disc-shaped area with a diameter of 439 pixels, as seen in Figure 4.3. While capturing images, the exposure time is automatically adjusted to maintain a set average image brightness. In other respects, the image-acquisition process corresponds to the one we describe in Section 3.2.1. Here, we also use the same intrinsic and extrinsic camera parameters as in Chapter 3.

We captured images in 43 different locations spread across six different environments. The six environments consist of four different offices, as well as two lab environments. In some cases, we positioned the robot in narrow spaces or underneath furniture, as in Figure 4.18. The resulting restricted field of view may pose a special challenge for tilt estimation. Our image database contains between four and eleven locations per environment. Due to this limited number, the locations do not provide a representative sample of their environment. However, completely covering each environment would require a large number of images. We also expect that many of these locations would be visually similar. Instead, we focused on picking a smaller, but highly varied set of locations.

At each location, we captured one untilted plus six tilted image, for a total of $43 \times (1 + 6) = 301$. We produced six distinct tilts by placing a $5\,\mathrm{mm}$ or $10.1\,\mathrm{mm}$ metal spacer under one of our robot's three wheels. The thickness of these spacers is similar to common sources of uneven ground: In preliminary experiments, various types of carpet raised the robot's wheels by $4.4\,\mathrm{mm}$ to $7.4\,\mathrm{mm}$. Similarly, a typical door threshold raised it by $5.3\,\mathrm{mm}$. Table 4.1 lists the tilts experienced by our robot for these wheel heights. Note that placing the spacers may cause slight changes in robot position or heading between images.

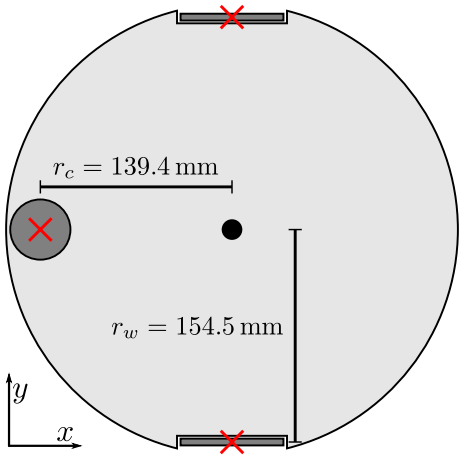


Figure 4.19.: The wheel layout of our cleaning-robot prototype. The panoramic camera (black circle) is mounted at the center of a ground plate (light gray). A red X marks the ground-contact points for each wheel, on which the robot rests. Relative to the center of the camera, the caster wheel's (gray circle) contact point is at $(-r_c, 0)$. The contact points of the left and right main wheel (gray rectangles) lie at $(0, \pm r_w)$, respectively. Here, we assume that the contact points are fixed and not affected by tilts.

We use a simple statics model to calculate the true tilt angle α_T and direction β_T from the measured wheel heights l, r and c. Assuming that the left wheel as our reference point, we calculate the relative heights $h_r = r - l$ and $h_c = c - l$. From the robot geometry shown in Figure 4.19, we find that

$$\sin(\alpha_T) = \frac{1}{2}\sqrt{\left(\frac{2h_c - h_r}{r_c}\right)^2 + \left(\frac{h_r}{r_w}\right)^2} \tag{4.42}$$

for the tilt angle α_T. If the robot is tilted with $\sin(\alpha_T) \neq 0$, the tilt direction is

$$\beta_T = \operatorname{atan2}\left(\frac{h_r}{2r_w \sin(\alpha_T)}, \frac{2h_c - h_r}{2r_c \sin(\alpha_T)}\right). \tag{4.43}$$

4.2.4. Experiment Design

We test the methods from Sections 4.2.1 and 4.2.2 on the images gathered in Section 4.2.3. The estimation-error angle

$$\epsilon = \arccos\left(\frac{\vec{n}_R^T}{\|\vec{n}_R\|}\frac{\vec{n}_{R,\text{est}}}{\|\vec{n}_{R,\text{est}\|}}\right) \tag{4.44}$$

serves as our measure of tilt-estimation accuracy. ϵ is the residual, uncorrected tilt angle that remains after a tilt correction based on the estimate $\vec{n}_{R,\text{est}}$. $\vec{n}_R$ and $\vec{n}_{R,\text{est}}$ are calculated from (α_T, β_T) and (α, β), respectively, by inverting Equation (4.1).

We evaluate numerous parameter values for our methods, seeking to minimize the mean error $\bar{\epsilon}$. For both methods, we try several gradient-intensity thresholds I_{min}. With the image-space method, we also vary the rejection fraction Q or the RANSAC threshold $\delta_{s,\text{max}}$ from Section 4.2.1.4. Similarly, we try multiple values for the RANSAC threshold $\delta_{o,\text{max}}$ from Section 4.2.2.2. Table 4.2 lists the specific values tested during our experiments.

For the image-space method, we determine the factor a from Equation (4.30) using cross-validation: Given a location L, the set T_L contains all tilted images not captured at L. Next,

Parameter	Values tested
Gradient intensity threshold I_{min}	100, 150, 200, 250, 300, 600
Rejection fraction Q	0.5, 0.7, 0.8, 0.9
RANSAC threshold $\delta_{s,\text{max}}$	2.5, 5, 7.5, 10, 12.5, 15 pixels
RANSAC threshold $\delta_{o,\text{max}}$	$0.5°$, $1°$, $1.5°$, $2°$, $2.5°$, $3°$, $3.5°$, $4°$, $4.5°$, $5°$

Table 4.2.: Parameter values evaluated in our experiments. Depending on the method and variant used, each value for I_{min} was tested with all other values for Q, $\delta_{s,\text{max}}$, and $\delta_{o,\text{max}}$. This results in $6 \times 5 = 30$ combinations for (I_{min}, Q), $6 \times 6 = 36$ for $(I_{\text{min}}, \delta_{s,\text{max}})$, and $6 \times 10 = 60$ for $(I_{\text{min}}, \delta_{o,\text{max}})$.

Method	Variant	Parameters	Estimation error ϵ (°)		
			$\bar{\epsilon}$ (σ_ϵ)	50%	95%
Image space	RANSAC	$I_{\text{min}} = 150$, $\delta_{s,\text{max}} = 10.0$	1.17 (1.03)	0.97	3.16
	reject-refit	$I_{\text{min}} = 200$, $Q = 0.8$	0.85 (0.81)	0.61	2.27
Vector consensus	RANSAC	$I_{\text{min}} = 600$, $\delta_{o,\text{max}} = 5.0°$	1.63 (0.93)	1.48	3.38
	", corrected	$I_{\text{min}} = 300$, $\delta_{o,\text{max}} = 2.0°$	1.03 (0.72)	0.92	2.30

Table 4.3.: The mean tilt-estimation error $\bar{\epsilon}$ and standard deviation (σ_ϵ) for the methods and variants tested in our experiments. The last line lists the vector-consensus results achieved when correcting the estimated tilt angle α using the factor a'_L (Section 4.2.4). For each method, we only list the results for the parameters given in the third column. Out of the possibilities from Table 4.2, these values gave the lowest $\bar{\epsilon}$. We also include the median (50th percentile) and 95th percentile of ϵ.

we compute the factor a_L by applying Equation (4.31) to the $m = (43 - 1) \times 6$ images in T_L. For images taken at location L, we then calculate the tilt angle α by using Equation (4.30) with $a = a_L$. Using the parameters from Table 4.3, we find $a_L \in [298, 314]$ pixels per radian when using RANSAC. Likewise, $a_L \in [234, 239]$ pixels per radian for the reject-refit variant. By comparison, the coefficient determined through camera calibration is $a_0 = 147$ pixels per radians (Equation (4.26)).

The vector-consensus method estimates the tilt angle α without major approximations, and should not require a correction factor. However, for the sake of fairness, we also tested this method with an optional correction factor a'. The corrected tilt-angle estimate is then

$$\alpha' = \frac{\alpha}{a'}. \tag{4.45}$$

ϵ for this corrected vector-consensus variant is then calculated from α' instead of α. We determine the specific a'_L for the location L using the same cross-validation scheme as for a_L. For the parameters in Table 4.3, we find a unitless factor $a'_L \in [1.77, 1.80]$.

When running on a robot's onboard CPU, the tilt-estimation methods may be subject to strict real-time constraints. As part of our experiments, we therefore measure execution times on two different systems. The first system is equipped with an Intel Atom N2600 CPU, which represents a typical embedded platform. This 1.6 GHz, dual-core processor is also used in our robot's onboard computer (Section 2.2). For comparison, we also include a modern desktop PC with a quad-core Intel Core i7-4790K CPU. We measure the wall-clock execution time using each system's monotonic, high-resolution clock. This includes all major steps required to estimate (α, β) from a camera image. Non-essential operations, such as loading and converting input data,

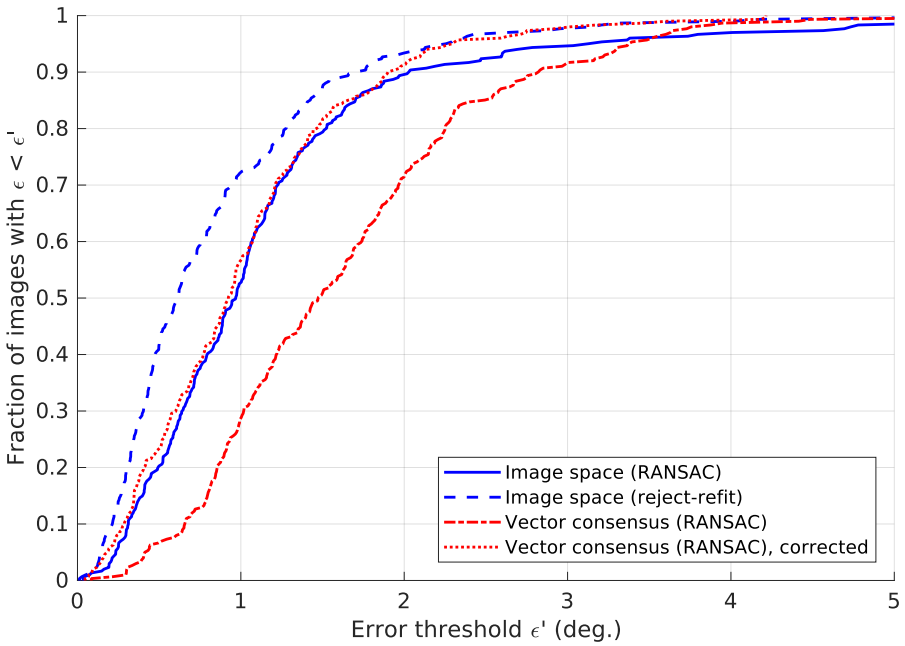


Figure 4.20.: The fraction of images for which the tilt-estimation error is $\epsilon \leq \epsilon'$. All curves were generated using the parameters from Table 4.3. For the sake of clarity, this figure was truncated to $\epsilon' \leq 5$. We note that high errors ϵ can occasionally occur for all methods.

were excluded from this measurement. All experiments involving RANSAC use the same random seed on both platforms. This ensures that the number of RANSAC iterations is identical. So far, our implementations do not consistently support multi-core execution. We therefore use Linux's non-uniform memory access (NUMA) policies to restrict each experiment to a single, fixed CPU core. These systems and procedures correspond to those used for the execution-time measurements in Section 3.2.4.2. Thus, we can directly compare the tilt-estimation execution times to our earlier pose-estimation results.

4.3. Results

4.3.1. Tilt Estimation Error

Table 4.3 lists the tilt-estimation error ϵ, while Figure 4.20 shows the cumulative distributions of ϵ. The latter specifies the fraction of images for which ϵ remains below a given threshold ϵ'. Figure 4.21 plots the error ϵ depending on the true tilt angle α_T. We also evaluate how the methods perform across the different environments in our database, resulting in Figure 4.22.

4.3.2. Computation Time

Table 4.4 contains the execution time taken by the methods, using the parameters from Table 4.3. In our experiments, this time depends heavily on the parameters used. However, these parameters also affect the quality of the results. Figures 4.23 and 4.24 therefore show the execution times in relation to the mean error $\bar{\epsilon}$.

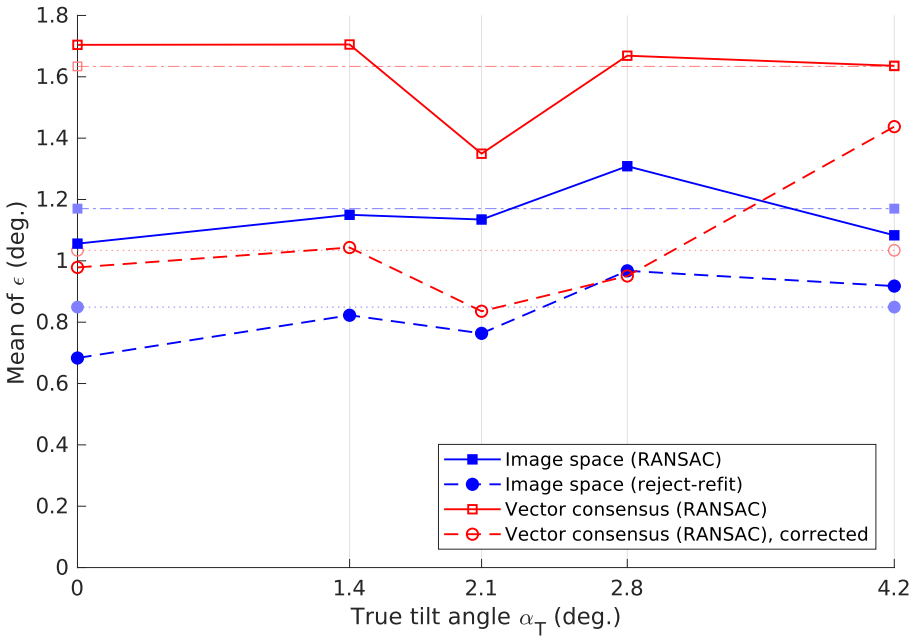


Figure 4.21.: The mean tilt-estimation error $\bar{\epsilon}$, depending on the true tilt angle α_T. The pale, dotted or dash-dotted lines represent each method's $\bar{\epsilon}$ across all images. As in Figure 4.20, we used the parameters from Table 4.3. The ground-truth tilt angle α_T was calculated using Equation (4.42).

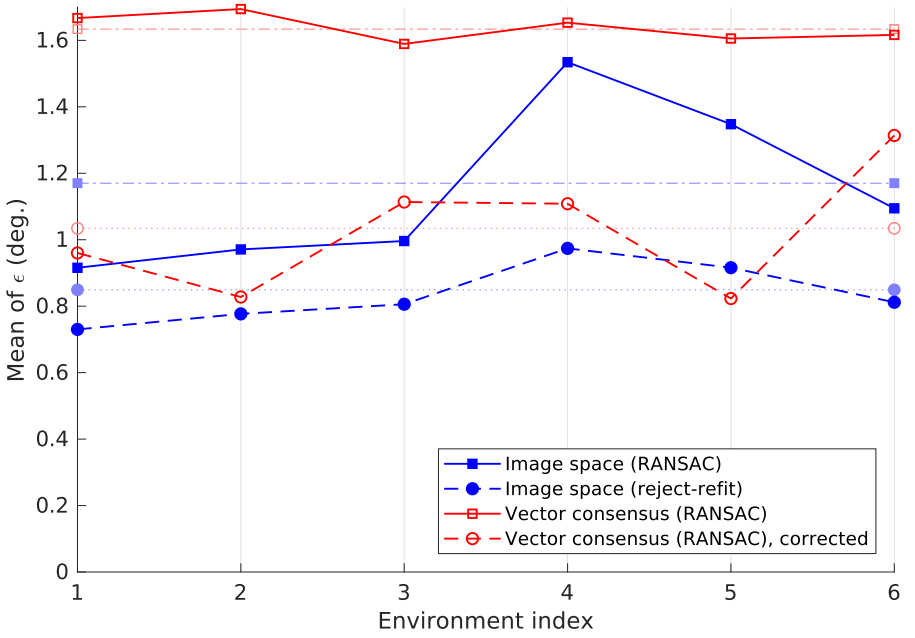


Figure 4.22.: The mean tilt-estimation error $\bar{\epsilon}$ for each of the six environments. The methods and parameters used were the same as in Table 4.3. The pale, dotted or dash-dotted lines represent each method's $\bar{\epsilon}$ across all images. Although there is some variation, each method's tilt-estimation results are broadly similar across the different environments.

Method	Variant	Desktop (ms)			Embedded (ms)		
		Mean (σ_t)	50%	95%	Mean (σ_t)	50%	95%
Image space	RANSAC	3.8 (1.43)	3.7	6.3	26.0 (6.89)	25.2	41.8
	reject-refit	2.5 (0.85)	2.5	4.1	15.7 (3.02)	15.3	20.7
Vector consensus	RANSAC	3.6 (1.22)	3.5	5.7	22.9 (4.69)	23.2	29.4
	", corrected	6.2 (1.88)	6.0	9.3	38.1 (9.72)	36.9	51.8

Table 4.4.: The time required for tilt estimation in milliseconds, as per Section 4.2.4. This table includes the mean, standard deviation σ_t, median (50th percentile), and 95th percentile. Times are given for the modern desktop CPU, as well as for the embedded CPU carried by our cleaning-robot prototype. Each method used the parameters listed in Table 4.3, which gave the lowest mean error $\bar{\epsilon}$. Note that the corrected vector-consensus method appears to be much slower than the uncorrected variant. This is not due to the correction step, which consumes little time. Instead, the corrected variant achieves its lowest $\bar{\epsilon}$ for different parameter values (Table 4.3). However, these values also lead to longer execution times. Compared to the uncorrected variant, the corrected variant actually attains better $\bar{\epsilon}$ in a similar amount of time (Figures 4.23 and 4.24).

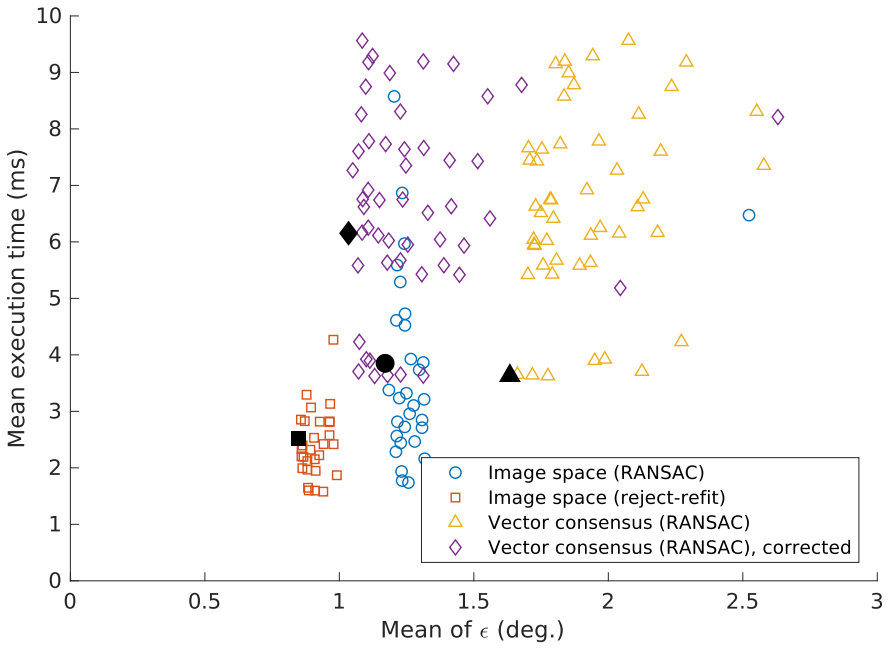


Figure 4.23.: The mean tilt-estimation error $\bar{\epsilon}$, plotted against the mean execution time. The time was measured on the modern desktop system, as described in Section 4.2.4. Each point represents one parameter combination from Table 4.2. The points with the lowest $\bar{\epsilon}$ from Table 4.3 are highlighted in black. As shown here, accepting a slightly higher $\bar{\epsilon}$ can sometimes notably reduce the execution time. We limit this figure to $\bar{\epsilon} \leq 3°$ and $t \leq 10\,\mathrm{ms}$. This causes a few points to be omitted, but greatly improves legibility.

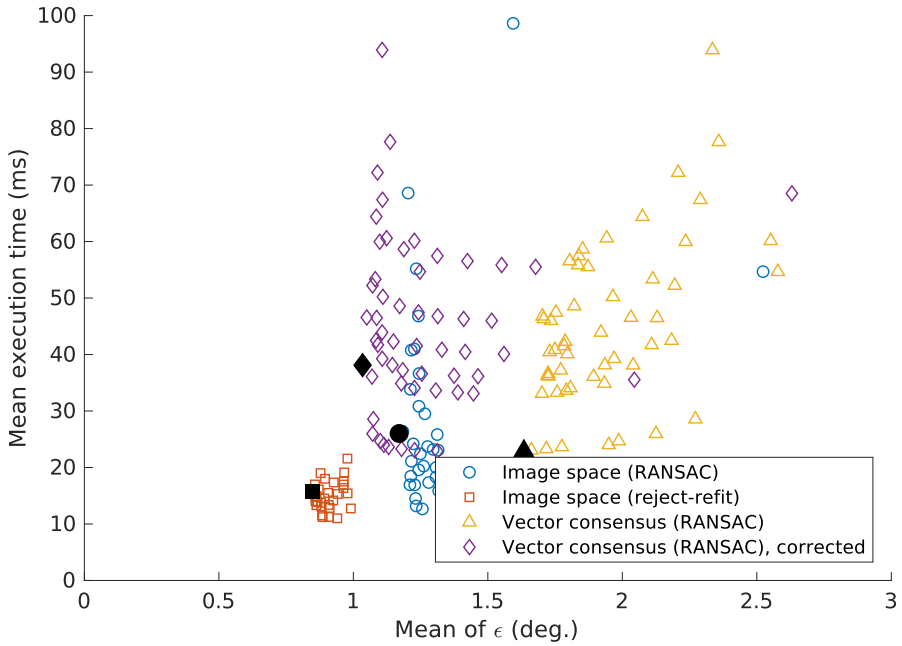


Figure 4.24.: This variant of Figure 4.23 shows the results for the embedded system described in Section 4.2.4. Similar to Figure 4.23, we limit this figure to $\bar{\epsilon} \leq 3°$ and $t \leq 100\,\mathrm{ms}$.

4.4. Discussion

As shown by Table 4.3, all variants achieve at least fair results, with $\bar{\epsilon} < 2°$. The image-space method with the reject-refit scheme achieves the lowest mean and median error. Combining the image-space method with a basic RANSAC scheme gives only mediocre results. In contrast, the vector-consensus approach led to the highest $\bar{\epsilon}$. However, its accuracy improves noticeably when the tilt-angle estimate is corrected by the factor a'.

The practical impact of the tilt-estimation error ϵ depends on the application. For example, our cleaning robot uses the planar-motion min-warping method [102] for visual relative-pose estimation (Section 2.5.2). As we discovered in Section 3.3.3 and visualize in Figure 4.25, the pose-estimation error increases rapidly for α over $\approx 1°$. Beyond $\approx 2°$, the error exceeds that of the nonplanar five-point method [138] which we tested in Chapter 3. In Figure 4.20, we see that the best candidate (image space, reject-refit) achieves $\epsilon < 1°$ for $>70\,\%$ of all images. All but the worst candidate (vector consensus, no correction) also attain $\epsilon \leq 2°$ for at least $\approx 90\,\%$ of images. For tilt correction, ϵ is the angle α of the remaining uncorrected tilt. We thus expect that a correction based on these estimates should make planar-motion pose estimation more resilient to tilts. Studying the quantitative effect of tilt correction on our robot's cleaning behavior will require extensive experimentation. Ideally, such experiments would measure the true tilt of a driving robot using an additional sensor. It may also be useful to mount the camera on a motorized platform, which is then carried by our robot. This way, we could induce arbitrary camera tilts while the robot is performing its regular cleaning run. Since this likely requires a dedicated study, we leave these experiments to future works.

Besides the comparatively low average errors, all methods do exhibit occasional high ϵ. These errors can even exceed the highest tilt angle $\alpha_T = 4.15°$ in the database. A correction with such an erroneous estimate would likely give worse results than any uncorrected tilt. The fraction of images for which this occurs varies by method, as shown in Figure 4.20. In our experiments, such failures generally result from a violation of our world assumption: Both methods rely on visually-distinct

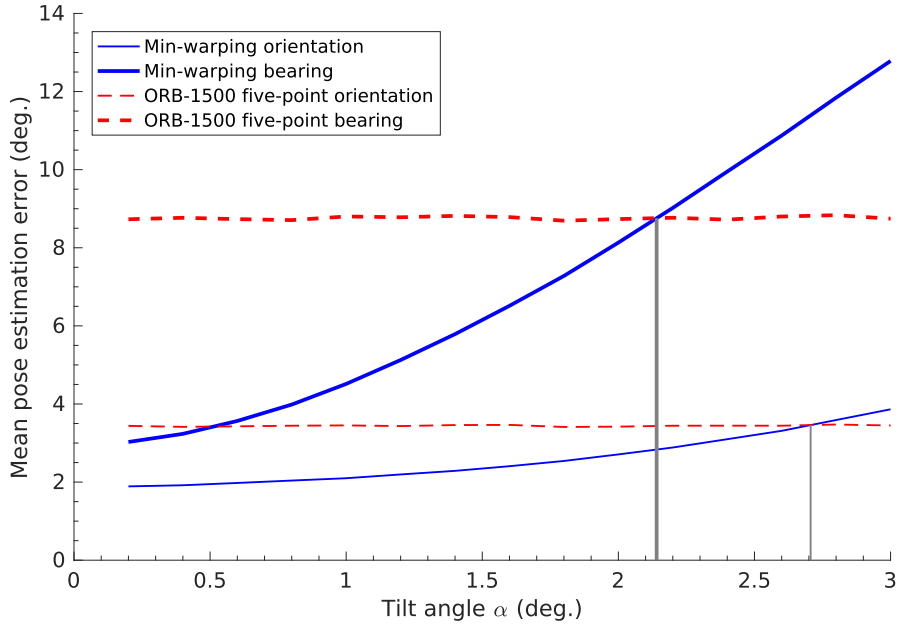


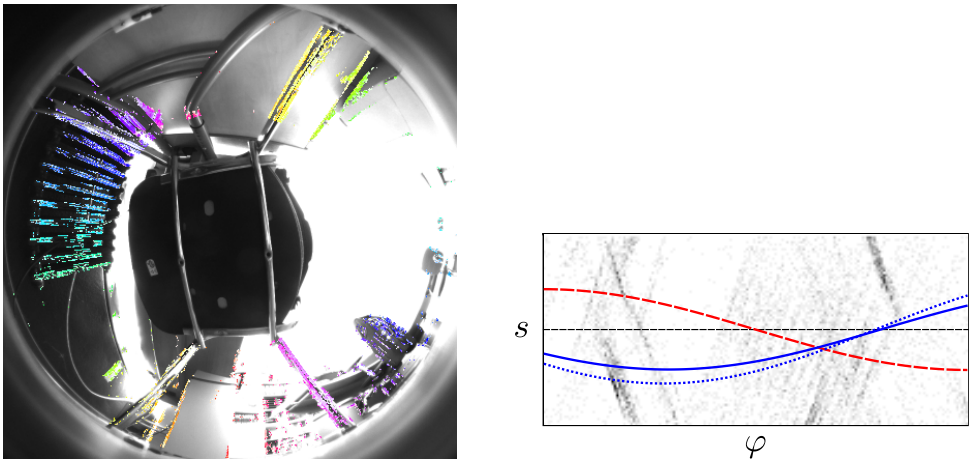
Figure 4.25.: The effect of the tilt angle α on the orientation and bearing error in visual pose estimation. For the planar-motion min-warping method [102], the pose-estimation errors (blue lines) increase with the tilt angle. In contrast, the red lines show constant errors for the nonplanar five-point method [138] with local visual features [129]. Gray lines mark the tilt angle beyond which the planar-motion error exceeds the nonplanar one; this occurs at about $\alpha > 2°$. This figure is based on Figure 3.18, which contains additional details.

vertical elements in the environment. In some locations, such elements are rare, or are drowned out by many near-vertical elements. As an example, Figure 4.18c shows the robot surrounded by a chair and table with angled legs. For the image-space method, this produces the worst tilt-estimation error out of all locations: Using the parameters from Table 4.3, we find $\epsilon = 7.4°$ for the reject-refit scheme, and $\epsilon = 7.7°$ for RANSAC. The reason for this poor estimate is illustrated in Figure 4.26: Most of the edge pixels extracted from the camera image do not correspond to vertical elements. The (φ, s) curve fitted to these incorrect edge pixels thus gives a poor tilt estimate. Conversely, the curve for the correct tilt parameters (α_T, β_T) matches few of the edge pixels.

While individual locations may produce high tilt-estimation errors, no environment causes a general failure for any method. Figure 4.22 illustrates this by showing $\bar{\epsilon}$ for each method and environment. Although there is some variation in $\bar{\epsilon}$, it remains below $2°$ in all cases.

In this discussion, we have frequently used the mean tilt-estimation error $\bar{\epsilon}$. This measure depends on the composition of the image set on which it was calculated. For example, a method may give especially high ϵ for images with a high α_T. In this case, a set with many such images gives a higher mean $\bar{\epsilon}$, compared to a set with few such images. We therefore calculated $\bar{\epsilon}$ for each true tilt angle α_T. In Figure 4.21, $\bar{\epsilon}$ shows only a moderate dependence on the tilt angle α_T. Note that tilt-estimation errors also occur for an untilted robot ($\alpha_T = 0°$).

In Section 4.2.4, we included a corrected vector-consensus variant that uses the correction factor a' from Equation (4.45). We first introduced a similar factor a (Equation (4.30)) to compensate for the approximations made in the image-space method. Figure 4.27 plots the true and estimated tilt angle with and without these corrections. We note that, on average, the uncorrected methods overestimate α for all true tilt angles α_T. However, the corrected methods merely divide α by a finite constant. Thus, they cannot correct the difference between α and α_T for $\alpha_T = 0$. In future experiments, it might be worthwhile to include an additive correction with $\alpha = l/a + b$ and



(a) Camera image **(b)** Edge pixel histogram

Figure 4.26.: (**a**) shows a forward-tilted image ($\alpha_T = 4.15°, \beta_T = 0°$) captured at the location in Figure 4.18c. As in Figure 4.9, the prefiltered edge pixels are highlighted in color. Few of these edge pixels correspond to vertical elements, while incorrect edge pixels are common. Similar to Figure 4.10, we visualize the parameters (φ_k, s_k) of the edge pixels in (**b**). Based on the ground truth, a dashed red line shows the expected relationship between φ and s for edge pixels from vertical elements. Unlike in Figure 4.10, few of the edge pixels actually lie close to this line. The solid blue line shows the (φ, s) curve for the incorrect (α, β) estimated by the reject-refit scheme. While it is a poor tilt estimate, this curve is a better fit for the (φ_k, s_k) in the histogram (**b**). Thus, the poor tilt estimate is likely caused by the incorrect edge pixels. The RANSAC-based estimate suffers from a similar error, as illustrated by the dotted blue line.

$\alpha = \alpha'/a' + b'$.

As shown in Figures 4.23 and 4.24, the tilt-estimation time noticeably depends on the parameters being used. Choosing parameters with a somewhat higher mean error $\bar{\epsilon}$ can reduce the mean execution time by more than half. Under such a trade-off, all methods achieve a mean execution time below $30\,\mathrm{ms}$ on our robot's embedded CPU (Figure 4.24). On the desktop system, this time is less than $5\,\mathrm{ms}$. However the time required for some images can be considerably higher, as seen in the 95th percentile values from Table 4.4. This must be taken into account for systems with strict real-time constraints. For our cleaning robot, one onboard relative-pose estimation process requires $\approx$165 ms (Table 3.10). In comparison, the reject-refit image-space method requires only $\approx 16\,\mathrm{ms}$ to estimate the tilt (Table 4.4). Adding tilt estimation to the pose-estimation process thus increases the total time by only $\approx 10\,\%$. Overall, we deem all methods suitable for onboard use on a domestic robot such as our own.

The utility of tilt corrections for planar-motion methods depends on the prevalence and magnitude of the tilts encountered in a given environment. Corrections can be beneficial if the robot is frequently tilted, or if uncorrected tilts have a large effect. In other cases, the tilt-estimation errors may cause greater problems than the actual tilts. Such errors occur even if the robot's motion is perfectly planar. The use of tilt correction thus needs to be evaluated for each application.

Specifically, we consider the visual relative-pose estimation used by our cleaning robot: In Section 3.3.3, the nonplanar five-point method [138] gave more accurate results for tilts over $\approx 2°$, as shown in Figure 4.25. However for the image-space method, the residual error ϵ will usually remain below this value (Figure 4.20). The combined execution time for tilt estimation and planar pose estimation is $\approx 16\,\mathrm{ms} + 165\,\mathrm{ms}$. This is much faster than the fastest nonplanar pose-estimation time of $\approx 1807\,\mathrm{ms}$ listed in Table 3.10. Thus, tilt estimation should help preserve the advantages

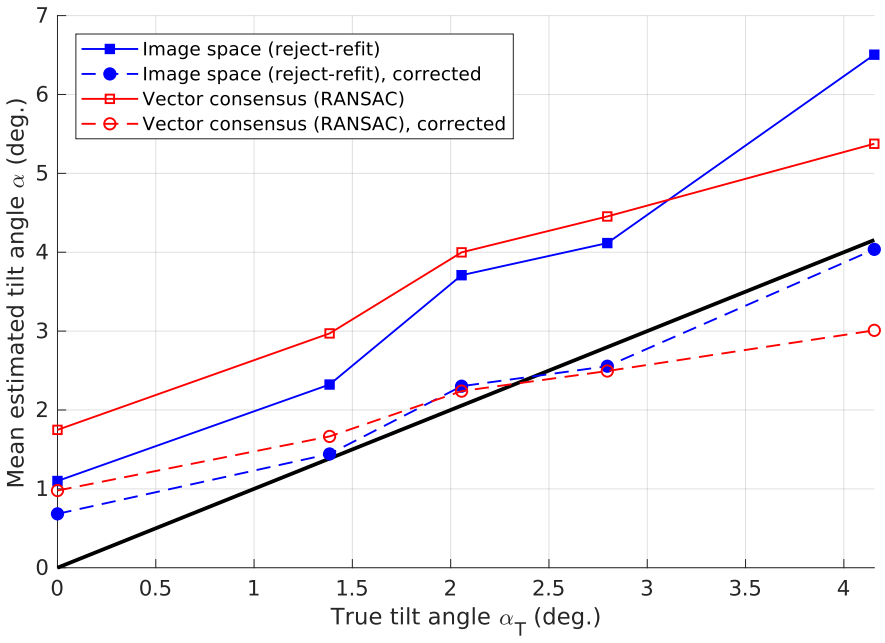


Figure 4.27.: The true and estimated tilt angle for corrected and uncorrected estimates. For each method, we used the parameters (Table 4.3) that minimize $\bar{\epsilon}$ after correction. This figure was generated using the factors $a = a_L$ and $a' = a'_L$ for each image location L, which we determined according to Section 4.2.4. For the uncorrected image-space method, we calculated α using Equation (4.26). A black line represents a perfect match between α_T and α.

of the planar-motion method if the robot is tilted. However, this is limited to small tilt angles and requires visually distinct vertical elements—limitations not shared by the nonplanar method.

4.5. Conclusions

In this chapter, we sought to measure a robot's tilt relative to a movement plane in an domestic environment. All methods tested here solve this problem based on panoramic images (Table 4.3). They do so across different environments (Figure 4.22) and tilt angles (Figure 4.21). Their fast execution time makes them suitable for real-time use, even on our robot's modest embedded CPU (Table 4.4 and Figure 4.24). Although average errors are low, tilt-estimation failures can occasionally occur (Figure 4.20). Such failures are likely if the environment lacks visually-distinct vertical elements (Figure 4.26). Furthermore, even an untilted robot will experience some tilt-estimation errors (Figure 4.21).

Overall, the image-space reject-refit variant had the lowest estimation error (Table 4.3) and execution time (Table 4.4 and Figure 4.24). A variant that uses RANSAC to reject incorrect edge pixels offered no advantage in quality or speed. The vector-consensus method was also slower, and resulted in higher errors. However, using the correction factor in Equation (4.45) reduced these errors. The vector-consensus method makes fewer approximations than the image-space method. It may thus still be useful in other applications where these approximations are invalid.

4.6. Outlook

In light of these results, we note several possibilities for future improvements: In this chapter, we exploit the visually-distinct vertical elements that appear in a typical domestic environment. If these

elements are rare or drowned out by near-vertical ones, large errors ϵ may ensure (Figure 4.26). A confidence measure would be useful to detect these incorrect results. Such a measure might be based on the fraction of incorrect edge pixels rejected during tilt estimation.

The methods in this study use only a basic RANSAC scheme. However, the literature contains numerous advanced RANSAC variants, which may achieve better results [148, 24, 115, 147, 124]. We currently use a very simple heuristic to correct the tilt-angle estimate α (Equations (4.30) and (4.45)). However, as seen in Figure 4.27, this is not sufficient to fully correct the error in α. A more sophisticated approach may instead calculate $P(\hat{\alpha}|l)$. This is the probability that the robot is tilted by $\hat{\alpha}$ given the vanishing-point shift l. We could use Bayesian techniques to evaluate $P(\hat{\alpha}|l) = P(l|\hat{\alpha})P(\hat{\alpha})P(l)^{-1}$. This also lets us incorporate the tilt-angle distribution $P(\hat{\alpha})$ for a specific environment. However, finding the parameters for such a probabilistic model may require a large amount of training data.

The implementations used in this chapter are not fully optimized. For example, a greater use of single instruction multiple data (SIMD) instructions would likely improve the execution speed. Such instructions are supported on both the desktop and embedded processors. However, we feel the speed of our implementations is adequate, and did not attempt such improvements here.

In this work, we focused on simple methods that leverage the specific properties of the tilt-estimation problem. As discussed in Section 4.1.1, other works present more general solutions to estimate camera orientations. We believe that a comprehensive comparison between these two approaches might be useful do determine their relative strengths and weaknesses.

5. Human-Like Room Segmentation in Domestic Environments

Our cleaning robot, and others like it, create maps of their surroundings for use in planning and navigation. In typical domestic indoor environments, such maps contain rooms connected by passageways. Segmenting a map into these rooms has several uses, such as hierarchical planning of cleaning runs or the definition of cleaning plans by a user. Especially in the latter application, the resulting room segmentation should match the human understanding of rooms. In this chapter, we present a method that solves this problem for our robot's topo-metric map (Section 2.3). First, a classifier identifies those map-graph edges that cross a border between rooms. This classifier utilizes data from multiple robot sensors, such as obstacle measurements and camera images. Next, we attempt to segment the map at these room-border edges using graph clustering. By training the classifier on user-annotated data, this produces a human-like room segmentation. We optimize and test our method on numerous realistic maps generated by our cleaning-robot prototype and its simulated version. Overall, we find that our solution produces human-like room segmentations in complex domestic environments. These results also surpass those achieved through a mere clustering of the map graph. However, unusual room borders that differ from the training data remain a challenge. Note that this chapter is based on an earlier publication by the author [46].

5.1. Introduction

To enable navigation and planning, an autonomous robot may build a map of the environment. For a domestic cleaning robot like ours, this map usually consists of rooms interconnected by passageways. Segmenting this map into its component rooms has multiple uses, including the following: First, the robot can refer to rooms when communicating with humans [6, 137]. A user may give instructions that reference rooms, such as "Robot, move to the kitchen".[1] Second, room segmentation can assist in place categorization by integrating information [107, 121]: For example, camera images captured at many points within the same room may be combined in an attempt to categorize the room. Third, room segmentation commonly plays a role in semantic mapping and multi-level planning (survey: [81]); for our cleaning robot, hierarchical cleaning and user-defined cleaning plans are of special interest. However, identifying this room structure is a nontrivial problem, in part due to ambiguous passageway- and room-like elements within the map.

In this chapter, we present a method for human-like room segmentation using the topo-metric map from Section 2.3.1. Specifically, we seek to label the map nodes so that each set of identically-labeled nodes represents a single room. Ideally, the resulting rooms should reproduce the judgment of a human observer. In brief, our method accomplishes this by performing four major steps: First, we preprocess the topo-metric map generated by our cleaning-robot, preparing it for segmentation. Second, we use the robot's sensor data to calculate a feature vector for every edge in the map graph. Features are based solely on the immediate vicinity of an edge, and thus require no global map consistency. In a third step, a classifier uses these features to estimate whether or not a map edge crosses a room border. Finally, we apply a graph-clustering step to segment the map graph into rooms, taking into account the room borders identified in the previous step.

[1] This also requires room labeling, a step which we do not consider here.

The rest of this chapter is structured as follows: We begin by discussing related works in Section 5.1.1, which we then compare to our own effort in Section 5.1.2. Next, Section 5.2 describes our method in detail, elaborating on the four steps listed above. Subsequently, we test our method across numerous environments using several experiments, as reported in Section 5.3. In Section 5.4, we discuss the results based on numerical quality measures, as well as examples of room-segmentation results. Finally, Section 5.5 contains our conclusions, together with an outlook on possible future developments.

5.1.1. Related Works

In the literature, there are several works addressing the problem of room segmentation within the context of mobile robots. For this overview, we are especially interested in solutions which overlap with the one we propose in Section 5.2. Here, we distinguish between two different approaches to room segmentation: Those from the first category perform place categorization, assigning labels such as *office* or *kitchen* to areas within the map. Such methods go beyond simple room segmentation, constructing semantic maps instead. However, the general problem of semantic mapping lies beyond the scope of this room-segmentation study. For a broader overview of semantic mapping for mobile robotics, we point to the survey by Kostavelis and Gasteratos [81]. Here, we focus on those semantic mapping works for which room segmentation is a central aspect. Conversely, members of the second category merely determine which map locations lie within the same room. They do not perform place categorization, and thus do not require information about potential place types.

5.1.1.1. Place Categorization

Methods from the first category commonly use a bottom-up approach: Here, a classifier determines which type of room surrounds a given place, based on sensor data the robot recorded at that point. For example, Mozos et al. [107] distinguish *corridors*, *rooms*, and *doorways* by applying a boosting classifier to features extracted from laser-range scans. The authors apply this scheme to simulated scans generated from an occupancy grid map to classify the map's cells. Connected cells with the same label are then joined together into regions, resulting in a room segmentation.

Friedman, Pasula, and Fox [54] introduce Voronoi random fields to segment occupancy grid maps. The authors extract a Voronoi graph from the map, and then use conditional random fields (CRF) to assign labels such as *hallway*, *room*, *doorway*, or *junction* to each node. These labels are chosen based on the obstacles in the vicinity of each node, as well as the information encoded in the Voronoi graph. Grouping contiguous nodes with the same label then segments the map. Shi, Kodagoda, and Dissanayake [136] combine CRF with support vector machines (SVM) to label the nodes of a generalized Voronoi graph based on simulated laser scans. Both the Voronoi graph and the laser scans were generated from occupancy grid maps. Here, the place types are more specific to the environment, for example *cubicle*, *kitchen*, or *printer room*.

Pronobis et al. [123] combine range scans with global [86] and local visual features [89] extracted from camera images. The authors apply separate classifiers to these features, using one multi-class support vector classifier (SVC) for each of the three feature types. A final SVC combines these feature-specific results into a single place label. These labels are comparatively fine-grained, such as *meeting room*, *office*, or *corridor*. The authors then accumulate results from close-by locations to label entire areas, producing a room segmentation. In a subsequent work [122], place types are defined by their properties. These include a place's geometric shape and size, as well as the types of nearby objects detected with a camera.

Ranganathan and Lim [126] utilize image sequences captured by a robot to label the cells of a grid representation. They use the place labeling through image sequence segmentation (PLISS)

system [125] to determine the probability that an image in a sequence depicts a certain type of place. In a novel approach, the authors then update the probabilities of those grid-map cells visible in the image, instead of the cell at which the image was taken. Occasional misclassifications are smoothed out by applying conditional random fields to the map. This work also uses fairly specific place labels, such as *lab* or *printer room*.

Some techniques use a room-segmentation heuristic as a preprocessing step for semantic mapping. Zender et al. [159] apply the classifier from Mozos et al. [107] to a robot's navigation graph, the nodes of which represent locations visited by the robot. Each node is classified as *corridor*, *room*, or *doorway* based on a laser scan taken at the corresponding location. *Doorways* are identified by a detector, which is triggered if the robot passes through an opening with the width of a typical door frame. The graph is then segmented into areas of connected *room* or *corridor* nodes, separated by *doorway* nodes. Hawes et al. [66] extend this scheme by introducing non-monotonic reasoning. This lets the robot incorporate previously undetected doorways while moving through the environment. According to the authors, this also counteracts problems caused by occasional failures of the doorway detector. Similarly, the cognitive mapping system by Vasudevan et al. [152] uses a door-detection heuristic to segment an environment based on obstacle data. Note that it could be argued that these works belong to the second category, since their place-categorization results do not influence the room segmentation.

5.1.1.2. Room Segmentation

Methods from the second category perform room segmentation without place categorization. Several of these identify rooms by applying heuristics to occupancy grid maps. A survey and analysis by Bormann et al. [19] compares three such methods, in addition to the place-categorization approach by Mozos et al. [107].

First, morphological segmentation [18] repeatedly applies an erosion operator to an occupancy grid. The resulting expansion of the walls eventually separates areas from the remainder of the map's unoccupied space. Such an area is labeled as a room if its size lies within a certain range. Any unlabeled grid cells are added to the nearest room through wavefront propagation.

Second, the distance-transform method [19] calculates the distance between each unoccupied grid cell and the nearest obstacle. Disregarding all cells with a distance below a certain threshold leads to a number of disconnected areas. A search identifies the threshold that maximizes the number of these areas, each of which then forms a room. As in the morphological segmentation, the remaining unlabeled cells are assigned to the nearest room.

Third, rooms can also be segmented using a Voronoi graph extracted from the occupancy grid. This graph consists of all map cells for which the two nearest obstacles are equidistant. Thrun [144] segments the Voronoi graph by first identifying its critical points. These are points where the distance to the nearest obstacle reaches a local minimum. Connecting each critical point with its two nearest obstacles gives the so-called critical lines. The occupancy grid map is then segmented by splitting it along these critical lines of the Voronoi graph. However, the resulting segments are usually too fine-grained, and have to be merged into actual rooms. This can be accomplished through a size-based heuristic [19].

In contrast to these deterministic heuristics, Liu and Wichert [87] present a probabilistic approach to room segmentation. Given an occupancy grid map M, they calculate the posterior probability $P(W|M)$ for a world W. The authors assume that W consists entirely of rectangular rooms bounded by four straight walls and connected by doors. After thus limiting the space of possible worlds, a Markov chain Monte Carlo technique searches for the world W^* that maximizes $P(W^*|M)$. The best candidate found by this search serves as the room-segmentation result.

Zivkovic, Booij, and Kröse [160] perform room segmentation without using a map, instead requiring only unordered image sets. These images are first assembled into a graph, with each node

representing one image. Edges are added based on the images' local visual features [89]: Using these features, the method estimates the relative direction and orientation between the locations of each image pair. If this estimate is judged plausible, an edge is inserted between the two corresponding graph nodes. Finally, spectral clustering is applied to the graph, with the nodes of each cluster corresponding to a room.

5.1.2. Our Contribution

The room-segmentation method we propose in Section 5.2 offers three main features: First, our method works with our robot's dense, topo-metric map; it does not require global metric map consistency. Second, the method utilizes a variety of edge features, derived from several different sensors. It is not intrinsically restricted to any specific sensor or feature of the environment. Third, we learn to detect room borders from human-annotated training data. This lets our solution produce more human-like room segmentations. Novel types of environments or edge features can also be integrated through re-training, without modifying the core method.

Compared to the existing methods, our approach occupies a niche between the two categories from Section 5.1.1: Here, the members of the first category all employ place categorization. While this is useful for building semantic maps, it is not strictly necessary for room segmentation. Such methods have to be provided with place categories, and have to learn their characteristics from training data. This requires a substantial effort, especially if these categories are fine-grained. Additionally, it is assumed that the environment contains only these types of places. Methods from the second category do not require this kind of knowledge. However, the schemes discussed here also do not learn from human-annotated training data. Since we desire a human-like room segmentation, such a functionality would be very useful. In contrast, our method learns room segmentation from human-annotated maps, yet without the added complexities of a general place-categorization scheme. We believe that this approach to room segmentation thus combines advantages from both categories.

Based on our description in Chapter 2, we note that our cleaning robot imposes several platform-specific requirements: Recall that our robot produces topo-metric maps without global metric consistency (Section 2.3.1). We thus cannot use techniques that rely on globally-consistent grid maps. As we discussed in Section 2.3.2, our robot's obstacle data is comparatively sparse and short-ranged. This may pose a problem for other methods that require real or simulated laser scans. We also seek to use our robot's panoramic camera images to aid in room segmentation. In Section 5.4.2, we show that utilizing these images improves our own results considerably. Methods that can incorporate image data may therefore be especially suitable for camera-equipped robots such as ours. We have taken these factors into account while developing our method, ensuring that it meets the requirements imposed by a robot such as our prototype.

5.2. Our Room-Segmentation Method

In this chapter, we tackle the room-segmentation problem within the context of our cleaning robot. However, our method should also be adaptable to other, similar domestic robots. To solve the room-segmentation problem for the topo-metric map from Section 2.3.1, we assign a room label to each node. Nodes with the same label should be part of the same room, and each room should only contain nodes with the same label.

Our solution follows the general procedure depicted in Figure 5.1. After preprocessing the map (Section 5.2.1), we employ supervised machine learning to identify those map edges that cross room borders. First, we build a feature vector for each map edge based on sensor data recorded in its vicinity, as well as the map information (Section 5.2.2). A support vector machine (SVM) [27,

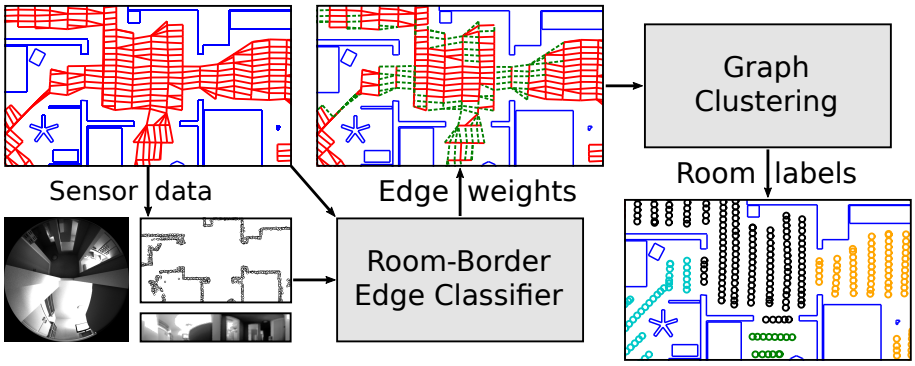


Figure 5.1.: An overview of the main components that make up our method. First, we construct a feature vector for every edge within the preprocessed map graph (**top left**, Section 5.2.1). These edge features are based on obstacle information and panoramic images (**bottom left**, Section 5.2.2), as well as information from the map. From these features, a classifier determines which edges cross a room border (**center**, room-border edges in green, Section 5.2.3). We then perform room segmentation through graph clustering, taking into account the edge classification result (**bottom right**, Section 5.2.4).

22] classifier then identifies room-border edges based on their feature vectors (Section 5.2.3). In order to learn human criteria for room borders, we train the classifier on human-annotated training data. However, simply segmenting the map at these room-border edges would make our method vulnerable to misclassified edges.

Instead we perform graph clustering, which identifies clusters of tightly-connected map nodes (Section 5.2.4). Each of these clusters is assigned a label, which in turn becomes the room label of the nodes within that cluster. Here, we use a spectral clustering algorithm which minimizes the normalized cut [91, 135]. We then encourage spectral clustering to cut the identified room-border edges by assigning them a lower weight. This makes it more likely that minimizing the normalized cut results in a human-like room segmentation. Since graph clustering attempts to optimize the segmentation across the entire map graph, the result is more robust against the effects of occasional misclassified edges. Note that we must specify the number of clusters before performing spectral clustering. In this study, we generally assume that this room count is known, for example by querying a human user. However, in Section 5.2.4.1 we also try to estimate this number from the map itself.

5.2.1. Map Preprocessing

The topo-metric map generated by our robot (Section 2.3.1) is primarily used for navigation and coverage planning. To make the map graph more suitable for room segmentation, we apply several preprocessing steps: First, we reduce the computational cost of our method by removing superfluous edges from the map graph. Second, we attempt to lessen the influence of the map's part-lane structure on the room-segmentation results.

Within our map, all adjacent and reachable node pairs are connected by edges. As seen in Figure 5.2, this results in a large number of edges, greatly increasing the overall processing time. Specifically, SVM training becomes prohibitively expensive if the number of training edges grows too large. Most of the edges are tightly-packed diagonals between nodes from neighboring lanes. Since they are fairly similar, we expect that many of these edges are not needed to solve the room-segmentation problem. Instead, each node should only be connected to its closest neighbor on each adjacent lane, as determined by the estimated node distance d. We then delete the superfluous edges using the heuristic from Algorithm 1.

Algorithm 1 : The heuristic used to remove unnecessary edges during map-graph preprocessing.

for each node $n \in$ map nodes N **do**
  $L \leftarrow \{m | m \in N \wedge \mathrm{part}(m) = \mathrm{part}(n) \wedge \mathrm{lane}(m) = (\mathrm{lane}(n) - 1)\}$
  $k \leftarrow \arg\min_{k' \in L} d(k', n)$
  for edge e between n and $L \setminus \{k\}$ **do**
    delete $\mathrm{edge}(l, n)$
  end for
  for each part p older than $\mathrm{part}(n)$ **do**
    $P \leftarrow \{m | m \in N \wedge \mathrm{part}(m) = p\}$
    $q \leftarrow \arg\min_{q' \in P} d(q', n)$
    for edge f between n and $P \setminus \{q\}$ **do**
      delete f
    end for
  end for
end for

Here, $\mathrm{part}(n)$ and $\mathrm{lane}(n)$ are the index of the part and lane to which the node n belongs. Thus, each node will have at most one edge connecting it to the previous lane, and at most one edge to each of the previously created parts. Basically, we keep those edges with the minimal spatial distance d. However, a node can still be connected to two or more other nodes from the same lane or part. This can occur if the node itself is the nearest neighbor of more than one node within a subsequent lane or part. As an example, Figure 5.3 shows the graph from Figure 5.2 after deleting the superfluous edges.

Our second preprocessing step reduces the influence of the map's part-lane structure on the room-segmentation result. As outlined in Section 5.1.2, we segment maps using the normalized-cut criterion. In our case, this criterion depends on the map edges cut by a room border, as explained in Section 5.2.4. When cutting the map graph along a line, the cost (here: the resulting increase in the normalized-cut criterion) should not depend on the line's orientation relative to the part-lane structure. Such an orientation-dependence could cause incorrect room segmentations: The robot traverses different passageways within the same map at varying orientations. If the cost of a linear cut strongly depends on this orientation, it may change the graph-clustering result. Such behavior is undesirable: if the underlying passageways are similar, they should be treated as such. The problem is exacerbated whenever the edge classification is unreliable. In that case, the classification-based edge weighting cannot reliably compensate for the orientation-based difference in cost.

Figure 5.4 demonstrates that the number of edges cut by a room border depends partly on the orientation of the lanes. This is due to the difference in node and lane spacing: While the nodes on each lane are placed approximately $10\,\mathrm{cm}$ apart, the distance between adjacent lanes is $\approx 30\,\mathrm{cm}$. Thus, a cut that runs parallel to the lanes crosses an edge every $\approx 10\,\mathrm{cm}$. Conversely, edges are cut at $\approx 30\,\mathrm{cm}$ intervals when cutting orthogonal to the lanes. For example, the number of cut edges in Figure 5.4a is greater than in Figure 5.4b, depending on whether the robot drove lanes that are parallel or orthogonal to the passageway.

We reduce this effect of the lane orientations by adjusting the edge weights. Here, we divide each edge's weight by the estimated distance between the two nodes. This way, the costs of parallel and orthogonal cuts of equal length become approximately identical. Unfortunately, the cost per

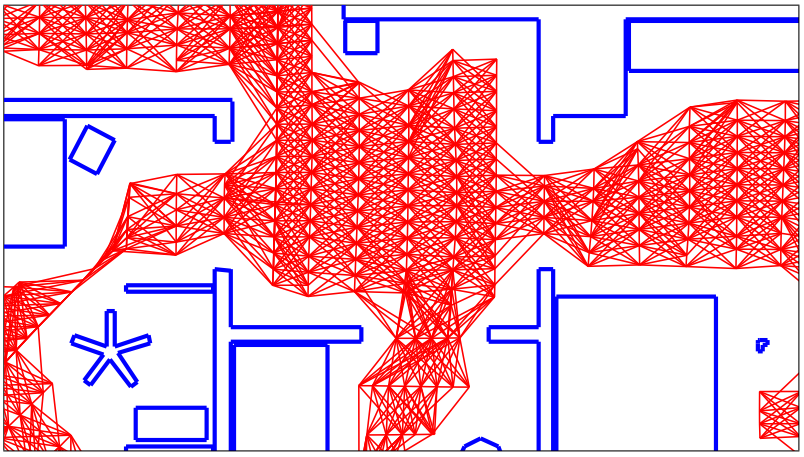


Figure 5.2.: A segment of the robot's topo-metric map graph, as used for navigation and planning. The red lines represent edges in the graph, each of which connects two nodes. For the sake of clarity, we do not show the actual nodes here. As seen in this illustration, there are many overlapping edges between the meandering lanes. The blue lines show the outlines of obstacles, such as walls and furniture. This figure is based on Figure 2.3, which we generated from a simulated cleaning run.

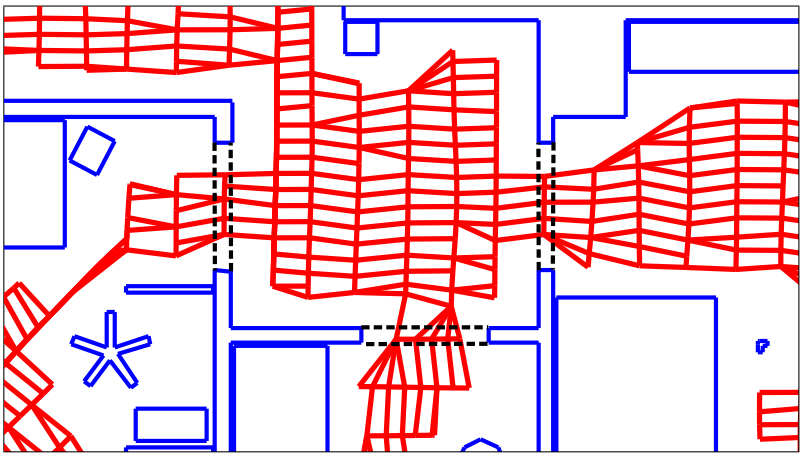
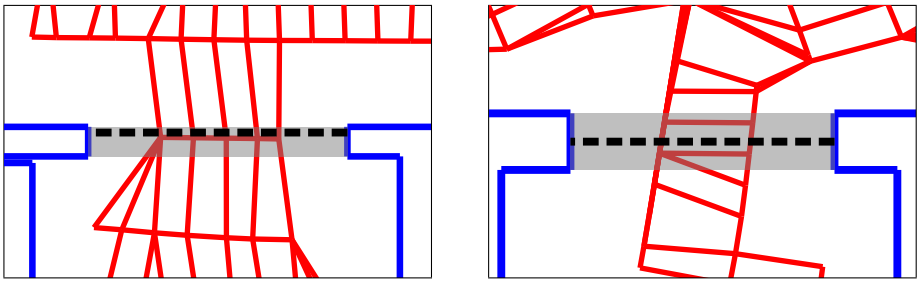


Figure 5.3.: The map graph from Figure 5.2 after deleting superfluous edges using Algorithm 1. This kind of comparatively sparse graph is used by all subsequent steps in our method. The black dashed lines show the possible room borders drawn by a human operator.



(a) Lanes parallel to passageway **(b)** Lanes orthogonal to passageway

Figure 5.4.: Two examples of the robot's map graph at a narrow passageway, shown in the style of Figure 5.2. Both (**a**) and (**b**) each contain a passageway between two rooms, which is marked in gray. The hypothetical room borders within these passageways are indicated by dashed lines. Both figures use the same scale, and both passageways are approximately identical in size. However, the room border in (**b**) intersects two edges, compared to five in (**a**). We compensate for this difference by adjusting the graph's edge weights. Note that (**a**) corresponds to the center-right passageway in Figure 5.3, rotated clockwise.

distance for diagonal cuts remains higher by a factor of $\approx \sqrt{2}$. Since this cannot easily be resolved by preadjusting the edge weights, we choose to accept this remaining anisotropy.

5.2.2. Map-Edge Features

We now need to identify those map-graph edges which cross a border between two rooms. To solve this classification problem, we first annotate each map edge with a feature vector. These feature vectors consist of individual scalar edge features, which are calculated from information acquired in the vicinity of the edge. Specifically, we use the length of an edge (Section 5.2.2.1), local obstacle data (Section 5.2.2.2), a visual doorway detection (Section 5.2.2.3), and two image-distance measures (Section 5.2.2.4). For this study, we select edge features based on experience gained during preliminary experiments. However, a rating heuristic might be helpful for judging potential edge features. In Section 5.2.2.5, we therefore evaluate two metrics for the usefulness of edge features.

5.2.2.1. Edge Length

For our first edge feature, we use the metric edge length l. This is the Euclidean distance between the estimated positions of the edge's two map nodes. Since edges only connect nearby nodes, we can reliably calculate this distance without global metric map consistency. There are two reasons for including the edge-length feature: First, our maps contain similar numbers of short and long edges, as shown in Table 5.1. This is a side-effect of our robot's cleaning strategy and the map preprocessing from Section 5.2.1. However, according to Table 5.1, the majority of room-border edges are long. Consequently, the edge length l itself carries information which is useful for room-border detection. Second, some of the other edge features strongly correlate with the edge length. This is most noticeable for the image-distance features described in Section 5.2.2.4. By knowing l, the classifier may be able to distinguish this effect from the effect of a room border.

	$b = \textbf{border}$	$b = \overline{\textbf{border}}$	$P(l')$
$l' = \text{short}\ (l < 0.2\,\text{m})$	0.008	0.992	0.498
$l' = \text{long}\ (l \geq 0.2\,\text{m})$	0.043	0.957	0.502
$P(b)$	0.026	0.974	

Table 5.1.: The conditional probability $P(b|l')$ that a random short or long edge crosses a room border. We give these values for short and long edges, which have a length of $l < 0.2\,\text{m}$ and $l \geq 0.2\,\text{m}$, respectively. The column labeled $P(l')$ lists the overall fractions of short and long edges. Similarly, the row labeled $P(b)$ contains the fraction of room-border ($b = \text{border}$) and within-room ($b = \overline{\text{border}}$) edges. We calculated these values from the maps described in Section 5.3.1.

5.2.2.2. Obstacle Data

For typical domestic environments, we expect that room borders coincide with narrow passageways such as doors. These passageways are implicitly represented in the structure of the map graph: Rooms separated by a narrow passageway tend to be connected by fewer edges. Since our method attempts to minimize the normalized cut, it is thus more likely to create a room border at a narrow passageway.

However, this behavior may also pose a problem: Below a certain width L (as defined below), narrow passageways are actually less likely to correspond to room borders within our maps. Among the edges passing through a passageway with $L < 0.5\,\text{m}$, only 0.5% actually cross a room border. This is much lower than the room-border fraction for all edges, which Table 5.1 lists as 2.6%. Thus, placing a room border at such a passageway is less likely to be correct. Instead, these very narrow passageways tend to occur between furniture or similar obstacles. We therefore include the passageway width L as an edge feature, hoping to improve the classification of such edges.

We estimate L from the robot's obstacle map, as illustrated in Figure 5.5: For an edge between the nodes i and j, we first retrieve the nearby obstacle points for i and j as per Section 2.3.2. In practice, some of these points may be the result of incorrect obstacle measurements. This problem is uncommon, but may cause incorrect passageway-width estimates. Like in Section 2.3.2.1, we reject such points through density-based clustering using the DBSCAN algorithm [39]. DBSCAN identifies those obstacle points which are not part of a sufficiently large, dense cluster. Here, clusters of less than three points within a distance of $10\,\text{cm}$ are discarded as false measurements.

We also discard obstacles outside of a search area, which runs orthogonal to the edge direction. As shown in Figure 5.5, this area is somewhat wider than the length of the edge. We consider this necessary to avoid overlooking obstacles when calculating L for short edges. The width of the search area is equal to l, plus an extension of $12\,\text{cm}$ on either side. If the edge is short ($l < 20\,\text{cm}$) and connects subsequent nodes on a lane, we further extend each side by up to $5\,\text{cm}$. For edges that connect nodes on the same lane, this may not extend the search area beyond that lane's beginning or end. We now search this area for the closest obstacle point on both sides of the edge. Finally, the metric distance between these closest points serves as our passageway width L. Note that L is only an approximation of the true width of the passageway. Its accuracy depends on the geometry of the passageway, and on the position and orientation of the edge within it.

For some map edges, we may not have enough nearby obstacle points to compute the passageway width. To allow the room-border classifier to work with these edges, we substitute a fixed value for L instead. This value should be distinct from the L calculated from actual obstacle measurements. The naive approach would be to use a very large value, such as $L = \infty$. However, such a large value would cause problems with the edge-feature scaling discussed in Section 5.2.3. In our maps, the highest obstacle-derived value is $L \approx 4.4\,\text{m}$, and we thus use a default of $L = 5\,\text{m}$.

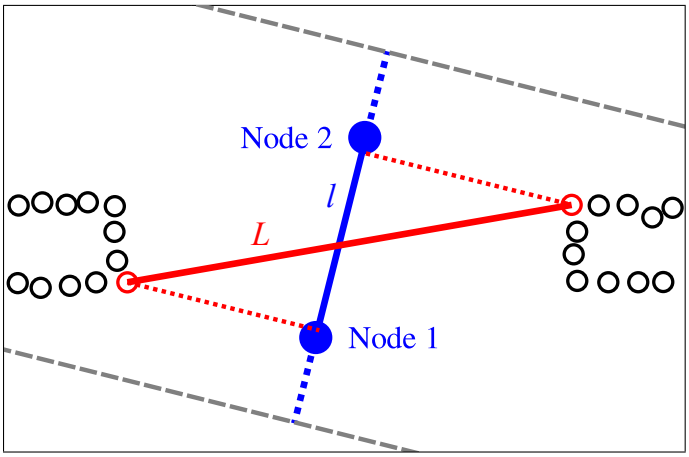


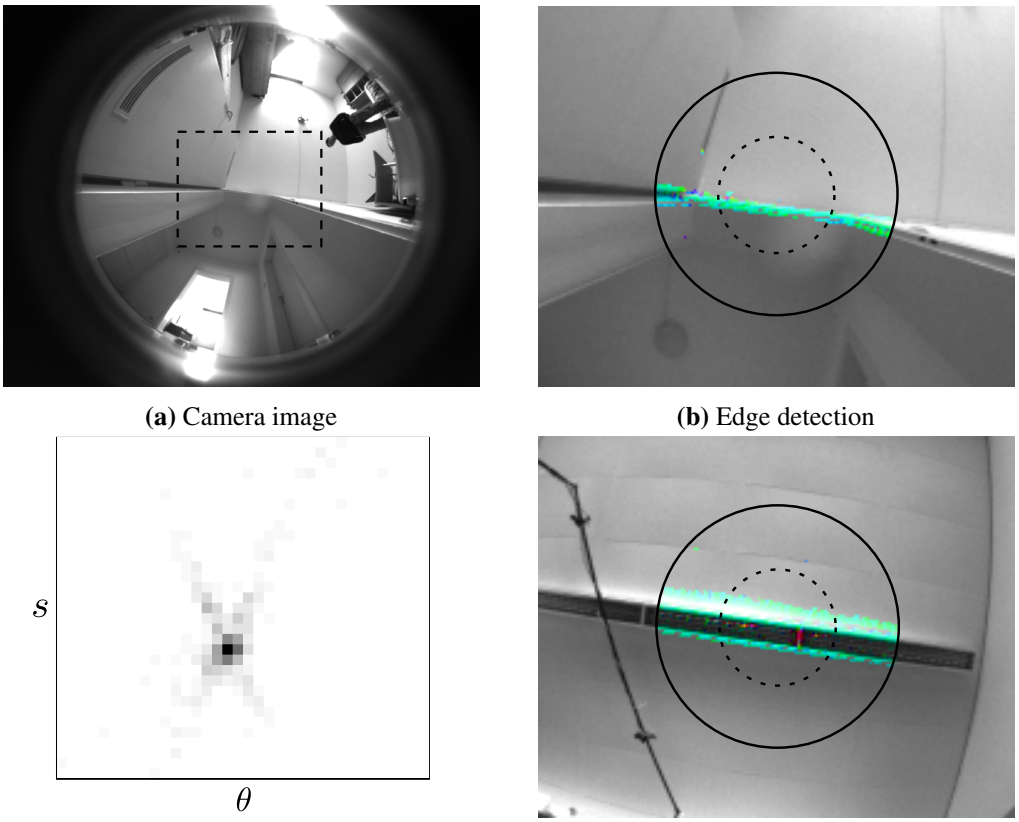
Figure 5.5.: An illustration of the passageway-width estimation. The blue dots represent the two nodes in the map graph; these nodes are connected by an edge (blue line) of length l. Nearby obstacle points are shown as black circles. A pair of dashed gray lines delimits the obstacle search area around the edge. Two red circles correspond to the two obstacle points which lie closest to the edge on each side, as shown by the dotted lines. The distance L (red line) between those two points is the approximate passageway width.

5.2.2.3. Visual Doorway Detection

In domestic environments, room borders often occur at visually distinct doorways or similar structural openings. We exploit this property by detecting such doorways in the images recorded by our robot's panoramic camera. In the literature, there are numerous methods for visually detecting doors, for example by Chen and Birchfield [23], Murillo et al. [110], and Yang and Tian [156]. These methods usually attempt to detect doors from afar, for example to guide a robot towards them. As we have discussed in Section 2.3, our map lacks global metric consistency. After detecting a distant door, we are thus unable to estimate its precise location within our map. Consequently, we also cannot determine which map edges cross through such a doorway. Instead, we use a simple heuristic to check for doorways in close proximity to each map node. To estimate whether a map edge crosses such a doorway, we then combine the results from the edge's two nodes.

Our method works by detecting image edges associated with doorways within the robot's camera image. These edges are often visually distinctive, as shown in Figure 5.6a. We note at least two approaches: One approach is based on the vertical posts on the sides, the other on the horizontal lintel at the top of the doorway. We found that vertical edges — such as from walls, window frames, or furniture — are quite common in our environments. During preliminary experiments, this frequently led to incorrect doorway detections. In comparison, non-doorway edges directly overhead the robot were less common. Additionally, detecting these edges does not require a panoramic camera. As demonstrated below, a ceiling-facing camera with a field-of-view as low as 38° could be sufficient. Although they are not immune to incorrect detections, we therefore focus on the overhead lintels. Egido et al. [38] previously employed an upward-facing sonar to detect these lintels with a mobile robot. Since we want to utilize our existing camera images, we instead use an edge histogram to detect straight image edges above the robot. This histogram technique is similar to the modification of the popular Hough transform (survey: [73]) presented by Davies [30], although our specific formulation differs.

Since we wish to detect lintels above the robot, we only consider a limited part of each camera image. However, we do not know the true dimensions of the doorways and lintels within a given environment. We therefore assume that a typical domestic doorway has a width of 80 cm and a



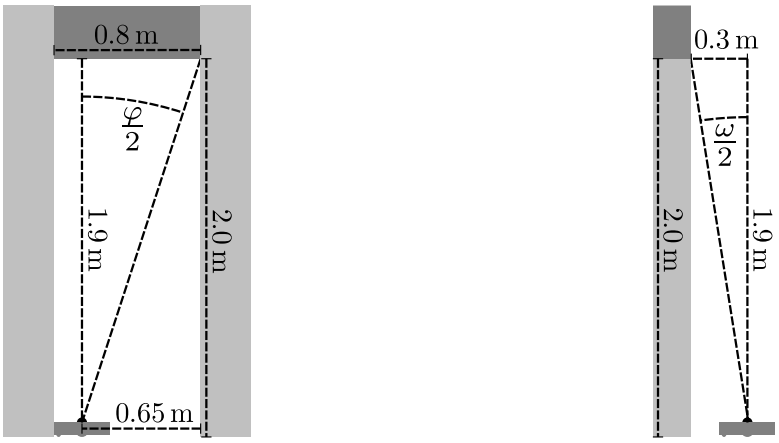
(a) Camera image

(b) Edge detection

(c) Edge histogram

(d) False positive

Figure 5.6.: An illustration of our visual doorway-detection heuristic. Doorway detection on the camera image from (**a**) identifies the edges shown in the detail (**b**); (**b**) corresponds to the dashed rectangle in (**a**). The solid outer ring represents the edge-search cone with an opening angle of $\varphi \approx 38°$. Similarly, the dashed ring corresponds to the edge-offset cone with an angle of $\omega \approx 18°$. Pixels identified as part of an image edge are highlighted in color. The hue indicates the orientation of each pixel's edge gradient. Next, we construct a histogram of the edge orientation θ and edge offset s from these pixels. The histogram in (**c**) shows a clear maximum near the center, which corresponds to the lintel. Unfortunately, this heuristic can be vulnerable to false positives. In (**d**), such a false positive is caused by a ceiling-mounted light fixture.



(a) Robot within doorway (frontal view)

(b) Robot next to doorway (side view)

Figure 5.7.: To determine the detection parameters φ and ω, we assume that a typical domestic doorway is $80\,\mathrm{cm}$ wide and $200\,\mathrm{cm}$ tall. Our robot is represented by the small shape near the bottom of each figure. The light and dark gray rectangles show the sides and lintel of the doorway, respectively.

height of $200\,\mathrm{cm}$. These dimensions are similar to those of real doorways we found in household and office environments. We now assume that our robot is located at one side of such a doorway, with the lintel directly above the robot's camera. As seen in Figure 5.7a, the distance between the camera and the furthest point of the lintel is $65\,\mathrm{cm}$ horizontally and $190\,\mathrm{cm}$ vertically. The entire lintel thus lies within a cone with an opening angle of $\varphi = 2\,\mathrm{atan}(\frac{65\,\mathrm{cm}}{190\,\mathrm{cm}}) \approx 38°$ above the camera. Using a calibrated camera model [132], we identify the camera pixels corresponding to this search cone. These pixels form the area represented by the solid circle in Figure 5.6b. To detect nearby lintels within the search cone, we thus search for edges within this image area.

As we explained in Section 2.2, our robot's camera uses a fisheye lens with an approximately equidistant projection. Since this projection is nonlinear, a straight edge in the world may appear curved in the camera image. However, recall that we limit our search to a small disc around the image center. Inside this disc, the projection is approximately linear, as shown in Figure 5.6a. We thus do not reproject the images, as we found that using the fisheye images gives adequate results.

To detect the edges, we apply a Scharr operator to the search area, which is similar to the well-known Sobel operator. However, the Scharr operator is specifically optimized for rotational invariance [74, 154]. This property is useful, as we wish to detect edges independent of their orientation within the image. In our experiments, we use the implementation from the OpenCV library [21]. We now know the horizontal and vertical edge gradients g_x and g_y for each pixel within the search area. From these values, we construct each edge pixel's gradient vector $\vec{v}$. For doorway detection, light-dark and dark-light edges should be treated equally. We therefore use a definition of $\vec{v}$ that remains unchanged if the pixel intensities are inverted:

$$\vec{v} = \begin{cases} (-g_x, -g_y)^T, & \text{if } g_y < 0 \vee (g_y = 0 \wedge g_x < 0), \\ (g_x, g_y)^T, & \text{otherwise.} \end{cases} \tag{5.1}$$

We also calculate each pixel's gradient intensity $I = \|\vec{v}\|$. For pixels with a low gradient intensity I, the comparatively strong camera noise leads to high uncertainty in $\vec{v}$. We therefore discard pixels for which I is lower than a threshold $I_{\min}$; this also reduces the overall processing time. Figure 5.6b shows the result of this step.

Next, we use a histogram to identify lintel edges from the individual edge pixels. The two axes of the histogram are the edge-gradient orientation

$$\theta = \mathrm{atan2}(v_2, v_1) \tag{5.2}$$

and the edge offset

$$s = \frac{\vec{v}^T}{\|\vec{v}\|}(\vec{p} - \vec{c}). \tag{5.3}$$

Here, $\vec{p}$ is the edge pixel position, $\vec{c}$ is the image center, and $\mathrm{atan2}$ is the quadrant-aware arctangent. Note that $\theta \in [0, \pi)$ due to the definition of $\vec{v}$ in Equation (5.1). We assign each pixel to the histogram bin (i, j), with

$$i = \left\lfloor \frac{\theta}{\Delta_\theta} \right\rfloor ;\ j = \left\lfloor \frac{s}{\Delta_s} \right\rfloor , \tag{5.4}$$

where Δ_θ and Δ_s are the bin widths. All pixels of a straight edge would share the same θ and s, and thus the same histogram bin. Consequently, a bin with a high number of pixels indicates that a straight edge is present in the image. Figure 5.6c demonstrates this through an example histogram.

The edge offset s represents the distance between an edge and the image center $\vec{c}$, as shown in Figure 5.8. Using the calibrated camera model, we use a $\vec{c}$ that corresponds to the camera's viewing direction. Since our robot's camera faces upwards, $\vec{c}$ also corresponds to a point directly above the robot. For a given map node, we want to ignore lintels that are unlikely to overlap any map edge connected to this node. In our map graphs, few edges are longer than $30\,\mathrm{cm}$. Thus, we try to exclude edge pixels from doorways more than $30\,\mathrm{cm}$ away. We do this by limiting the edge-pixel histogram to $s \in (-s_{\max}, s_{\max})$. To choose $s_{\max}$, we again assume a doorway that is $200\,\mathrm{cm} \times 80\,\mathrm{cm}$ in size. The geometry resulting from these assumptions is illustrated in Figure 5.7b: Here, the maximum distance between the camera and the lintel is $30\,\mathrm{cm}$ horizontally and $190\,\mathrm{cm}$ vertically. A lintel within this horizontal distance must intersect a cone above the camera with an opening angle of $\omega = 2\,\mathrm{atan}(\frac{30\,\mathrm{cm}}{190\,\mathrm{cm}}) \approx 18°$. From this value of ω, we then calculate $s_{\max} = 23\,\mathrm{pixel}$ using the calibrated camera model. Figure 5.6b demonstrates the effect of $s_{\max}$: The line of colored edge pixels clearly intersects the dotted inner circle, which corresponds to $s_{\max}$. Thus, the s of these pixels is less than $s_{\max}$, and they are added to the histogram. We also illustrate this in Figure 5.8.

We can now detect a straight image edge from the histogram: If the histogram's maximum value $\hat{H} = \max_{i,j} H_{i,j}$ is high, many edge pixels share a similar orientation and offset; we thus assume that a straight edge is present. Here, $H_{i,j}$ is the number of edge pixels in the histogram bin with index (i, j). Note that this method cannot distinguish one uninterrupted edge from multiple shorter edges with the same (θ, s). On one hand, this makes the method robust against interrupted edges. Such interruptions could occur due to occlusion, or due to discarded pixels with a gradient intensity I below $I_{\min}$. On the other hand, a large number of very short edges might cause a false doorway detection. For the purpose of this study, we are willing to accept this trade-off.

In practice, camera noise also causes noise in each pixel's θ and s. As a result, pixels from a single, straight edge might be spread across neighboring histogram bins. This could reduce the value of $\hat{H}$, causing a false negative detection. We therefore calculate three additional histograms, where θ and / or s are shifted by half a bin width: Here, pixels are assigned to the bins (i', j), (i, j'), or (i', j'), with

$$i' = \left\lfloor \frac{(\theta + \frac{1}{2}\Delta_\theta) \bmod \pi}{\Delta_\theta} \right\rfloor , \tag{5.5}$$

$$j' = \left\lfloor \frac{s}{\Delta_s} + \frac{1}{2} \right\rfloor , \tag{5.6}$$

$$a \bmod b = a - \left\lfloor \frac{a}{b} \right\rfloor b. \tag{5.7}$$

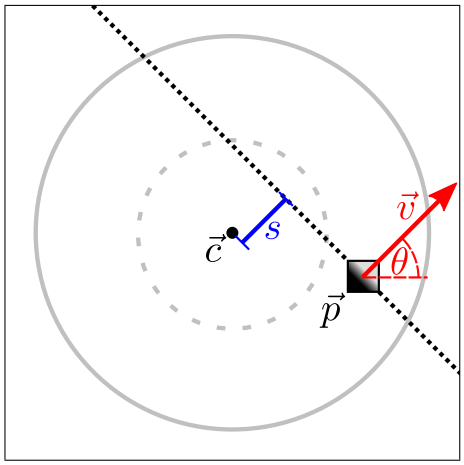


Figure 5.8.: The geometry of the edge pixels for doorway detection. The position $\vec{p}$ contains an edge pixel with gradient vector $\vec{v}$ and edge-gradient orientation θ. Mentally extending the edge from $\vec{p}$ results in the dotted black line. If the pixel at $\vec{p}$ is part of a straight lintel, the lintel would correspond to this line. The distance between the line and the image center $\vec{c}$ is the edge offset s. The gray circles correspond to the search cones shown in Figure 5.6b. If $\vec{p}$ lies within the outer search cone, and s is less than the radius of the inner search cone, this pixel will be added to the histogram.

We then search for the maximum $\hat{H}$ across all four histograms. This reduces the influence of the noise, as long as its effect on θ and s is smaller than the bin sizes.

Finally, we calculate the doorway edge feature $E_{k,l}$ for the edge between the map nodes k and l. We could simply use the minimum of the two per-node doorway-detection results $E_{k,l} = \min(\hat{H}_k, \hat{H}_l)$. Here, $\hat{H}_k$ is the doorway-detection result $\hat{H}$ for the map node with index k. However, this solution does not consider the orientation of the doorway relative to the edge. A doorway running approximately parallel to the edge (k, l) would still lead to a high $E_{k,l}$. This is undesirable, since $E_{k,l}$ should only be sensitive to doorways that intersect the edge (k, l).

To solve this problem, we calculate the edge orientation $\beta_{k,l}$ from the estimated node positions. We also calculate Θ_i, which is the gradient orientation θ for the histogram bin (i, j). Θ_i is perpendicular to the orientation of the image edge itself, as shown in Figure 5.8. Consequently, if

$$|\beta_{k,l} - \Theta_i| \bmod \pi \leq \epsilon, \tag{5.8}$$

then the image edge from the bin (i, j) is approximately perpendicular to the map edge (k, l). For a given edge orientation β, we therefore only consider bins (i, j) with

$$i \in I'_\beta = \left\{ \hat{i} | (|\beta - \Theta_{\hat{i}}| \bmod \pi) \leq \epsilon \right\}. \tag{5.9}$$

In other words, we only search for lintels which are nearly orthogonal ($\pm\epsilon$) to the given map edge. From this, we arrive at the angle-dependent edge feature $\tilde{E}_{k,l}$ with

$$\hat{H}_{k,\beta} = \max_j \max_{i \in I'_\beta} (H_k)_{i,j}, \tag{5.10}$$

$$\tilde{E}_{k,l} = \min(\hat{H}_{k,\beta_{k,l}}, \hat{H}_{l,\beta_{k,l}}). \tag{5.11}$$

Here, $(H_k)_{i,j}$ is the entry (i, j) from the histogram H_k of the node k.

Finally, we need to choose the parameters $I_{\min}$, Δ_θ, Δ_s, and ϵ. Unlike the cone angles φ and ω, we cannot easily estimate these parameters from the environment. Instead, we perform a search

Parameter	Values
I_{min}	100, **200**, 400
N_θ, N_s	(18, 17), **(36, 33)**
ϵ	$50°$, $55°$, $\mathbf{60°}$, $65°$, $70°$, $80°$, $90°$

Table 5.2.: Doorway-detection parameters tested during the search. The values with the best area-under-curve are printed in bold, and are used throughout the rest of this chapter. Note that the bin sizes Δ_θ and Δ_s are derived from the number of bins N_θ and N_s.

across a number of reasonable values, as listed in Table 5.2. Ideally, we could find the values that give the best overall room-segmentation result for our maps. This is not practical, however, because these results also depend on several other parameters; we elaborate on this in Section 5.2.3.3.

We instead optimize the doorway-detection parameters in isolation, using a criterion further discussed in Section 5.2.2.5: First, we identify room-border edges by merely applying a threshold to the edge feature $\tilde{E}$. Second, we construct the receiver operating characteristic (ROC) [42] curve for this simple classifier. Finally, we select the I_{min}, Δ_θ, Δ_s, and ϵ, which maximize the area under the resulting ROC curve. Table 5.2 lists the parameter search space and the actual values selected by our search.

5.2.2.4. Image Distances

As described in Section 2.3.1, each map node k contains a panoramic image I_k captured at that node's location. Thus, a map edge (k, l) between two nodes k and l also connects the images I_k and I_l. We suspect that the image distance $d(I_k, I_l)$ will tend to be greater if the edge (k, l) crosses a room border. This could be due to occlusion, or due to differences in the visual appearances of adjacent rooms. We therefore use $d(I_k, I_l)$ as an image-distance edge feature.

We now select specific image distance functions d, based on several criteria: d should not depend on specific local image structures, such as corners or edges. Relying on such specific structures could lead to problems in environments where they are not present. Instead, the distance function d should incorporate all pixels in the input images. This is a major difference to the visual doorway detection from Section 5.2.2.3.

As discussed in Section 2.5.2, our robot uses the holistic min-warping method [102] for visual relative-pose estimation. This method operates on low-resolution panoramic images, which have been "unfolded" through reprojection (Section 3.2.2). For this reason, the images I_k stored in our map are also of this type. Figures 5.17 and 5.19 provide example images from the maps used in our experiments. Here, all pixels from the same image column correspond to the same azimuth in robot coordinates. Similarly, all pixels of the same row share the same elevation angle.

In this chapter, we process and unfold the images as described in Section 3.2.2.2. Thus, the resulting images have a resolution of 288×48 pixels, and include elevation angles from $0°$ to $75°$. To avoid aliasing, we blur the original camera image with a 7×7 pixel averaging filter before unfolding it. Note that we deliberately calculate d from these low-resolution images to speed up computations. As an added benefit, this makes the resulting edge feature suitable for robots with only a low-resolution camera. This is another difference to the visual doorway detection from Section 5.2.2.3, which operates on the higher-resolution fisheye images.

The images I_k and I_l are usually recorded under different robot orientations. However, this difference should not affect the image-distance edge features. We therefore require that the distance function d is invariant under azimuthal rotation. In this study, we employ two different distance functions d_c and d_s as edge features. d_c is based on the visual compass introduced by Zeil, Hofmann,

Edge Feature	AUC	J
Edge length	0.71	0.35
Obstacle data	0.78	0.56
Doorway detection	0.87	0.61
Image distance d_c	0.77	0.40
Image distance d_s	0.66	0.29

Table 5.3.: The area-under-curve (AUC) and Youden's J statistic for each of the five edge features. When used as a heuristic for edge-feature selection, high values should indicate a useful feature.

and Chahl [158]. To determine d_c, we calculate the Euclidean image distance

$$\|I_k - I_{l,\delta}\| = \sqrt{\sum_{x,y} (I_k(x,y) - I_l((x+\delta) \bmod w, y))^2} \tag{5.12}$$

for the relative azimuthal image-orientation offset δ. Here, $I_k(x,y)$ refers to the intensity of the pixel (x,y) in the image I_k, while w is the width of the unfolded images. d_c is then the lowest image distance across all possible δ, with

$$d_c(I_k, I_l) = \min_{\delta \in [0,w)} \|I_k - I_{l,\delta}\|. \tag{5.13}$$

The second distance function d_s is based on the image signatures introduced by Menegatti, Maeda, and Ishiguro [96] and extended in [58];

$$d_s(I_k, I_l) = \|s_{\text{afc}}(I_k) - s_{\text{afc}}(I_l)\| \tag{5.14}$$

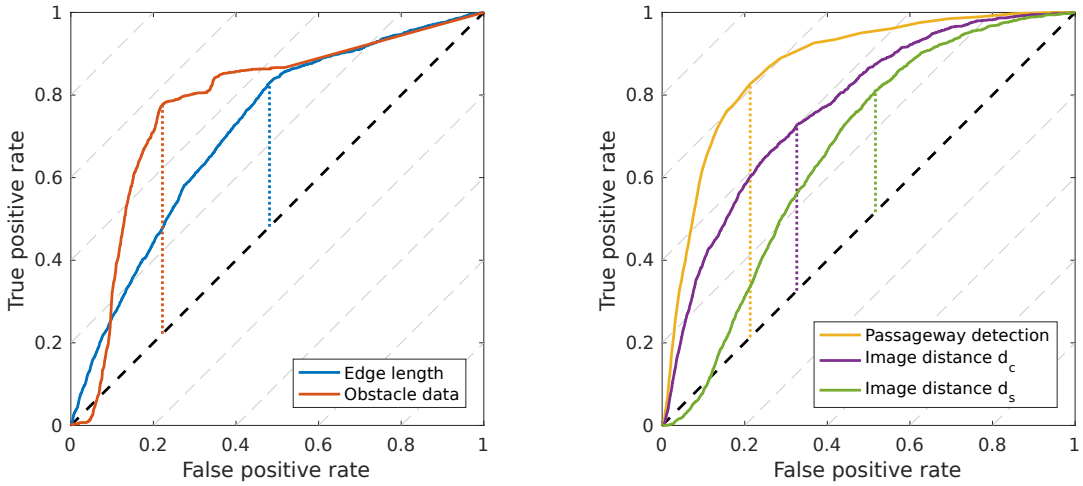
is the Euclidean distance between the image signatures s_{afc} of the two images I_k and I_l. To calculate the signature $s_{\text{afc}}(I_k)$, the 288×48 unfolded image I_k is split into eight equally-sized horizontal segments. A segment consists of $48/8 = 6$ image rows and spans $75^\circ/8 = 12.5^\circ$ of elevation. We then average the rows of each segment, resulting in eight vectors of 288 entries. Next, we calculate the first twelve Fourier coefficients for each of these eight vectors. Finally, $s_{\text{afc}}(I_k)$ is a vector containing the absolute values of all 8×12 Fourier coefficients. By using the absolute values, we eliminate the phase information from the Fourier coefficients. This makes the signatures invariant to the image orientation.

5.2.2.5. Evaluation

In the previous sections, we presented a number of edge features. We now want to ensure that each edge feature is actually useful for room-border detection. In Section 5.3.3, we test the impact of individual edge features on the final room-segmentation results. However, this is computationally expensive, especially when repeated for many different feature combinations. We therefore also try to find a straightforward procedure for identifying useful edge features.

Here, we use receiver operating characteristic (ROC) [42] analysis to evaluate the edge features: First, we classify room-border edges by comparing a single scalar edge feature to a threshold. Varying this threshold then gives us the ROC curve for that feature, as shown in Figure 5.9. In addition, Table 5.3 lists the area-under-curve (AUC) and Youden's J statistic [157]. As indicated in Figure 5.9, J is the maximum height of the ROC curve above chance level. We calculated these results using the combined graph edges from the maps introduced in Section 5.3.1.

According to this analysis, every edge feature presented so far offers at least some use. However, this evaluation is only an approximation, as the actual classifier discussed in Section 5.2.3 is



(a) Camera-independent edge features

(b) Camera-based edge features

Figure 5.9.: Receiver operating characteristic (ROC) curves for the five edge features used in this chapter. (**a**) contains the edge features that do not utilize camera images, while (**b**) contains those that do. For each edge feature, the ROC curve is indicated by a solid line. The location and magnitude of Youden's J statistic is indicated by a dashed line of the same color. A black, dashed diagonal line indicates the chance level.

not linear. Furthermore, the ROC curves of the individual features cannot represent the mutual information between these features. Finally, the map-graph clustering tends to segment the map graph at narrow passageways, as discussed in Section 5.2.4. Correctly classifying these critical edges may thus be more important than a high general classification accuracy. However, this ROC analysis does not take these factors into account. Since this heuristic may be flawed, we also perform room-segmentation experiments with limited subsets of edge features in Section 5.3.3. In Section 5.4.2, we compare the heuristic with those actual room-segmentation results.

5.2.3. Map-Edge Classification

We now determine which map edges cross a room border based on the edge-feature vector introduced in Section 5.2.2. By training a classifier with human-annotated maps, we hope to produce more human-like room segmentations. In practice, we use a support vector machine (SVM) [27] to classify the edges. SVMs are powerful, well-documented, and relatively easy to use. Furthermore, at least one high-quality implementation is readily available to the public [22]. As Hsu, Chang, and Lin [70] have pointed out, the performance of an SVM depends on well-chosen parameters. We therefore perform a systematic search (Section 5.2.3.3) to choose the core parameters used by our method (Section 5.2.3.4).

Since SVMs are well described in the literature, we give only a short overview here; Bishop [13, Chapter 7] offers a more general introduction. Here, we employ a C-SVM maximum-margin classifier [27]. This classifier varies the model parameters $\vec{w}$ and b to optimize

$$\min_{\vec{w},\vec{\xi}} \frac{\vec{w}^T\vec{w}}{2} + C\sum_i \xi_i \tag{5.15}$$

under the constraints

$$y_i(\vec{w}^T\phi(\vec{x}_i) + b) \geq 1 - \xi_i, \tag{5.16}$$

$$\xi_i \geq 0 \; \forall i. \tag{5.17}$$

$\vec{x}_i \in \mathbb{R}^n$ and $y_i \in \{-1, 1\}$ are the training vectors and class indicators, and $C > 0$ is a regularization parameter. Note that Equation (5.16) can always be fulfilled by increasing the slack variables ξ_i, even in case of non-separable training data. However, this also increases the value of Equation (5.15) according to the regularization parameter C. Additionally, ϕ is a function that maps each input vector to a higher-dimensional space. This is necessary to solve classification problems that are not linearly separable in the input space. After training the model, we can classify a given input vector $\vec{x}$ with the decision function

$$\mathrm{sgn}(\vec{w}^T\phi(\vec{x}) + b). \tag{5.18}$$

Instead of the function ϕ, we can also make use of a kernel function K. In this case, $\vec{w}$ is written as a linear combination of the vectors $\phi(\vec{x}_i)$ according to the factors α_i; this results in $\vec{w} = \sum_i \alpha_i y_i \phi(\vec{x}_i)$ [22]. Substituting this in $\vec{w}^T\phi(\vec{x})$, we get

$$\vec{w}^T\phi(\vec{x}) = \sum_i \alpha_i y_i K(\vec{x}_i, \vec{x}), \tag{5.19}$$

with the kernel function $K(\vec{x}_i, \vec{x}_j) = \phi(\vec{x}_i)^T\phi(\vec{x}_j)$. Within our method, we use a radial basis function (RBF) kernel

$$K(\vec{x}_i, \vec{x}_j) = e^{-\gamma\|\vec{x}_i - \vec{x}_j\|^2}. \tag{5.20}$$

We choose this kernel because it is commonly regarded as a good first choice for novel problems [70].

In our experiments, we employ the C-SVM implementation from the `libsvm` library [22]. In general, we follow the usage guidelines provided by the library authors [70]. However, we occasionally deviate from this procedure, as required by our specific classification problem. We will discuss these changes where they occur.

5.2.3.1. Data Scaling

As recommended by Hsu, Chang, and Lin [70], we scale the individual edge features. Without scaling, features with a very large value range would drown out those with a smaller range. The `libsvm` authors recommend a linear scaling that maps each feature to a range of $[-1, 1]$. This mapping depends solely on the minimum and maximum of a given feature. It is therefore very vulnerable to outliers, which occur for some of our edge features. Instead, we use the standardized value $x_i' = \sigma^{-1}(x_i - \bar{x})$ for the feature x_i. Here, $\bar{x}$ and σ are the mean and standard deviation for the given feature in the training data. As $\bar{x}$ and σ depend on all training values, we expect this standardization to be less sensitive to outliers.

5.2.3.2. Training and Cross-Validation

We now train the C-SVM on our training data using the appropriate `libsvm` functionality. Our map-edge classification problem consists of just two classes: The first class contains edges where both nodes lie within the same room. The second class consists of room-border edges, for which the two nodes are part of different rooms. After training, we use this model to adjust the edge weights of new maps: For edges classified as crossing a room border, we divide the weight by the edge-weight factor ρ. Figure 5.10 shows an example edge-classification result in a map graph.

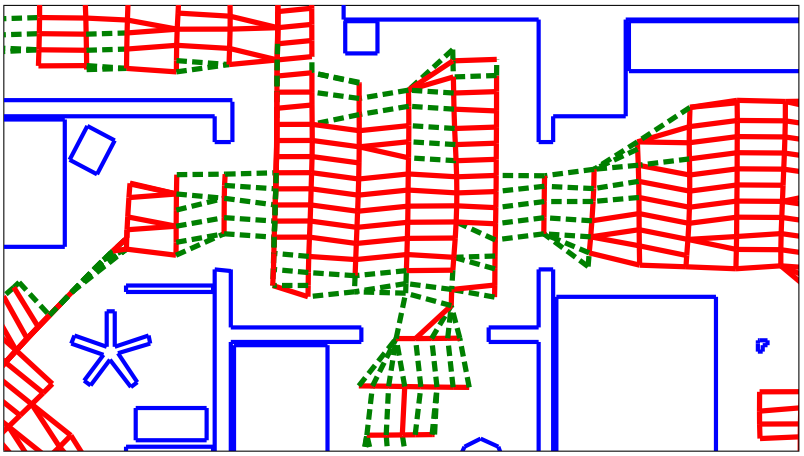


Figure 5.10.: The effect of edge classification on the map graph from Figure 5.3. Edges classified as crossing a room border are drawn as green, dashed lines. While most room-border edges are classified correctly, we also note some false-positive results within the rooms.

In a typical environment, only a small fraction of the edges cross a room border. As a result, the two classes in our edge-classification problem are unbalanced. For our training data, we find the ratio between the classes to be ≈ 38. The SVM may neglect the correct classification of room-border edges in favor of the more common within-room edges. One solution to this class-balance problem has been presented by Osuna, Freund, and Girosi [118]: For those training data $(\vec{x}_i, y_i)$ that belong to the room-border class, we replace C with a higher value of $C^+ = wC$; here, w is the class weight. Thus, misclassification of the second class has a higher impact on the objective function from Equation (5.15), which compensates for the class imbalance.

5.2.3.3. Parameter Selection

We now have to select the regularization parameter C, the kernel parameter γ, the class weight w, and the edge-weight factor ρ. Hsu, Chang, and Lin [70] suggest choosing the SVM parameters C and γ through an exhaustive search using cross-validation. In this case, we would first split the training data T into equally-sized subsets $t_1, ..., t_n$. To evaluate a given parameter (C, γ), we would then perform n-fold cross validation: For every $k \in [1, n]$, we would train the SVM on the set $T \setminus t_k$ and test it on the subset t_k. Next, we would compute the average classification accuracy across all test subsets t_k. By repeating this cross-validation step for different parameters, we could select the best (C, γ).

However, this parameter-selection method is not ideal for our problem. For our purposes, the SVM classification accuracy is only a secondary concern. Instead, the primary goal is to optimize the room-segmentation result. We thus select our parameters using a criterion based directly on that result. This also lets us to expand the search to include all four parameters C, γ, w and ρ.

There are many possible criteria to judge a graph-clustering result [94, Chapter 16.3]. For a systematic, large-scale search, the criterion must be easy to compute without human input. In this study, we use a cluster impurity based on the well-known cluster-purity measure [94]: Following the procedure from Section 5.3.1.1, each map node is assigned to a ground truth room j, forming the node sets R_j. Thus, each set R_j corresponds to a single room j in our ground truth. Next, we calculate the purity ψ for the node clusters found during graph clustering (Section 5.2.4). If each cluster i is represented by a node set C_i, the purity for a graph with n nodes is

$$\psi = \frac{1}{n} \sum_i \psi_i = \frac{1}{n} \sum_i \max_{j'} \left| C_i \cap R_{j'} \right| . \tag{5.21}$$

Here, ψ_i is the largest number of nodes in C_i that shares the same room j'.

We consider two types of potential errors within the room-segmentation result: For the first type of error, one cluster contains nodes from multiple rooms, and thus ψ_i is reduced. In the second type of error, one room is split into several clusters. If these clusters do not contain nodes from other rooms, then the purity is not affected. This property of the purity is similar to our room-segmentation goals: For user-robot interaction, the user will usually assign room names to the clusters. If multiple clusters are assigned to the same room, the clusters can easily be merged. For place recognition, clusters can also be merged if they are found to belong to the same place. This is not true in the opposite case, where a cluster contains nodes from multiple rooms. Here, we do not know which nodes in the cluster belong to which room. We thus consider it important that our criterion is sensitive to the first type of error. In the case of the second type of error, one room is split into multiple clusters. However, for our method, the number of rooms is also equal to the number of clusters. Consequently, another cluster must then contain nodes from more than one room. Within our experiments, the purity criterion is thus also sensitive to the second type of error. We therefore use the purity criterion to judge the quality of a given room segmentation.

We also modify the cross-validation scheme for the parameter search. As described in Section 5.3.1, we use the same environment to generate multiple training maps. The basic parameter-selection method does not account for this during cross validation. Subsequently, maps from every environment might be included in both the training and validation sets. Our method would thus never encounter previously unseen environments during validation. As a result, the validation would be less informative regarding our method's performance in such novel environments. To prevent this, we ensure that each cross-validation subset t_i only contains maps from a single environment. Maps from this specific environment will thus not occur in the training set $T \setminus t_i$. This way, the training process will have no knowledge of the validation environment from t_i.

We can now evaluate a given parameter combination (C, γ, w, ρ) using this modified scheme: First, we perform the cross-validation scheme described above. We split the training maps T into subsets t_i according to their environment of origin. For each t_i, we train the SVM on the set $T \setminus t_i$. We then use this SVM to perform room segmentation on every map in t_i. Next, we calculate the purity ψ for every room-segmentation result, followed by the mean purity $\bar{\psi}$. Finally, we rank the given parameter combination based on its mean impurity $\bar{v} = 1 - \bar{\psi}$.

5.2.3.4. Parameter Selection Results

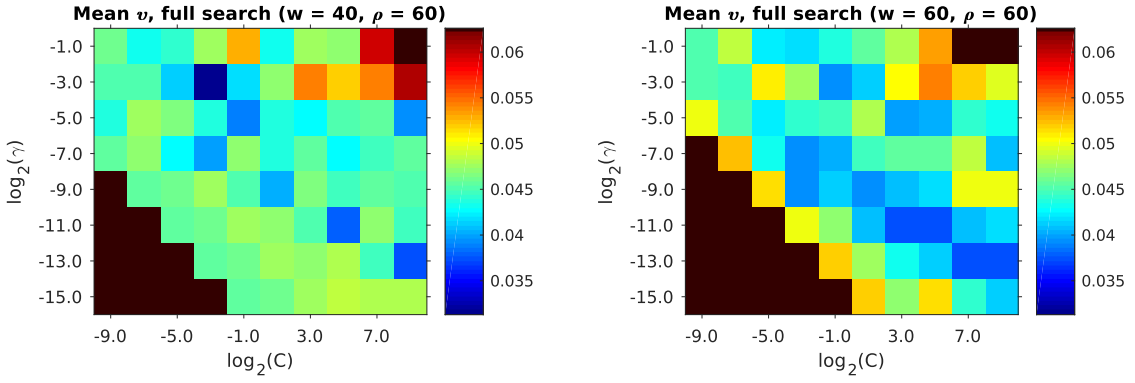
We now select the best parameters $(\hat{C}, \hat{\gamma}, \hat{w}, \hat{\rho})$ for our room-segmentation method, based on the maps from Section 5.3.1. As recommended by Hsu, Chang, and Lin [70], we begin with a coarse search using exponential step sizes for C and γ. The search space is specified by the `full` entry from Table 5.4. We found SVM convergence to be very slow for values of $C \gtrsim 2^9$, occasionally even reaching the default `libsvm` iteration limit. This area of the parameter space may still provide good room-segmentation results. However, due to the computational effort required, we do not generally extend our search in this direction.

Figure 5.11 and Table 5.5 show that comparatively low mean impurities $\bar{v}$ occur across a wide variety of parameters. Next, we perform a fine search around the parameter combination with the lowest $\bar{v}$ listed in Table 5.5. This is followed by an even finer search over an even smaller parameter space. These searches correspond to the `fine` and `extrafine` search spaces in Table 5.4. Figure 5.12 gives an overview of these results. As with the coarse search, we included the parameters with the lowest $\bar{v}$ in Table 5.6.

Through these searches, we have now determined the values for $(\hat{C}, \hat{\gamma}, \hat{w}, \hat{\rho})$ which we will use in Section 5.3. We also have an upper bound for the lowest mean impurity $\bar{v}$ achieved by our full method. As stated before, low mean impurities $\bar{v}$ appear over a wide range of parameter combinations. However, Figure 5.11 also shows that large areas of the parameter space are

Experiment	$\log_2(C)$	$\log_2(\gamma)$	w	ρ	p
`full`	−9:2:9	−15:2:−1	20, 40, 60	20:20:120	1440
`fine`	−9:1:5	−9:1:0	40	50:10:80	600
`extrafine`	−6.5:0.5:0.5	−5.5:0.5:−0.5	40	50, 60, 70	495

Table 5.4.: Parameter search spaces used for our room-segmentation method. For each search, all possible values of C, γ, w and ρ are combined, resulting in a total of p parameter combinations. Here, $n{:}s{:}m = \{n + ks | k \in \mathbb{Z} \wedge (n + ks) \in [n, m]\}$, for example $1{:}2{:}7 = \{1, 3, 5, 7\}$.



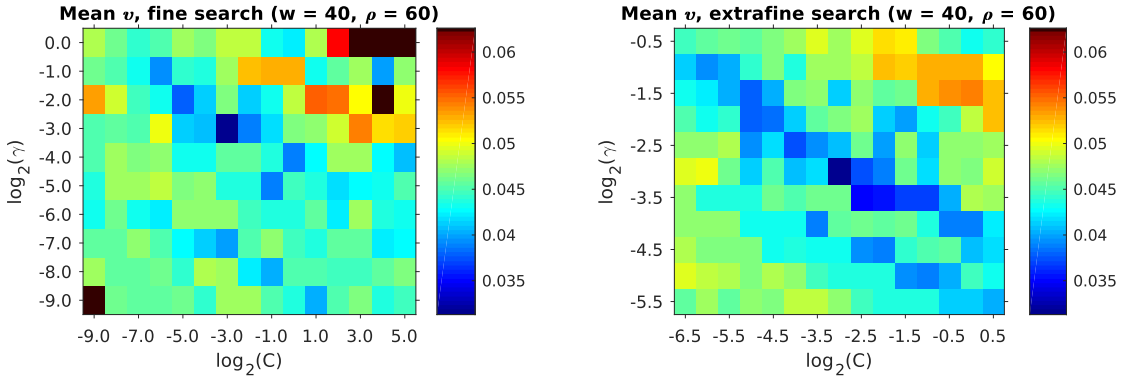
(a) Full search, $w = 40$

(b) Full search, $w = 60$

Figure 5.11.: Plots showing the mean impurity $\bar{v}$ for the `full` search space from Table 5.4. Values of $\bar{v}$ greater than the maximum of the color scale are shown in dark brown. Only the plots for $w \in \{40, 60\}$ and $\rho = 60$ are shown here. According to Table 5.5, these values for w and ρ result in some of the lowest $\bar{v}$.

$\log_2(C)$	$\log_2(\gamma)$	w	ρ	$\bar{v} \times 100$	$\tilde{v} \times 100$
−3	**−3**	**40**	**60**	**3.13**	**1.04**
−3	−3	40	80	3.15	1.12
−3	−3	40	100	3.55	1.12
−3	−3	40	120	3.55	1.12
3	−11	60	60	3.72	1.30
9	−13	60	60	3.73	1.37
7	−13	60	60	3.74	1.30
5	−11	60	60	3.74	1.37

Table 5.5.: The results of the parameter search over the `full` search space from Table 5.4. We sort the parameter combinations according to the lowest mean impurity $\bar{v}$, and list the first eight out of all 1440 entries. The values with the lowest mean impurity are given in bold, and are used in most of our subsequent experiments. This table also contains the median impurity $\tilde{v}$. For the sake of readability, the impurities have been multiplied by a factor of 100.



(a) Fine search **(b)** Extra-fine search

Figure 5.12.: Plots showing the mean impurity $\bar{v}$ for the `fine` and `extrafine` search space listed in Table 5.4. Values of $\bar{v}$ greater than the maximum of the color scale are shown in dark brown. Only the plots for $w = 40$ and $\rho = 60$ are shown, as these contain the lowest $\bar{v}$ according to Table 5.6.

$\log_2(C)$	$\log_2(\gamma)$	w	ρ	$\bar{v} \times 100$	$\tilde{v} \times 100$
−3.0	**−3.0**	**40**	**60**	**3.13**	**1.04**
−3.0	−3.0	40	70	3.13	1.04
−3.0	−3.0	40	50	3.19	1.37
−2.0	−3.5	40	50	3.35	1.65
−2.5	−3.5	40	60	3.50	1.04

Table 5.6.: The results of the parameter search over the `extrafine` search space from Table 5.4. We sort the parameter combinations according to the lowest mean impurity $\bar{v}$, and list the first five entries. All other details are as in Table 5.5.

unsuitable, due to their high $\bar{v}$. This cross-validation parameter search is thus an important step in achieving good room-segmentation results. Unfortunately, the finer searches (Figure 5.12 and Table 5.6) failed to find better parameters than the initial coarse search.

5.2.4. Map-Graph Clustering

After preprocessing the map graph and classifying its edges, we now segment the map into rooms. In a naive approach, we could simply delete those edges identified as crossing a room border. For a perfect edge-classification result, each of the resulting disconnected map segments would represent one room. Since our edge classification is imperfect, this simple procedure will fail in practice.

Instead, we perform room segmentation by clustering the map graph. As discussed previously, we specifically aim to minimize the normalized cut. To calculate the normalized cut, we adapt the definition from [91]: We begin by constructing the matrices W and D from the n-node map graph. $W = (w_{k,l})$ is the $n \times n$ symmetric weighted adjacency matrix. If the map nodes with index k and l are connected by an edge, the entries $w_{k,l} = w_{l,k}$ are equal to the weight of that edge. If no edge (k, l) exists, then $w_{k,l} = w_{l,k} = 0$. The degree of a node is the sum of the weights from all edges connected to it. This leads to the diagonal degree matrix $D = (d_{k,k})$, with $d_{k,k} = \sum_l w_{k,l}$. For a graph that is split into the m disjoint subsets S_i, the normalized cut is then

$$\mathrm{Ncut}(S_1, ..., S_m) = \sum_{i=1}^{m} \frac{\mathrm{cut}(S_i, \bar{S}_i)}{\mathrm{vol}(S_i)}, \tag{5.22}$$

$$\mathrm{cut}(S_i, \bar{S}_i) = \sum_{k \in S_i, l \notin S_i} w_{k,l}, \tag{5.23}$$

$$\mathrm{vol}(S_i) = \sum_{k \in S_i} d_{k,k}. \tag{5.24}$$

We expect that normalized-cut graph clustering is a good approximation for the room-segmentation problem. In general, minimizing the normalized cut results in compact clusters with relatively weak connections between each other [91]. Similarly, rooms in our maps are usually compact areas connected through narrow passageways. Minimizing the normalized cut also penalizes clusters with a low $\mathrm{vol}(S_i)$. This precludes overly small clusters, even if they would have a low $\mathrm{cut}(S_i, \bar{S}_i)$. Since rooms also typically have a certain minimum size, we consider this to be a useful attribute. Unfortunately, normalized-cut graph clustering is an NP-complete problem [135]. However, several approximate but fast solutions exist [135, 114, 34]. Here, we employ spectral clustering, which was previously used for room segmentation by Zivkovic, Booij, and Kröse [160]. As recommended in [91], we use the variant first presented by Shi and Malik [135].

Since spectral clustering is well-described in the literature, we only give a short summary of the method. First, we calculate the graph Laplacian $L = D - W$ for our map graph. We then solve the generalized eigenproblem $L\vec{v} = \lambda D\vec{v}$ [135]. This is equivalent to solving the eigenproblem $L_{\mathrm{rw}}\vec{v} = \lambda\vec{v}$ for the normalized graph Laplacian $L_{\mathrm{rw}} = D^{-1}L$ [91]. In our implementation, we use Matlab's (version 2016b) `eig` function to solve the eigenproblem. We thus know the resulting eigenvalues λ_i and eigenvectors $\vec{v}_i$. To split the graph into m clusters, we use the eigenvectors $\vec{v}_1, ..., \vec{v}_m$ associated with the m smallest eigenvalues $\lambda_1, ..., \lambda_m$. These eigenvectors form the columns of the matrix $V = (\vec{v}_1, ..., \vec{v}_m)$. V contains n rows, each corresponding to one of the n map nodes.

Each row vector in V now represents one graph node, and we cluster the nodes according to these row vectors. To identify the clusters, we perform k-means clustering in m dimensions [13, Chapter 9]. Here, we use the `kmeans` implementation provided by Matlab (version 2016b), with default parameters. However, the solution found by k-means depends on the randomly chosen initial cluster

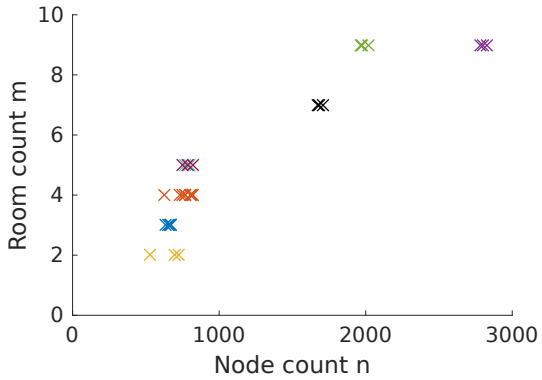


Figure 5.13.: The relationship between the number of nodes n and room count m. Each mark corresponds to one of our maps. Marks of different color represent different environments.

centers. It is possible that badly-chosen initial centers will negatively affect the final clustering result. We therefore repeat the k-means clustering 100 times, each time using different initial centers. From these repetitions, we then select the clustering with the lowest summed distance

$$d_{\text{sum}} = \sum_{j=1}^{n} \|\vec{V}_j - \vec{c}(j)\|. \tag{5.25}$$

$\vec{V}_j$ is the j-th row vector of V, while $\vec{c}(j)$ is the centroid of the cluster that contains the node j. Finally, we form the node sets $C_1, ..., C_m$ from the chosen clustering. Each set C_i contains all the map nodes within the cluster i, and thus represents one of the m rooms.

The normalized-cut criterion does not require that the nodes within the resulting clusters are connected. As a result, spectral clustering could potentially create clusters that consist of several disconnected segments. However, we assume that the floor space within our rooms is connected. These disconnected clusters thus do not fit our room-segmentation goal. An additional step can be added to correct this problem [160]. Since this problem did not occur in our experiments, we did not implement the correction step.

5.2.4.1. Room Count Estimation

Spectral clustering requires the number of clusters, here the number of rooms m, as a parameter. An incorrect value for m would cause an incorrect room-segmentation result. In this chapter, we generally assume that the true room count is known. However, estimating m from the map graph may still be useful for some applications. We therefore test two simple heuristics for room-count estimation.

Node Count Regression

As seen in Figure 5.13, the room count m is related to the number of map nodes n. Our first method exploits this connection to find the room-count estimate $\tilde{m}_1$. Using linear least-squares regression [13, Chapter 3] on our training set, we fit the parameters a, b, c for the model

$$m = f(n) = a\sqrt{n} + bn + c. \tag{5.26}$$

This lets us predict the room count $\tilde{m}_1 = f(n')$ for a new map with n' nodes.

Method	$\overline{\lvert e \rvert}$	$p_{e=0}$	$p_{\lvert e \rvert \leq 1}$
Map-node regression	0.68	0.42	0.90
Eigenvalue gap	0.77	0.45	0.81

Table 5.7.: The room-count estimation results achieved during cross-validation. $\overline{|e|}$ is the average absolute estimation error across all maps. The fraction of maps for which each method gives the correct result is $p_{e=0}$. Similarly, $p_{|e|\leq 1}$ is the fraction for which the error is one or less.

Eigenvalue-Gap Heuristic

The second method for estimating the room count is specific to spectral clustering. This heuristic is based on the eigenvalue gap $g_i = |\lambda_{i+1} - \lambda_i|$. Here, λ_i is the i-th smallest eigenvalue of the normalized graph Laplacian L_{rw} from Section 5.2.4. According to the eigenvalue-gap heuristic [25, 91], for a graph with m easily separable clusters, we find that

$$g_m = |\lambda_{m+1} - \lambda_m| \gg g_l \; \forall l < m. \tag{5.27}$$

As per Section 5.2.4, we assume that rooms correspond to such easily separable clusters. If there are m rooms, we should therefore find m easily separable clusters in the map graph. Consequently, we can estimate the room count $\tilde{m}_2$ from the eigenvalue gaps.

To find $\tilde{m}_2$, we use a classifier to detect the eigenvalue gap that corresponds to the room count. For each training map with a ground truth room count m, we gather the m first eigenvalue gaps g_i. These are assigned a class label of $y_i = 0$ for the first $m - 1$ gaps, and $y_i = 1$ for the mth gap. We repeat this process for every map in our training set, gathering the resulting (g_i, y_i). Next, we train a logistic-regression classifier ([13], chapter 4) on this training data. With the resulting model parameters a, b, the predicted class label $\hat{y} \in \{0, 1\}$ for a gap g is

$$\hat{y} = h(g) = \left\lfloor \frac{1}{1 + e^{-(ag+b)}} + \frac{1}{2} \right\rfloor. \tag{5.28}$$

For a new map with n' nodes, we then calculate the eigenvalues λ'_i and eigenvalue gaps g'_i. From this, we estimate the room count $\tilde{m}_2$ as

$$\tilde{m}_2 = \min \left\{ i | i \in \left[1, n'\right) \wedge h(g'_i) = 1 \right\}. \tag{5.29}$$

In our implementation, we fitted the model parameters a, b to the training data using Matlab's (version 2016b) `mnrfit` function. We also employed Matlab's `mnrval` function to calculate the eigenvalue-gap class labels $h(g'_i)$.

To evaluate these two room-count estimation methods, we employ a cross-validation scheme similar to Section 5.2.3.3. For each of the eight environments in Section 5.3.1, we first train both methods using all maps from the other seven. We then estimate the room count for each map from the current validation environment. Finally, we calculate the estimation error e for each map and method. Here, e is the difference between the estimated and ground truth room count.

In this experiment, the two room-count estimation methods gave mixed results. Table 5.7 shows that both methods determine the correct room count for fewer than half of the maps. However, the mean absolute error was small, with $\overline{|e|} < 1$. In most cases, the estimate was off by ± 1 or less, as shown by the values for $p_{|e|\leq 1}$. Unfortunately, even a small room count error will prevent a correct room segmentation. We thus consider these methods to be of limited practical use, at least in their present form.

5.3. Experiments and Results

In this section, we evaluate the room-segmentation method from Section 5.2 using several experiments. These experiments require training and test data, as well as a ground truth. We generate such data from both real and simulated environments in Section 5.3.1. Next, we present some of the room-segmentation results achieved by our method under cross-validation in Section 5.3.2. Here, we place a special emphasis on those results that deviate from the human-derived ground truth. The experiments in Section 5.3.3 evaluate our method while using different subsets of edge features. This includes clustering map graphs without the SVM classifier, instead using uniform edge weights. Finally, we test our method on previously unused data in Section 5.3.4.

5.3.1. Training and Test Data

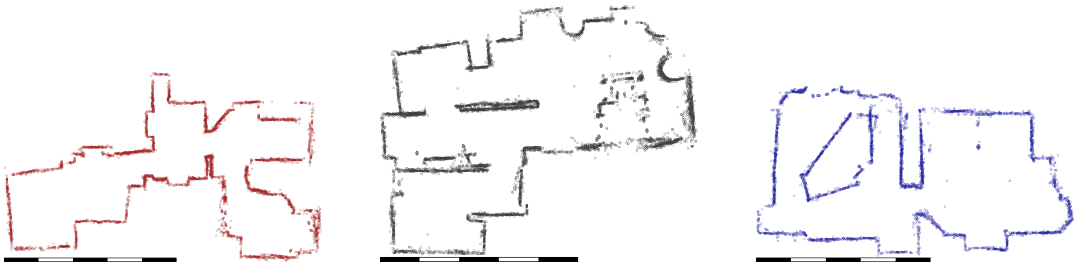
To evaluate our method, we need a sufficiently large number of maps. These maps consist of a map graph, obstacle information, and camera images captured at each map node. Here, we use maps acquired by our robot during cleaning runs. These maps were captured in an office space, a private apartment, and an apartment-like test environment. However, we were not satisfied with the number and variety of these environments. Using a robot simulator, we therefore generated additional maps from five simulated apartments.

Our simulator executes the cleaning-robot control framework from Chapter 2 in a virtual environment. Since the same framework controls both the real and simulated robot, they show a similar behavior. For this study, we built simulated environments from the floor plans of real-world apartments. Additionally, we created detailed 3D models of these environments. These models allow us to generate plausible, panoramic camera images using a raytracing renderer. Our experiments do not differentiate between maps from real and simulated environments. Instead, we always use real-world and simulated maps simultaneously; thus, our method must be able to operate on such a combination.

Our experiments thus make use of eight different indoor environments, three physical and five simulated. For a quick overview, we depict these environments in Figures 5.14 and 5.15. In this study, we attempt to use environments that differ according to several factors, as seen in Table 5.8. For example, they can consist of two to nine rooms; the amount of floor space also varies to a similar degree. We also altered numerous attributes while constructing the 3D models for our simulator experiments. Here, we used a variety of materials for the walls, ceilings, door frames, and other objects. Additionally, some environments are lit mostly by interior lamps, while others are predominantly lit through windows. To simplify robot movement in the simulator, all doors are considered to be fully opened. We therefore modeled only the door frames, but not the movable doors themselves.

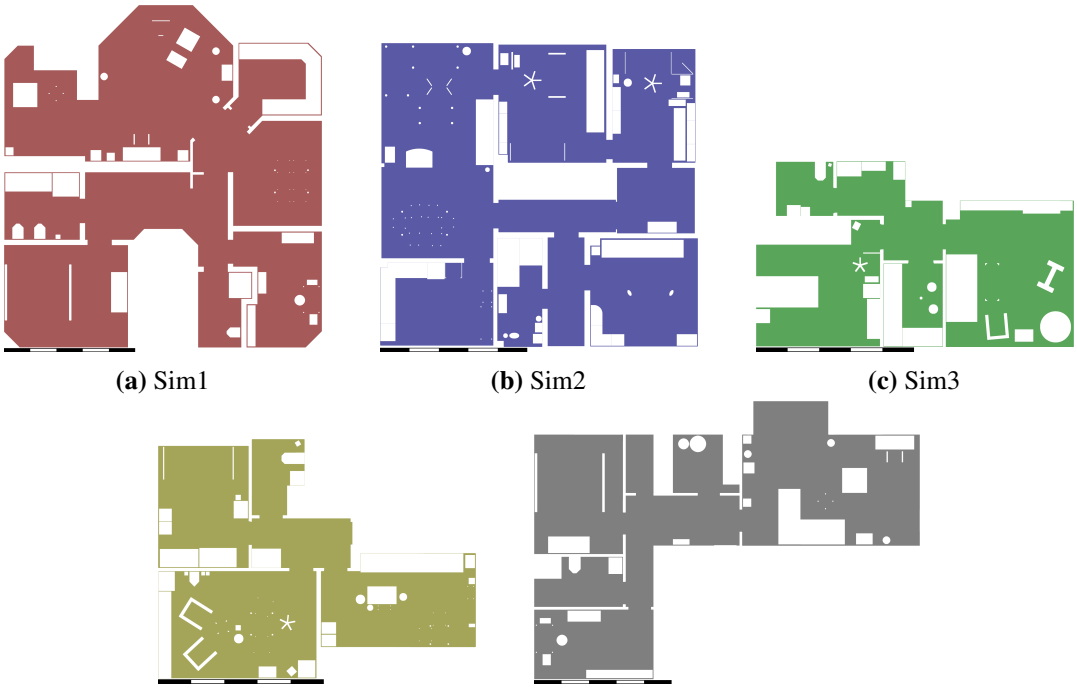
Our physical robot is equipped with the panoramic camera described in Section 2.2. Using this camera, the robot captures a 640×512 pixel fisheye image at the location of every map node. As per Section 3.2.1, a controller adjust the camera's exposure time to maintain a constant average image brightness. Example images from each of the three real environments are shown in Figure 5.16. These are the images used by the doorway detection from Section 5.2.2.3. Figure 5.17 contains the corresponding low-resolution, unfolded images described in Section 5.2.2.4.

Our simulator experiments generate images that are similar to those captured by our robot. Here, we created the 3D models of the environments using the Blender 3D software suite (version 2.78a) [14]. Using Blender's built-in Cycles raytracer, we then render a camera image for each node within the simulated maps. The 3D scene files for the environments are available on our website [45]. These scene files also include the render settings used to generate the images. As with the real camera, the rendered panoramic images use an equidistant fisheye lens. The field of view and angular resolutions of the real and simulated images are also approximately equal. Due



(a) Real1 **(b)** Real2 **(c)** Real3

Figure 5.14.: The three real environments included in this study, visualized using our robot's obstacle maps. Each obstacle point is represented by a small circle. Since our robot's obstacle map lacks global metric consistency, some walls appear to be slightly curved. Each environment is shown with a checkered scale bar indicating a length of $5\,\mathrm{m}$.



(a) Sim1 **(b)** Sim2 **(c)** Sim3

(d) Sim4 **(e)** Sim5

Figure 5.15.: The unoccupied floor space in each of the five simulated environments. Note that some of the floor space may be inaccessible to the robot due to nearby obstacles. Each environment is shown with a checkered scale bar indicating a length of $5\,\mathrm{m}$.

Name	#	Area [m^2]	Map nodes	Rooms	Passageway contrast	Lighting
Sim1	3	125	2806	9	High	Interior
Sim2	3	101	1988	9	Low	Mixed
Sim3	3	51	787	5	Low	Exterior
Sim4	3	46	783	5	Medium	Exterior
Sim5	3	79	1688	7	High	Interior
Real1	5	-	656	3	Low	Exterior
Real2	8	-	760	4	Low	Exterior
Real3	3	-	649	2	High	Exterior

Table 5.8.: The properties of our test and training environments. The second column lists the number of different maps included in our training set. The *area* is the total floor-space, as calculated from the 3D model. This includes space that is covered by furniture, or otherwise inaccessible to the robot. Unfortunately, this value is not available for the real environments. We also list the average number of *map nodes*; this is approximately proportional to the area covered while cleaning. The *rooms* column gives the nominal room count derived from the ground truth. The *passageway contrast* describes the visual distinctiveness of the majority of room-border passageways. This qualitative judgment is based on a visual inspection of the camera images. High-contrast passageways are visually distinct relative to the surrounding walls and ceiling. Conversely, a low-contrast passageway may appear similar to the surrounding structure. Finally, the *lighting* attribute indicates whether an environment is lit predominately by interior or exterior light sources.

(a) Real1 **(b)** Real2 **(c)** Real3

Figure 5.16.: Panoramic images acquired by our robot in three different real environments. These images were captured using our robot's upward-facing panoramic camera. Only the exposed, inner disc is used by our method.

(a) Real1 **(b)** Real2 **(c)** Real3

Figure 5.17.: Low-resolution, unfolded panoramic images acquired by our robot. These images are created by reprojecting, blurring and histogram-equalizing the images in Figure 5.16. In this study, we use such images to calculate the image distances from Section 5.2.2.4.

(a) Sim1 **(b)** Sim2 **(c)** Sim3

(d) Sim4 **(e)** Sim5

Figure 5.18.: Rendered camera images, created from the 3D models of our simulated environments. Here, we show one image for each environment. These images are analogous to the real images shown in Figure 5.16. The overall image dimensions are somewhat different from the real camera images. However, subsequent steps use the same image area and field of view from both image types.

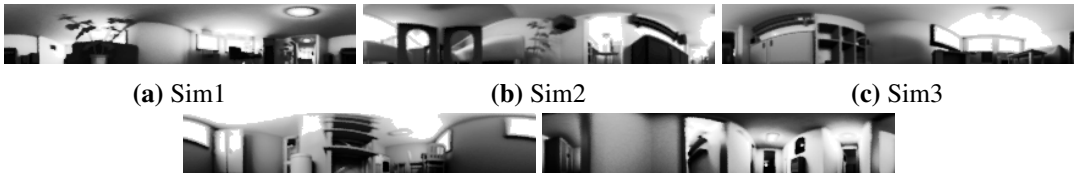
to these similarities, our method processes both simulated and real images in the same manner. Figure 5.18 shows rendered example images for each simulated environment. We also include the corresponding unfolded low-resolution images in Figure 5.19.

We also use a simulated exposure control to approximate the behavior of our real camera: The raytracer renders images with a linear color space and unlimited dynamic range. Thus, the value i of a pixel is proportional to the intensity of the light it receives. Assuming a camera with a linear response and limited dynamic range, we calculate each pixel's resulting value $i' = \min\{ai, i_{\max}\}$. Since we use monochrome images with eight bits per pixel, $i_{\max} = 255$. For each image, we choose a so that the average pixel value $\bar{i'}$ in the unfolded image is $\approx 50\%$ of $i_{\max}$. Starting from $a = 1$, we accomplish this by repeatedly updating $a \leftarrow \frac{1}{2}a(i_{\max}/\bar{i'})$ until $\bar{i'} \approx \frac{1}{2}i_{\max}$. This behavior is similar to our robot's exposure-time controller, which also tries to maintain an average pixel value of 50%.

5.3.1.1. Ground Truth

To train and evaluate our method, we also require a ground truth for each map. This includes ground truth room labels for the map nodes, as well as room-border labels for the edges. Since we aim for a human-like room segmentation, we use a ground truth created by a human operator.

First, the operator is presented with a visualization of the map graph. This visualization is based on the robot's node-position estimates and obstacle map. Second, the operator marks room borders by drawing lines across them. For doors and similar deep openings, several lines can be drawn



(a) Sim1 **(b)** Sim2 **(c)** Sim3

(d) Sim4 **(e)** Sim5

Figure 5.19.: Unfolded low-resolution images, based on the simulated camera images from Figure 5.18. These images were created in the same manner as those shown in Figure 5.17.

to cover the passageway. The lines should be drawn so that they only intersect those map edges that cross the room border. These intersected edges are then marked as room-border edges in the ground truth. Third, the operator also provides the correct number of rooms.

We then use spectral clustering to segment the map graph into ground truth rooms. Here, the room-border edges marked by the operator are assigned a weight of 10^{-4}; all other weights are set to 1. This high weight ratio ensures that spectral clustering will cut the designated room-border edges. After visually confirming the correctness of the resulting room labels, we use them as our ground truth.

5.3.2. Room-Segmentation Experiments

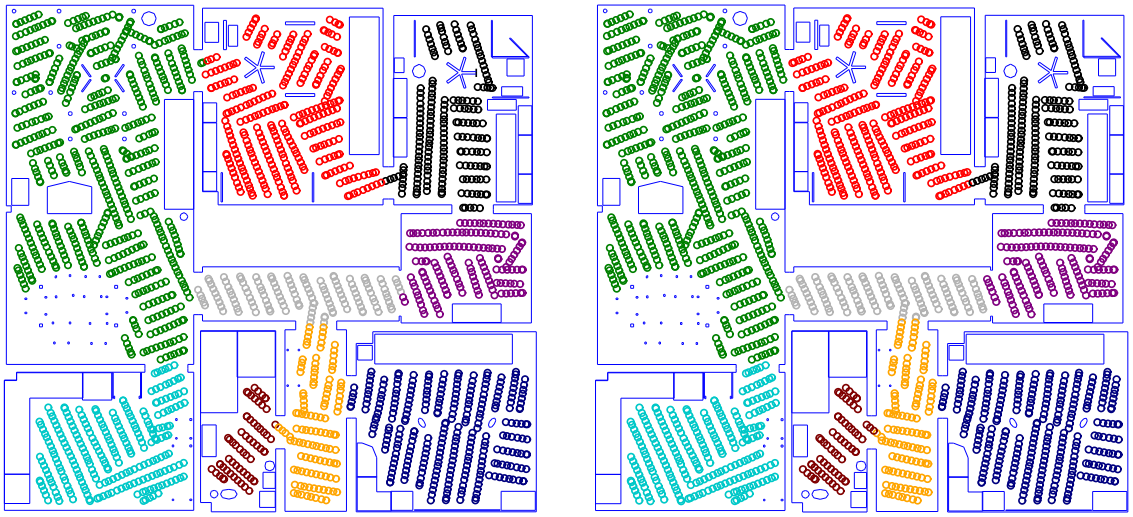
As a basic experiment, we test our room-segmentation method on the maps from Section 5.3.1. Here, we employ a cross-validation scheme, as described in Section 5.2.3.3. When segmenting a map from the environment i, we thus use a classifier trained on all maps not from that environment. For these experiments, the parameters $(\hat{C}, \hat{\gamma}, \hat{w}, \hat{\rho})$ are equal to the boldfaced values in Table 5.6. Under these circumstances, we achieve a mean impurity of $\bar{\upsilon} \times 100 = 3.13$ and a median impurity of $\tilde{\upsilon} \times 100 = 1.04$ across all environments. However, these numbers offer little intuitive understanding of the actual room-segmentation results. We therefore include some of the results to serve as specific examples. For the majority of these maps, the segmentations from our method are very close to the ground truth. In this section, we thus focus on those maps for which this is not the case.

Figure 5.20 gives an example of a successful room-segmentation result. Here, the segmentation found by our method is nearly identical to the ground truth. Our method achieves such results for all maps from the `Sim2`, `Sim3`, `Sim5`, `Real1` and `Real3` environments. This is not the case for the `Sim1`, `Sim4`, and `Real2` environments, where our method may deviate from the ground truth. We illustrate this in Figure 5.21, Figure 5.22, and Figure 5.23, respectively. We will further analyze and discuss these deviations in Section 5.4.1.

5.3.3. Edge-Feature Experiments

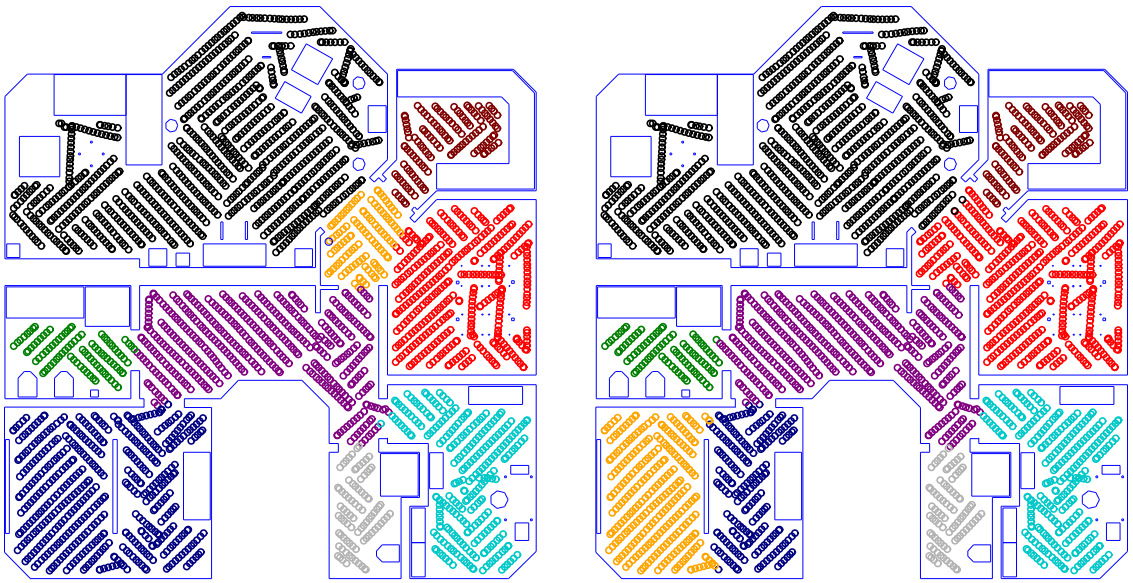
We also perform room-segmentation experiments that do not use the edge classifier. This lets us evaluate its effects on the quality of the results. Here, no edge features are computed and no SVM is trained, and thus the edge weights are not adjusted. The resulting room segmentation is based purely on the map graph, without additional information. Naturally, no parameter search or cross-validation is necessary. Before applying spectral clustering, these uniform edge weights are nevertheless adjusted to correct for map anisotropy, as per Section 5.2.1.

Additionally, we study the importance of individual edge features for our room-segmentation method. We accomplish this by removing specific edge features from the feature vector described in Section 5.2.2. For the `no-camera` experiments, we assume that the robot was not equipped with a camera. We therefore disable the two image-distance features and the visual doorway detection. If the robot was equipped with a narrow-angle ceiling camera, doorway detection would still be



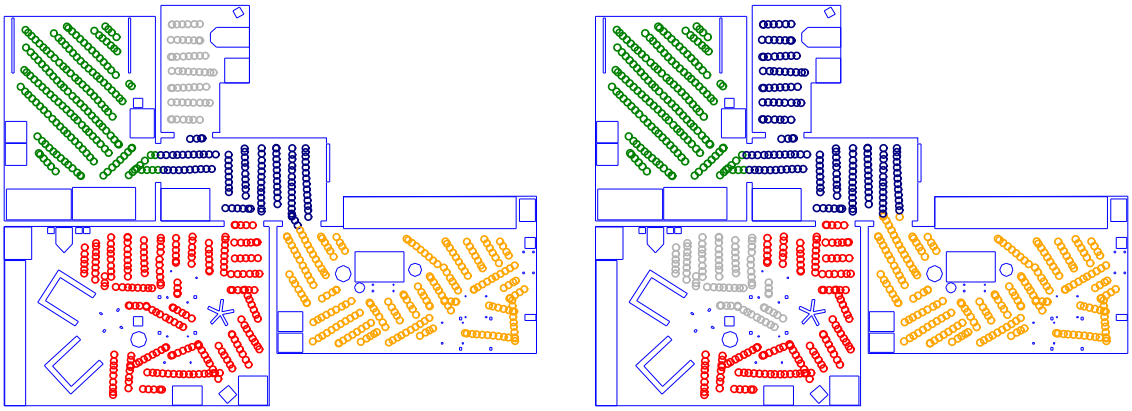
(a) Ground truth **(b)** Result ($v \times 100 = 0.56$)

Figure 5.20.: An example room-segmentation result for the `Sim2` simulated environment. (**a**) shows the ground truth according to Section 5.3.1.1. Similarly, (**b**) displays the result of our room-segmentation method, using the best parameters from Table 5.6. Comparing the two subfigures, we see that our result closely matches the ground truth. Map nodes are represented by circles, and are shown at their true location. Nodes of the same color share the same room label, and therefore belong to the same room. We manually assigned colors to room labels, attempting to associate each color with the same room in both subfigures. Blue lines represent obstacles such as walls or furniture.



(a) Ground truth **(b)** Result ($v \times 100 = 4.03$)

Figure 5.21.: This figure shows a room-segmentation result from the `Sim1` environment. Its style is identical to that of Figure 5.20. In this example, our method failed to correctly segment the small hallway in the upper-right quadrant. As our method uses a fixed room count, the lower-left room is incorrectly split in return.



(a) Ground truth **(b)** Result ($v \times 100 = 7.77$)

Figure 5.22.: A room-segmentation example from the `Sim4` environment, in the style of Figure 5.20. Here, our method fails to segment the small room near the top. Instead, the larger lower-left room is split at a narrow opening between a wall and a chair.

(a) Ground truth **(b)** Result ($v \times 100 = 21.34$)

Figure 5.23.: A room-segmentation result for the `Real2` environment. This plot is similar to Figure 5.20, but based on a real-world robot experiment. Consequently, map nodes are plotted according to the robot's internal position estimate. Small black rings represent points from the robot's obstacle map. As we discuss in Section 5.4.1, some room borders in this environment are relatively indistinctive. This causes incorrect room segmentations.

Experiment	$\log_2(C)$	$\log_2(\gamma)$	w	ρ	p
`basic`	−9:2:9	−15:2:−1	40	20:20:120	480
`no-camera`	−9:2:11	−15:2:1	40	2.5, 5, 10, 20	316
`no-pano`		same as `basic`			0
`no-ceiling`	−9:2:15	−19:2:−1	40	20:20:120	300
`no-obst`		same as `basic`			0

Table 5.9.: Parameter search spaces for experiments with partial edge features, using the notation from Table 5.4. All `no-`$\star$ experiments also include the `basic` search space. For these experiments, p is the number of parameter combinations not already included in the `basic` search space.

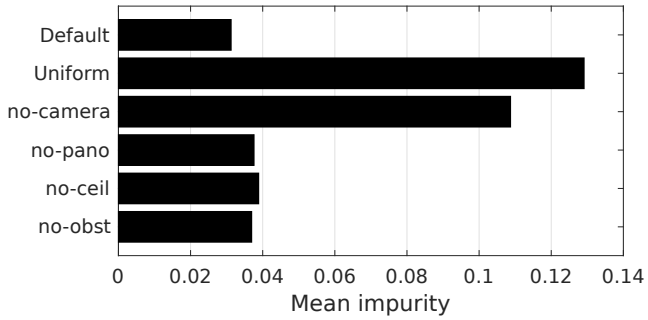


Figure 5.24.: An overview of the mean impurity $\bar{v}$ for experiments with different edge features. Values are taken from Table 5.10.

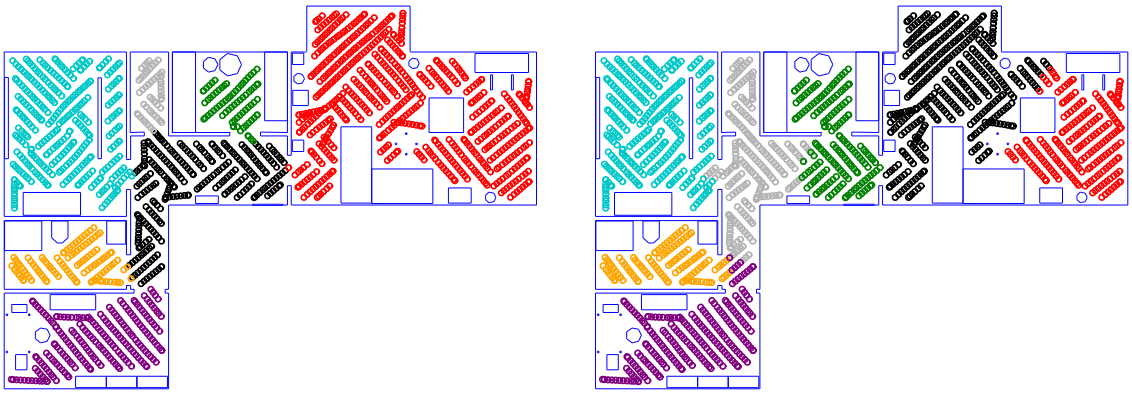
possible. However, we would be unable to compute the panoramic-image distances. We thus deactivate these two image-distance features in the `no-pano` experiments. Similarly, the robot might be equipped with a panoramic camera that does not cover the ceiling. We evaluate this case by excluding the visual doorway detection in the `no-ceiling` experiments. Finally, we switch off the passageway-width feature in the `no-obst` experiments. This lets us test the importance of the robot's obstacle map.

These experiments follow the previous procedure from Section 5.3.2. However, the optimal parameters (C, γ, w, ρ) depend on the composition of the edge-feature vector. For this reason, we have to perform a new parameter search for each of these experiments, as per Section 5.2.3.3. The basic row in Table 5.9 lists the default search space for these experiments. We selected this based on the most promising results from our initial search. Note that we have fixed the class-weight parameter $w = 40$, close to the actual class ratio of approximately 38. In the initial parameter search from Section 5.2.3.3, other values of w offered no improvement. As explained in Section 5.2.3.4, the fine-grained parameter searches also had little effect. We therefore omit such a search for these experiments. These limitations were added to keep the computational effort feasible. For the `no-camera` and `no-ceiling` experiments, the lowest mean impurities $\bar{v}$ occur at the fringe of the `basic` search space. In these cases, we extend the search space to include at least a local minimum for $\bar{v}$; these extensions are listed in Table 5.9.

Table 5.10 and Figure 5.24 contain the results of these edge-feature experiments. Removing all camera-based features or disabling the SVM classifier greatly reduces the room-segmentation quality, as indicated by the increased mean impurity. As an example, we also demonstrate the difference between the methods with the highest and lowest $\bar{v}$: In Figure 5.25, our regular method closely matches the ground truth. The `uniform` variant, which does not use an edge classifier, gives a markedly inferior result. We will discuss this outcome in Section 5.4.2.

Experiment	$\log_2(C)$	$\log_2(\gamma)$	w	ρ	$\bar{v} \times 100$	$\tilde{v} \times 100$
Default	-3	-3	40	60	3.13	1.04
`uniform`	-	-	-	-	12.91	15.46
`no-camera`	-1	-3	40	5	10.87	9.67
`no-pano`	5	-13	40	100	3.76	1.07
`no-ceiling`	15	-13	40	60	3.89	1.37
`no-obst`	-5	-3	40	80	3.69	1.21

Table 5.10.: The results of the experiments with partial edge features. For each variant, we give the parameters with the lowest mean impurity $\bar{v}$. We also include the median impurities $\tilde{v}$. *Default* refers to the results achieved with all edge features, as in Table 5.5. For the `uniform` results, we did not use an edge classifier. Instead, all edges in the map graph had a uniform weight of 1. For this reason, the corresponding row also contains no parameters.



(a) With classifier ($v \times 100 = 0.36$)

(b) No classifier ($v \times 100 = 9.38$)

Figure 5.25.: These plots show results from the `Sim5` environment, with and without using the edge classifier. Our regular method gave the result seen in (**a**), which is nearly identical to the ground truth (not shown). For (**b**), we performed room segmentation with uniform edge weights. This results in several incorrect room borders. The style of this plot is based on Figure 5.20.

Figure 5.26.: The unoccupied floor space of the additional simulated environment, shown in the style of Figure 5.15. This environment includes a wide passageway connecting the top-left room, very small rooms (**top right**), and furniture subdividing larger rooms (**left**).

5.3.4. Additional Tests

Finally, we test our method on previously unseen data by conducting two additional experiments. For the first experiment, we use five new maps, one from each of the simulated environments `Sim1`–`Sim5`. While these environments are not new, the robot will start with a different location and initial heading. The resulting maps are therefore somewhat dissimilar from the existing maps of the same environment. Testing new maps of existing environments is important to our cleaning-robot application: Here, a floor-cleaning robot may clean the same apartment repeatedly from different starting locations, each time resulting in a different map.

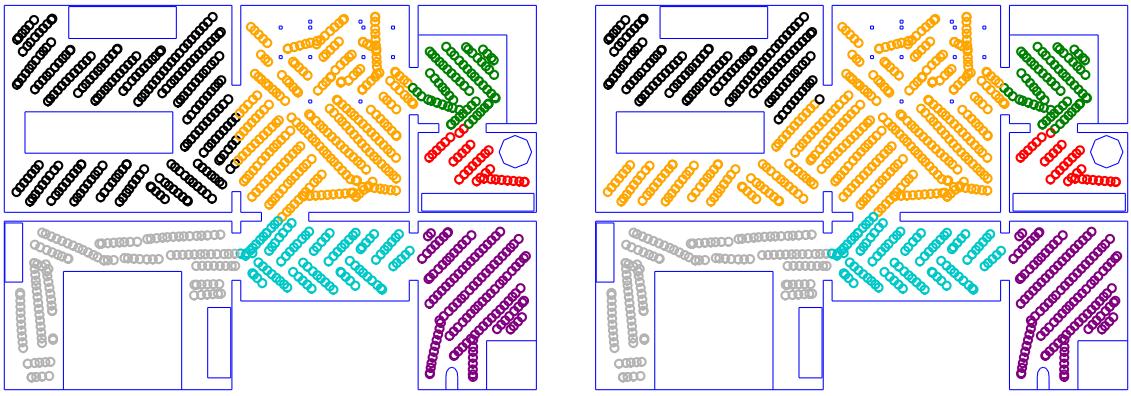
These tests use the bold-faced parameters $(\hat{C}, \hat{\gamma}, \hat{w}, \hat{\rho})$ from Table 5.6, as selected in Section 5.2.3.3. By using new maps, we ensure that they did not influence the selection of these parameters. As before, maps from the test environment are excluded from each experiment's training set. Although the new maps were not used during the parameter search, the outcome was still very similar to the results in Section 5.3.2: Again, the results from the `Sim2`, `Sim3`, and `Sim5` environments were nearly identical to the ground truth. The `Sim1` result exhibited the same problem already shown in Figure 5.21. Similarly, in the `Sim4` results, one of the environment's four room borders was placed incorrectly.

For the second experiment, we test our method on five maps from a completely new environment. This simulated apartment shown in Figure 5.26 is not based on a real-world floor plan. Instead, it was specifically designed to encompass a number of room-segmentation challenges. These include a wide, non-standard passageway, tightly-connected small rooms, and furniture which subdivides large rooms into smaller areas. Since this environment differs from the other environments, we do not perform cross-validation. Instead, we train our classifier with the existing maps from Table 5.8, using the bold-faced parameters from Table 5.6.

Testing on the five maps from the novel simulated environment gave mixed results. In two cases, our room segmentation was nearly identical to the ground truth. For the remaining three maps, our method failed to correctly identify one of the room borders; Figure 5.27 shows such a result. Nevertheless, the other rooms within the environment were segmented correctly.

5.4. Discussion

In this section, we discuss the results from the three types of experiments in Section 5.3. Sections 5.4.1 to 5.4.3 each deal with the results from Sections 5.3.2 to 5.3.4, respectively.



(a) Ground truth **(b)** Result ($v \times 100 = 11.27$)

Figure 5.27.: A room-segmentation result for the additional simulated environment from Figure 5.26, visualized as in Figure 5.20. As seen in (**b**), our method failed to identify the wide passageway separating the upper-left room. However, all other rooms were segmented correctly.

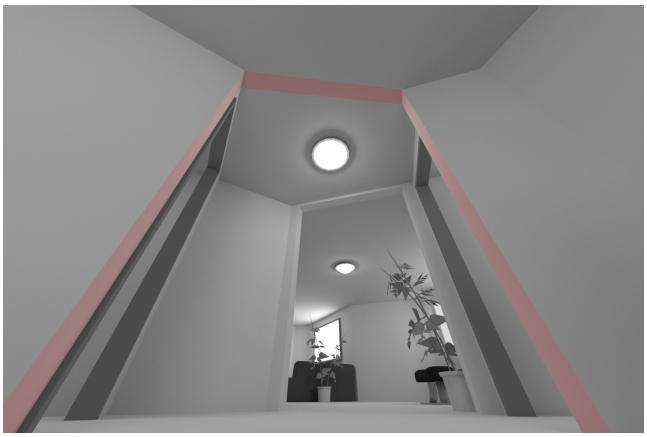
5.4.1. Room Segmentation

Overall, our method gives good results for most environments, as demonstrated in Section 5.3.2. Here, most of the maps are segmented very similarly to the human-derived ground truth, as for example in Figure 5.20. However, our method produces flawed results in the `Sim1`, `Sim4`, and `Real2` environments, as seen in Figures 5.21 to 5.23. These flaws are usually limited to a single misplaced room border. Due to the predetermined room count, such a misplaced border usually affects two rooms: One room is not segmented from a neighboring one, while another room is incorrectly split in two. These failures do not seem to be purely random, but tend to involve specific locations in the environments. We inspected these locations to identify a possible cause for these failures. Subsequently, we found that our method occasionally struggles with room borders that coincide with passageways dissimilar from those found in the training data. This is most noticeable for the failures in the `Sim1` and `Real2` environments.

All our room-segmentation results for the `Sim1` environment contain the same error. Here, our method fails to segment a small hallway from a neighboring room. Figure 5.21 shows this location near the top-right of the map. The passageway between these two rooms is quite unusual, as seen in Figure 5.28a. Unlike most passageways, the highlighted passageway is wider, has no door frame, and lacks an overhead lintel. For comparison, Figure 5.28a also contains more distinctive, regular doorways to the left and right. Figure 5.28b shows the corresponding robot-camera image. In this image, the problematic passageway appears visually indistinct.

For the `Real2` environment, we find partial failures for four out of eight maps. Figure 5.23 gives one example of such a failure. The `Real2` environment also contains several unusual passageways, two of which are highlighted in Figure 5.29a. Our method sometimes fails to detect the room borders at these passageways. We suspect this is because they differ from ordinary passageways, which are much more common in the training data. As before, Figure 5.29b shows the image taken by the robot camera. Compared to the two doorways seen near the bottom of this image, the passageway at the robot's location is less distinctive. The ceiling in the `Real2` environment is also unusual. For example, its height changes in places that do not correspond to room borders. The resulting image edges may cause problems for the visual doorway detection from Section 5.2.2.3.

It would be interesting to view these results in comparison to those achieved in other works (Section 5.1.1). However, like ours, these methods make specific assumptions regarding the robot, map structure, and sensor data. For example, the methods presented in the survey by Bormann et al. [19] operate on occupancy grids with global metric consistency. Our robot does not generate



(a) Passageway overview

(b) Robot camera image

Figure 5.28.: Our method fails to detect this unusual room border in the `Sim1` environment. (**a**) shows a rendered image of the location. The light red coloration highlights the location of the undetected room border. (**b**) contains a robot-camera image taken at the room border; here, the passageway appears visually indistinctive.

(a) Passageway overview

(b) Robot camera image

Figure 5.29.: The `Real2` environment also contains unusual room borders. In (**a**), we highlight two such room borders in red and blue. (**b**) shows the image captured by our robot at the location of the red room border.

such a map, and we thus cannot directly compare these approaches to our own. Consequently, a meaningful comparison would require modifying and optimizing a number of methods for a common experimental framework. This is a significant undertaking, which we leave for future studies.

5.4.2. Edge Features

The results in Section 5.3.3 demonstrate the advantages of using a classifier to adjust the map-edge weights. In comparison to our full method, spectral clustering with uniform edge weights leads to a much higher mean impurity $\bar{v}$. This is shown in Figure 5.24 and Table 5.10. In our experiments, spectral clustering without a classifier will rarely reproduce the human-derived ground truth. The results shown in Figure 5.25b serves as an example for this problem.

Figure 5.24 shows that camera images are useful for our room-segmentation method: $\bar{v}$ remains low if the image distance (Section 5.2.2.4) or visual doorway-detection (Section 5.2.2.3) features are available to the classifier. One or both of these features are present for the `no-pano`, `no-ceiling`, and `no-obst` experiments. In comparison, the passageway width derived from the obstacle data (Section 5.2.2.2) is less useful: Without this edge feature, $\bar{v}$ increases only slightly, as shown in the `no-obst` experiment. Conversely, using only the passageway-width and edge-length features results in a high $\bar{v}$; we demonstrate this in the `no-camera` experiment.

Table 5.10 shows a very high $\hat{C} = 2^{15}$ for the `no-ceiling` experiment. As explained in Section 5.2.3.4, we wish to avoid such high values of C, since they greatly slow the training of the classifier. Initially, we used the `basic` search space listed in Table 5.4, for which $C \leq 2^9$. With this search space, we achieved a lowest mean impurity of $\bar{v} \times 100 = 9.43$. However, the $\bar{v}$ plots from this search showed that $\bar{v}$ decreases further as C increases. Due to the high mean impurity, we decided to extend the search space accordingly. The resulting `no-ceiling` search space is included in Table 5.4. This extended search does indeed result in the lower $\bar{v}$ shown in Figure 5.24. Nevertheless, we decided not to extend the search space towards $C > 2^9$ for the other edge-feature experiments. While lower $\bar{v}$ might be achieved this way, the computational cost is very high. Even when distributed across 40 modern CPU cores, covering this extended search space takes several days. The $\bar{v}$ plots of the other experiments also do not hint at improvements in $\bar{v}$ for these very large values of C.

In Section 5.2.2.5, we tried to predict the usefulness of the edge features from the ROC curves shown in Figure 5.9. We already noted that this is merely a coarse heuristic. Some limitations of this approach become clear when comparing it to the edge-feature experiments: In Table 5.3, there are notable differences in the area-under-curve and Youden's J of the various edge features. However, removing only one type of edge feature causes merely a small increase in the mean impurity $\bar{v}$. This is shown in the `no-ceiling`, `no-pano` or `no-obst` experiments in Table 5.10. In contrast, removing both the doorway-detection and image-distance features greatly increases $\bar{v}$. We failed to predict this behavior based on the ROC analysis alone. In Section 5.2.2.3, we also used ROC analysis to optimize the parameters for visual doorway detection. Due to the limitations of the ROC-based evaluation, it is possible that other parameters would actually lead to better room-segmentation results. However, a joint optimization of the passageway-detection and room-segmentation parameters would be extremely computationally intensive, and is thus not attempted here.

5.4.3. Additional Tests

Based on Section 5.3.4, our tests with previously unseen maps offered only limited further insights. For the new maps from the `Sim1–Sim5` environments, results were similar to those discussed in Section 5.4.1. Most rooms within the new simulated environment were also segmented correctly.

Overall, two out of five maps from the new environment were segmented similar to the ground truth. The other three maps each contained one missed room-border, as for example in Figure 5.27. This is similar to the results from some of the preexisting environments, such as in Figure 5.21.

5.5. Conclusions

The method presented in this work attempts to reproduce a human-like room segmentation from the topo-metric maps generated by our cleaning-robot framework. As discussed in Section 5.4.1, it does so across a variety of real and simulated environments. However, occasional failures do occur, usually when the method fails to identify a room border. These problems often involve unusual passageways, which are unlike those in the training data.

Using a classifier to adjust the map-edge weights is a key component of our method. This way, the result produced by spectral clustering is closer to a human-derived room segmentation. Specifically, these results are better than those achieved using uniform edge weights. According to Section 5.2.3.4, our method's performance strongly depends on the parameters (C, γ, w, ρ). An extensive parameter search is therefore required to make good use of our method. In Section 5.4.2, we investigated the effect of using just a subset of our edge features. Here, the edge features derived from camera images proved to be especially important. Without them, much of the beneficial effect of the classifier was lost. Although not strictly required, a camera is therefore a useful tool for room segmentation using our method.

Since our method uses spectral clustering, the room count must be known in advance. In Section 5.2.4.1, we tested two heuristics for determining this number from the map graph. Unfortunately, both methods gave only mixed results in their current state. If possible, a human operator should therefore provide the room count instead.

5.6. Outlook

In this chapter, we presented an initial version of our room-segmentation method. Due to the number of components, there are many avenues for future improvements. Furthermore, our solution could also be tested in additional environments, or using different sensors or robots.

According to Section 5.4.2, the room-segmentation results strongly depend on the available edge features. Finding more suitable edge features may therefore improve our method. In Section 5.4.1, we noticed that misplaced room-borders tend to involve challenging passageways. Edge features that lead to the correct classification of such passageways would thus be especially beneficial. Unfortunately, we have not yet found a simple measure to accurately predict the usefulness of edge features for room segmentation. For this reason, selecting good edge features currently requires computationally intensive experiments.

Improving the classifier itself should also result in a more accurate room segmentation. Extending the parameter search spaces beyond those listed in Table 5.4 would be a simple first step. Since we already performed extensive searches, we are uncertain whether this would offer meaningful improvements. Using different SVM variants or kernel functions might also improve the classification result [22]. Finally, completely different types of classifiers could be evaluated. Due to the large number of classifiers described in the literature, this would be a considerable task.

Currently, the classifier is trained with all map edges from all training maps. However, this does not take into account the subsequent graph-clustering step: Most of the map edges exist within open spaces, where room borders are unlikely to occur. In contrast, edges in narrow openings and passageways might be more relevant for room segmentation. We therefore suggest a modified training scheme to emphasize these edges while also shrinking the training set: Initially, the training set would consist of all room-border edges, plus a small fraction of the within-room edges. Using

this set, we perform the usual parameter search for the lowest mean impurity $\bar{v}$. Those map edges that were incorrectly cut by misplaced room borders are now added to the training set. By repeating the process, we attempt to grow a training set of edges critical to room segmentation. We hope that this could improve the our results, while keeping the size of the training set manageable.

At the moment, we set the map-edge weights according to the edge-classification result. However, this binary decision does not consider the confidence of the classification. Using the approach proposed by Platt [119], we may estimate the probability p_b that a given map edge crosses a room border. Its edge weight would then be multiplied by $1 + (\rho^{-1} - 1)p_b$, instead of just the fixed values of 1 or ρ^{-1}. We hope that this might reduce the impact of false classifications on the resulting room segmentation.

We also wish to further evaluate our method with regard to different aspects: So far, our maps mostly contain conventional, western-style domestic spaces. First, we are interested in how well our method performs when faced with a greater variety of environments. It is also possible that a larger amount of training data might improve the quality of the results. Second, we wish to compare our method against existing room-segmentation schemes. As mentioned before, these methods usually operate on different types of input data. Thus, the different candidates will have to be modified to operate on common test data. Third, we may try to extend our method to different types of robots. This may require adjustments that account for maps with different structures, as well as different sensors and thus edge features. At the moment, we cannot directly compare our approach to those of others, as discussed in Section 5.4.1. However, it may be possible to adapt a number of existing techniques to work within a shared experimental framework. This would allow for a systematic and meaningful comparison between the results from several methods.

Finally, we would like to incorporate room segmentation into our robot's cleaning strategy. During preliminary simulator experiments, we examined the use of room segmentation for human-robot interaction. Here, a human operator was shown the room-segmentation result on a laptop, visualized as in Figure 5.23b. After labeling the rooms, the operator could issue a series of cleaning commands, such as "Clean the bath, clean the kitchen, then return home". The robot would then execute these tasks using information from the existing map. However, this application still requires extensive testing and development, especially under real-world conditions.

6. Summary, Conclusion and Outlook

In this final chapter, we first review and summarize the content of this dissertation in Section 6.1. In Section 6.2, we then present the overall conclusions drawn from this work. Finally, we consider the general possibilities for future research in Section 6.3. For these last two sections, we emphasize points that were not already covered in the individual chapters.

6.1. Overall Summary

As we have discussed in Chapter 1, floor cleaning in domestic environments is a popular application for autonomous robots. Our group has developed a cleaning-robot framework that solves this task by covering an unknown floor in a systematic manner. Throughout this project, we encountered several unsolved problems which are also relevant to domestic robots in general. Consequently, the objective of this dissertation is to study and resolve some of these questions in the context of our cleaning robot. After describing our framework in Chapter 2, we thus focus on three core problems in Chapter 3, Chapter 4, and Chapter 5. As we shall see, the limited resources available to a typical domestic cleaning robot often lead to unconventional and novel solutions. We now summarize these chapters, briefly discussing the objective, methods, results, and conclusions for each one.

6.1.1. Chapter 2: An Introduction to our Autonomous Cleaning Robot Framework

In Chapter 2, we introduce both the cleaning-robot framework and the physical robot prototype developed by our group. This cleaning robot should systematically traverse the entire floor in an unknown domestic environment. To reduce energy consumption, noise, and component wear, it should do so in the shortest possible distance and time. Limited onboard computing power and sensors add to the challenge of performing this task in real time. To our knowledge, ours is the first academic project that presents a complete framework capable of systematically covering complex, multi-room environments. We limit the project's complexity by emphasizing solutions which are comparatively easy to design and implement: In general, our robot extends the cleaned area by driving a series of straight, parallel lanes. These non-overlapping, meandering lanes form larger parts which are free of gaps; complex floor shapes can then be covered using multiple parts. This systematic approach simplifies the planning problem, and also ensures that the robot does not get lost.

To implement this cleaning strategy, our framework employs three main components: A map stores information about the environment, which is then used by the planners to determine the robot's actions. The resulting plans are executed by the control component, which in turn extends the map based on sensor data. Within this group project, the author made improvements to the planners and the control automatons (Section 2.1.3). These changes enhanced the robot's cleaning performance, especially in narrow spaces. The author also contributed to the robot's collision handling, and improved the mapping, detection, and avoidance of obstacles. Next, we briefly summarize the physical robot and the three framework components introduced in Chapter 2.

Our physical robot resembles commercial domestic models, and is propelled by two powered wheels with differential drive (Section 2.2). As its main sensor, our robot carries a panoramic

fisheye camera which captures the entire hemisphere above the chassis. We use the camera's images to perform numerous tasks, including localization, mapping, navigation, and obstacle detection. All computations are handled by an onboard Intel Atom N2600 dual-core CPU running at 1.6 GHz; this low-power processor is typical for embedded applications.

Throughout its cleaning run, our robot constructs a topo-metric map of the covered area (Section 2.3.1). This map consists of a graph, in which each node represents a location visited by the robot. Map edges indicate whether direct travel between adjacent nodes is possible. Each node is annotated with a metric position estimate and a camera image captured at the corresponding location. This map is used for localization, navigation, and to detect uncleaned space during planning. It does not, however, contain any landmarks extracted from the environment, which reduces the map's complexity. Our robot also builds a map of the obstacle points recorded by the range and collision-detection sensors (Section 2.3.2). This map is only needed to detect free space and to avoid obstacles, and is not used for self-localization. Note that neither the topo-metric nor the obstacle map enforce global metric consistency. Thus, all metric position estimates are only valid relative to those of nearby locations. This greatly simplifies map construction and maintenance.

Based on these maps, our robot uses a planning component to choose its next action (Section 2.4). To simplify this process, we do not plan multiple steps ahead, and select only one action at a time. The part-lane planner determines the next lane by which the robot should extend the cleaned area. Besides the regular meandering lanes, this planner can also generate piercing lanes to clean through narrow openings. A second planner then determines a path through the map graph which leads the robot to the planned lane's starting point.

The plans are translated into motor commands by lane-driving and path-following automatons (Section 2.5). These automatons may make minor adjustments to the plan, such as driving around an unexpected obstacle. This allows for flexible reactions to unforeseen situations, while also keeping the underlying plans simple. Our automatons run inside a main loop, which processes sensor data and handles common housekeeping tasks. By separating these tasks into the main loop, we can simplify the design of the individual automatons. This loop also maintains a probabilistic estimate of the robot's pose, which is updated by integrating the wheel odometry [104]. We correct this estimate by determining the relative pose between the robot and nearby map nodes. For this purpose, we apply a holistic visual pose-estimation method [102] to low-resolution (≈ 0.015 megapixels) panoramic images. The control automatons then use this pose estimate to drive the robot along the desired trajectory.

To test our framework, we conducted experiments in a variety of real and simulated environments (Section 2.6). We found that our robot successfully covers the floor in complex multi-room apartments, offices, and laboratories (Section 2.7). However, covering narrow, tangled spaces with straight lanes can be challenging. A lack of map consistency and limited look-ahead during planning may also reduce our framework's efficiency. We consider these to be minor trade-offs, made necessary by the limited resources available to our robot.

6.1.2. Chapter 3: Comparing Holistic and Feature-Based Methods for Visual Relative-Pose Estimation

Visual relative-pose estimation has many applications in mobile robotics, including in our cleaning-robot framework. To solve this problem, a method determines the relative orientation and bearing between the capture poses of two images. The literature contains many holistic (Section 3.1.2) and feature-based (Section 3.1.1) solutions for this task. Holistic methods incorporate all image pixels into their pose estimate. In contrast, feature methods extract only a limited set of local visual features. They then determine the relative pose from the image positions of matching feature pairs. However, a lack of comparative evaluations makes it difficult to choose an appropriate method for a given application.

We therefore evaluate a selection of pose-estimation methods for their quality, speed, and robustness. Since domestic robots often travel on even floors, we include methods that assume such a planar motion. To our knowledge, this is the first such in-depth comparison made available to the public. Our experiments are based on novel panoramic-image databases, which we recorded using our cleaning robot. These databases include a laboratory and office environment during both day- and nighttime (Section 3.2.1). All images and metadata are available for download [47], enabling others to replicate and extend our results.

To study the accuracy and precision of our candidates, we compare their pose estimates to a ground truth (Sections 3.3.1 and 3.4.1). Besides the average pose-estimation error, we also look for outliers which may have adverse effects in some applications. Note that the typical distance between the image-capture positions depends on the application in which visual pose-estimation is used. We therefore also study how the estimation error is affected by this distance. Here, we observe noticeable differences between the methods, especially at very short or long distances. We also notice that the planar-motion methods usually outperform their nonplanar alternatives.

A high pose-estimation speed is essential to meet the real-time constraints of a domestic mobile robot. We thus measure each candidate's execution time on a desktop computer and on the embedded system carried by our prototype (Section 3.3.2). Some of the candidates also show a marked variance in their execution time. This uncertainty makes it more difficult to use these methods under real-time constraints. For this reason, we also look for outliers in each candidate's execution time. We found that the mean and variance of the execution times can vary widely between the competitors. Furthermore, several candidates are too slow for real-time use on a typical domestic cleaning robot (Section 3.4.2). We also note that the fastest methods are those which make use of the planar-motion assumption.

Next, we evaluate the robustness of our candidates to strong changes in illumination (Section 3.3.3). Such changes are likely to occur in real-world environments, where lighting conditions are not controlled. To this end, we test the candidates using mixed day-night image pairs. The impact on the quality and speed varies widely depending on the method (Section 3.4.3); for some, the mean bearing-estimation error even exceeds 60° under this day-night contrast.

Recall that the planar-motion methods assume that the images are captured by a camera moving in a two-dimensional plane. This reduces the relative pose's degrees of freedom, and leads to fast and accurate pose estimates. However, this assumption is violated if our robot tilts, which commonly occurs on rough floors. We discovered that such a tilt markedly increases the pose-estimation error of the planar-motion methods. For tilt angles greater than $\approx 2^\circ$, the error surpasses that of some non-planar candidates; as expected, the latter remain unaffected by tilts.

Based on these results, we found that no candidate is clearly superior in all respects. While all candidates are at least reasonably accurate under constant illumination, their execution time and robustness varies markedly. We thus provide practical advice for selecting an appropriate method for a given application (Section 3.5). The holistic min-warping method [102] used in our framework produces good pose estimates, which are fairly robust to day-night changes. It is also one of the fastest candidates and provides a near-constant execution time. Consequently, min-warping remains the method of choice for our cleaning-robot project. However, its accuracy remains below that achieved by some of the feature methods. As a planar-motion method, min-warping's estimation errors also increase markedly if the robot is tilted. Methods based on the scale-invariant feature transform (SIFT) [89] are highly accurate and robust to strong illumination changes. Unfortunately, they are currently too slow for real-time use within our cleaning-robot framework. In comparison, feature methods based on binary robust invariant scalable keypoints (BRISK) [84] are much faster, but less robust to illumination changes. The literature contains additional methods and variants not considered in this study; these may be evaluated in future experiments.

6.1.3. Chapter 4: Visually Estimating Camera Tilts from Panoramic Images in Domestic Environments

Mobile domestic robots commonly move across an even floor, and rotate around an axis orthogonal to this plane. Under these circumstances, the degrees of freedom in the robot's pose are reduced from six to three. This planar-motion assumption simplifies many problems in mobile robotics; it is also widely used in our cleaning-robot framework. As demonstrated in Chapter 3, this can also improve the accuracy and speed of visual methods. However, tilting the robot violates the assumption, which quickly degrades the results of such planar-motion methods. We therefore propose two new schemes for visually estimating the robot's tilt relative to the floor plane. Since our robot's camera cannot see the floor, we exploit the vertical elements commonly found in domestic indoor environments. By estimating merely the tilt instead of a full camera orientation, our methods remain simple compared to those in the literature (Section 4.1.1). The tilt-estimation process should be accurate and fast enough to be suitable for real-time tilt correction. Such a correction should make planar-motion visual methods more robust under real-world conditions. We use our previous results from Chapter 3 to define meaningful criteria for the necessary accuracy and speed.

Both our solutions operate on edge pixels extracted from our robot's panoramic images. The image-space method estimates the tilt from the apparent vanishing point of the vertical elements (Section 4.2.1). Here, we make several approximation to compensate for the nonlinear projection of our camera's fisheye lens. We then estimate the tilt directly from the vanishing point's position in the image space. Under the approximations made above, we can easily find this point from the positions and gradients of the edge pixels. To reject edge pixels from non-vertical elements, we introduce an efficient reject-refit scheme; we also evaluate random sample consensus (RANSAC) [44] as a comparison. While this image-space approach is very fast, the approximations also introduce a systematic error in the tilt estimate. We therefore add a final correction step, which multiplies the tilt angle with a correction factor learned from training data.

The vector-consensus method estimates the tilt through conventional linear algebra (Section 4.2.2). Given a calibrated camera model, each edge pixel provides a partial constraint for the tilt in 3D space. After applying RANSAC to reject edge pixels from non-vertical elements, we use a least-squares approach to estimate the tilt from these constraints. This solution is more computationally intensive, but makes fewer approximations than our image-space approach. For the sake of fairness, we also test the machine-learning correction step with this method.

To test our solutions, we record a database of tilted images using our cleaning robot (Section 4.2.3). Based on preliminary experiments, we first predict the tilts our robot is likely to encounter in domestic environments. We then capture images with similar tilts in several different surroundings. This includes locations for which we expect tilt estimation to be difficult, such as underneath furniture. After applying our two methods to these images, we compare the result to a ground truth derived from a statics model. In these experiments, we employ cross-validation to choose appropriate parameter values for our methods.

We find that our solutions achieve good tilt estimates across a range of environments and tilt angles. For the best variants, the average estimation error remains below 1° (Section 4.3.1). Overall, the image-space method with the reject-refit scheme gives the most accurate and fastest results. In comparison, the vector-consensus method is usually slower and less accurate. This method might however be suitable for applications where the approximations made by the image-space method are invalid. Since the methods require little computing time, they are suitable for real-time use on our robot (Section 4.3.2). However, images that contain few vertical elements or many near-vertical elements remain a challenge. Note that some error is always present in these estimates, even if the robot is untilted (Section 4.4). In environments where the robot rarely tilts, a subsequent tilt-correction step may thus be detrimental. In conclusion, we believe that our tilt-estimation methods are suitable for use in tilt correction; this should make planar-motion visual methods

more robust (Section 4.5). However, the actual benefit likely varies depending on the specific environment and application. For future experiments, adding a confidence measure to our methods may assist in rejecting incorrect tilt estimates. We might also correct our estimates using prior knowledge about the magnitude of the tilts in a given environment.

6.1.4. Chapter 5: Human-Like Room Segmentation in Domestic Environments

Domestic robots such as ours commonly operate in indoor environments which consist of rooms connected by passageways. Identifying these rooms within the robot's map has several uses, including in planning and human-robot interaction. In Chapter 5, we thus try to segment our topo-metric maps in a way that matches the room layout identified by a human. To achieve this goal, we propose a novel scheme that learns room segmentation from human-annotated training maps. Unlike existing methods (Section 5.1.1), we achieve this capability without full-scale semantic mapping (Section 5.1.2). Thus, our method does not require preexisting knowledge about the types of rooms which are present in the environment. To take full advantage of our robot's sensors, our approach combines different sources of information. These include the map topology and geometry, obstacle data, and panoramic images recorded by the robot.

Our approach combines three main steps to segment the topo-metric map (Section 5.2): First, we calculate a set of features for each edge in the map graph. We use these features to determine which map edges cross the border between two rooms (Section 5.2.2). Here, we include the length of the edge, a passageway detection based on the obstacle map, a visual doorway detection, and two image-distance measures. We also investigate heuristics for selecting suitable edge features using receiver operating characteristic (ROC) analysis (Section 5.2.2.5). A support vector machine (SVM) classifier [27, 22] then identifies the room-border edges based on their feature vectors (Section 5.2.3). This SVM is trained on manually segmented maps and then used to select the edge weights, which leads to a human-like room segmentation. Finally, we employ spectral clustering [91, 135] to segment the map graph into rooms (Section 5.2.4). This step identifies compact clusters which are weakly connected by the identified room-border edges. Applying spectral clustering also makes our method more robust to incorrectly-classified map edges. In this study, we assume that the correct number of rooms is known. However, we also test two heuristics which estimate this number from the map graph.

We evaluate our solution using numerous maps from both real-world and simulated apartments and offices (Section 5.3.1). These include complex spaces with up to nine differently-shaped rooms and $125\,\mathrm{m}^2$ of floor area. To generate plausible camera images for the simulated environments, we apply a physically-based renderer to detailed 3D models. To judge our results, we select an appropriate metric which compares our room segmentations with a human-derived ground truth. Using this metric, we can then optimize our method's parameters through cross-validation; here, we found that well-chosen parameters are vital to achieve good results. Finally, we conduct a series of room-segmentation experiments, presenting both individual results and a quantitative evaluation (Section 5.3.2). Besides our full method, we also tested variants that exclude some or all of the edge features (Section 5.3.3).

We find that our method successfully reproduces human-like segmentations in both real and simulated environments (Section 5.4.1). By employing machine learning to identify room-border edges, we achieve markedly better results than graph clustering alone (Section 5.3.3). However, unusual room borders which do not occur in the training data may occasionally lead to incorrectly-segmented rooms. We also discovered that our vision-based edge features are especially useful for room segmentation with our method. They offer a greater benefit than the features derived from the map geometry or obstacle data. For this application, we thus recommend equipping domestic robots with panoramic cameras. Our attempts to predict the relative performance of these edge

features through ROC analysis offered only limited insights. Evaluating possible new edge features thus currently requires full-scale testing, as performed in this work. Unfortunately, our room-count estimation heuristics were only partially successful. At the moment, manually specifying the room number thus still gives the best results. In future experiments, we seek to train and test our solution in additional environments, especially in those with unusual room borders. It may also be beneficial to incorporate a measure for the edge classifier's confidence into our method.

6.2. Overall Conclusions

While working on this dissertation, we arrived at some conclusions that are not specific to any chapter. First among these is a strong preference for simple methods which solve specific problems. More complex methods may be manageable in isolation, for example during stand-alone experiments. However, integration into a complete system often causes troublesome interactions. These may increase the complexities of the system to a degree that is nearly impossible to manage. We thus recommend that the components of a domestic robot should be kept as simple as possible, even if some performance must be sacrificed. This issue has also been raised by the developers of the Roomba cleaning robot, who explicitly proclaim that "Complexity kills." [75] Like these authors, we also found that "it is sometimes better to implement a needed feature by inventing a simple, new system than to add two or more familiar systems to the robot" [75].

We believe that real-world testing is essential when developing a domestic robot. Only tests of the entire system can reveal some of the problematic interactions between its components. Experiments under real-world conditions also expose flaws which may only occur in a specific environment. In fact, we are not the first to discover that each new test environment can also highlight a new type of problem [75]. Consequently, even fairly simple robots are bound to require extensive testing throughout the development process [113]. Unfortunately, full-scale real-world experiments also demand a lot of time and resources. Isolating problems discovered during such tests can also be challenging, making it difficult to analyze them. Thus, limited tests in simulations and under controlled conditions can be useful to guide the robot's development. Results from generalized experiments not specific to a particular application may also be more useful to other practitioners. For these reasons, we make frequent use of such experiments throughout this dissertation. When building a complete framework, these results should nevertheless be verified through full-scale testing whenever possible.

Throughout this dissertation, we repeatedly found visual methods to be highly useful. Applications include the control of our robot (Chapter 2), relative-pose estimation (Chapter 3), and tilt measurements (Chapter 4). Such methods also proved highly useful in reproducing human-like room segmentations (Chapter 5). Furthermore, cameras are highly versatile sensors which can provide detailed information across a large field of view. They require little power and are cost-effective, lightweight, and compact. Because cameras are passive sensors, their images are however sensitive to changes in illumination, as for example in Chapter 2. Processing high-resolution images can also be computationally intensive, making it more difficult to meet real-time constraints. This can be especially challenging when working with a low-performance embedded computer, as in our cleaning robot. We found that this problem can be alleviated by using only a fraction of each image. For example, the holistic pose-estimation method in Chapter 3 uses a very low image resolution of ≈ 0.015 megapixels. Similarly, our tilt-estimation schemes from Chapter 4 work on just a small set of edge pixels. In both cases, this results in methods suitable for real-time use on our cleaning robot. Since images are highly regular data structures, visual methods can also make good use of parallel processing. Overall, we therefore consider cameras our sensor of choice for domestic robots like our own.

6.3. Overall Outlook

In this final section, we speculate on possible directions for our cleaning-robot research. These are large-scale future developments, which extend beyond the specific advancements proposed at the end of each chapter.

It may be useful to integrate the results from this work into our cleaning-robot framework: By implementing a tilt correction based on Chapter 4, the visual methods should become more robust to uneven ground. While we have previously added such a correction, we still need to study its effects through full-scale, real-world testing. The room-segmentation method from Chapter 5 opens up new avenues for hierarchical planning and user interaction. As per Section 5.6, we performed initial experiments in which an operator orders the robot to clean specific rooms. In this case, the robot needs to maintain two different maps: On one hand, we use a persistent map of the whole environment for room segmentation and user interaction. This persistent map also guides the robot to the target rooms. On the other hand, we must construct a new map to record the areas covered in the current cleaning run. This is a major modification of our framework, which still requires additional development and testing. Finally, some of the pose-estimation candidates from Chapter 3 were more accurate or robust than our current method. However, these alternatives are also significantly slower, and would thus require a much more powerful onboard computer. Integrating these improvements into a complete prototype will likely require a dedicated research project.

Lawn-mowing robots have recently grown more popular in the consumer market. Their coverage task is similar to that of cleaning robots, but must be performed outdoors. It may thus be possible to derive a lawn-mowing framework from our cleaning-robot research. However, the planar-motion assumption used in our framework is usually not valid in outdoor spaces. We must therefore redesign the map, planning, and control components for such nonplanar environments. Additionally, the framework must be able to deal with different seasonal and weather conditions. Unlike the floor in an indoor space, a lawn is not necessarily enclosed by obstacles. We thus have to detect the boundaries of the grass, possibly through visual methods. Members of our group have already worked on some of these problems, such as visual localization in nonplanar environments [139, 35] or robustness to weather-induced changes [36, 68]. However, a complete lawn-mowing framework has not yet been developed.

An entirely new cleaning-robot framework may tackle some of the basic flaws in our current approach. For example, it might be worthwhile to abandon the straight lanes and rigid part-lane structure from Chapter 2. Instead, the robot would perform a curved trajectory that works well even in tight spaces. However, this additional freedom may complicate the planning problem. Finally, we may use multiple robots that cooperate with each other to solve the cleaning task. Acting as a networked group, they could combine their sensor data to build a single map. Since this lies far beyond our current framework, it would require a new, dedicated research project.

A. Software and Settings for Visual Pose-Estimation Experiments

This appendix contains the software versions and algorithm settings used for our visual relative-pose estimation experiments in Chapter 3.

Component	Version
Linux kernel	3.13.0
GNU C/C++ compilers	4.8.4
OpenCV library	2.4.9
OpenGV library	2015-11-18
WarpingSIMD	Release Code6

Table A.1.: A list of major components used in this work, together with their version numbers. For the OpenGV library, the date at which the library was checked out from its official repository is given.

Parameter	Value
Unfolded image size	288×48
Unfolding rect. avg. filter	7×7
Elevation range	$0°$ to $75°$
n_ψ, n_α	96
n_ψ search fraction	0.3
Pixel scale	2310.72
Post scale	227.552
ρ range	0 – 100 (= off)
Scale planes	9
Scale-plane precision	8 bit
Search precision	16 bit
Fine search	Off
Interpolation	Off
Max. scale factor	2.0
Max. threshold	2.5

Table A.2.: Settings for min-warping. Details on these parameters can be found in [98]. We reused values that gave good results during previous robot experiments. The pixel scale and post scale were determined for the databases used in our experiments.

Parameter	Value
Number of features	500 or 1500
Pyramid scale factor	1.2
Pyramid levels	8
Edge threshold	31
Feature scoring	Harris
BRIEF patch size	31

(a) ORB settings

Parameter	Value
Detection threshold	30
Number of octaves	3
Sampling pattern scale	1

(b) BRISK settings

Parameter	Value
Number of features	Unlimited
Layers per octave	3
Contrast threshold	0.04
Edge threshold	10
Initial Gaussian filter	$\sigma = 1.6$

(c) SIFT settings

Parameter	Value
Hessian threshold	350
Number of octaves	4
Layers per octave	2
Descriptor size	128
Rotation invariance	Enabled

(d) SURF settings

Table A.3.: Settings for ORB, BRISK, SIFT and SURF features, as passed to the OpenCV library. All values except for the number of ORB features and the SURF threshold are library defaults. Here, we adjusted the SURF threshold to detect an average of 1500 features per image.

B. Bibliography

[1] D. Aguilera, J. G. Lahoz, and J. F. Codes. "A new method for vanishing points detection in 3D reconstruction from a single view." In: *Proceedings of the ISPRS Commission V*. International Society for Photogrammetry and Remote Sensing. 2005.

[2] A. Alahi, R. Ortiz, and P. Vandergheynst. "FREAK: Fast retina keypoint." In: *Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 510–517. DOI: 10.1109/CVPR.2012.6247715.

[3] P. Alcantarilla, J. Nuevo, and A. Bartoli. "Fast explicit diffusion for accelerated features in nonlinear scale spaces." In: *British Machine Vision Conference*. BMVA Press, 2013. DOI: 10.5244/C.27.13.

[4] H. Andreasson and T. Duckett. "Topological localization for mobile robots using omni-directional vision and local features." In: *Symposium on Intelligent Autonomous Vehicles*. International Federation of Automatic Control. 2004. DOI: 10.1016/S1474-6670(17)31947-X.

[5] M. E. Antone and S. Teller. "Automatic recovery of relative camera rotations for urban scenes." In: *Conference on Computer Vision and Pattern Recognition*. Vol. 2. IEEE. 2000, pp. 282–289. DOI: 10.1109/CVPR.2000.854809.

[6] H. Asoh et al. "Jijo-2: An office robot that communicates and learns." In: *Intelligent Systems* 16 (5), 2001, pp. 46–55. DOI: 10.1109/MIS.2001.956081.

[7] H. Bay et al. "Speeded-up robust features (SURF)." In: *Computer Vision and Image Understanding* 110 (3), 2008, pp. 346–359. DOI: 10.1016/j.cviu.2007.09.014.

[8] J.-C. Bazin et al. "Globally optimal line clustering and vanishing point estimation in Manhattan world." In: *Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 638–645. DOI: 10.1109/CVPR.2012.6247731.

[9] J.-C. Bazin et al. "Rectangle extraction in catadioptric images." In: *International Conference on Computer Vision*. IEEE. 2007, pp. 1–7. DOI: 10.1109/ICCV.2007.4409208.

[10] J.-C. Bazin et al. "Rotation estimation and vanishing point extraction by omnidirectional vision in urban environment." In: *The International Journal of Robotics Research* 31 (1), 2012, pp. 63–81. DOI: 10.1177/0278364911421954.

[11] D. Bekele, M. Teutsch, and T. Schuchert. "Evaluation of binary keypoint descriptors." In: *International Conference on Image Processing*. IEEE. 2013, pp. 3652–3656. DOI: 10.1109/ICIP.2013.6738753.

[12] D. Binding and F. Labrosse. "Visual local navigation using warped panoramic images." In: *Towards Autonomous Robotic Systems*. University of Surrey, Guildford. 2006, pp. 19–26.

[13] C. Bishop. *Pattern recognition and machine learning*. New York, NY, USA: Springer, 2007.

[14] *Blender 2.78a*. 2016. URL: http://www.blender.org/ (visited on 03/14/2018).

[15] O. Booij, B. Kröse, and Z. Zivkovic. "Efficient probabilistic planar robot motion estimation given pairs of images." In: *Robotics: Science and Systems*. Cambridge, MA, USA: MIT Press, 2010, pp. 201–208.

[16] O. Booij et al. “Navigation using an appearance based topological map.” In: *International Conference on Robotics and Automation*. IEEE. 2007, pp. 3927–3932. DOI: 10.1109/ROBOT.2007.364081.

[17] O. Booij, Z. Zivkovic, et al. *The planar two point algorithm*. Tech. rep. IAS-UVA-09-05. University of Amsterdam, Faculty of Science, Informatics Institute, 2009. URL: http://hdl.handle.net/11245/1.308876 (visited on 03/14/2018).

[18] R. Bormann, J. Hampp, and M. Hägele. “New brooms sweep clean: An autonomous robotic cleaning assistant for professional office cleaning.” In: *International Conference on Robotics and Automation*. IEEE. 2015, pp. 4470–4477. DOI: 10.1109/ICRA.2015.7139818.

[19] R. Bormann et al. “Room segmentation: Survey, implementation, and analysis.” In: *International Conference on Robotics and Automation*. IEEE. 2016, pp. 1019–1026. DOI: 10.1109/ICRA.2016.7487234.

[20] T. Botterill, S. Mills, and R. Green. “Bag-of-words-driven, single-camera simultaneous localization and mapping.” In: *Journal of Field Robotics* 28 (2), 2011, pp. 204–226. DOI: 10.1002/rob.20368.

[21] G. Bradski et al. “The OpenCV library.” In: *Doctor Dobbs Journal* 25 (11), 2000, pp. 120–126. URL: http://www.opencv.org/ (visited on 03/14/2018).

[22] C.-C. Chang and C.-J. Lin. “LIBSVM: A library for support vector machines.” In: *ACM Transactions on Intelligent Systems and Technology* 2 (3), 2011, p. 27. DOI: 10.1145/1961189.1961199.

[23] Z. Chen and S. T. Birchfield. “Visual detection of lintel-occluded doors from a single image.” In: *Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. IEEE. 2008, pp. 1–8. DOI: 10.1109/CVPRW.2008.4563142.

[24] O. Chum and J. Matas. “Matching with PROSAC-progressive sample consensus.” In: *Conference on Computer Vision and Pattern Recognition*. Vol. 1. IEEE. 2005, pp. 220–226. DOI: 10.1109/CVPR.2005.221.

[25] F. R. Chung. *Spectral graph theory*. Vol. 92. CBMS Regional Conference Series in Mathematics. Providence, RI, USA: American Mathematical Society, 1997.

[26] P. Corke, D. Strelow, and S. Singh. “Omnidirectional visual odometry for a planetary rover.” In: *International Conference on Intelligent Robots and Systems*. Vol. 4. IEEE. 2004, pp. 4007–4012. DOI: 10.1109/IROS.2004.1390041.

[27] C. Cortes and V. Vapnik. “Support-vector networks.” In: *Machine Learning* 20 (3), 1995, pp. 273–297. DOI: 10.1007/BF00994018.

[28] J. M. Coughlan and A. L. Yuille. “Manhattan world: Orientation and outlier detection by bayesian inference.” In: *Neural Computation* 15 (5), 2003, pp. 1063–1088. DOI: 10.1162/089976603765202668.

[29] M. Cummins and P. Newman. “FAB-MAP: Probabilistic localization and mapping in the space of appearance.” In: *The International Journal of Robotics Research* 27 (6), 2008, pp. 647–665. DOI: 10.1177/0278364908090961.

[30] E. R. Davies. “Image space transforms for detecting straight edges in industrial images.” In: *Pattern Recognition Letters* 4 (3), 1986, pp. 185–192. DOI: 10.1016/0167-8655(86)90018-8.

[31] A. P. Dempster, N. M. Laird, and D. B. Rubin. “Maximum likelihood from incomplete data via the EM algorithm.” In: *Journal of the Royal Statistical Society, Series B*, 1977, pp. 1–38.

[32] P. Denis, J. H. Elder, and F. J. Estrada. “Efficient edge-based methods for estimating Manhattan frames in urban imagery.” In: *European Conference on Computer Vision*. Springer. 2008, pp. 197–210. DOI: 10.1007/978-3-540-88688-4_15.

[33] A. Denuelle and M. V. Srinivasan. “Bio-inspired visual guidance: From insect homing to UAS navigation.” In: *International Conference on Robotics and Biomimetics*. IEEE. 2015, pp. 326–332. DOI: 10.1109/ROBIO.2015.7418788.

[34] I. S. Dhillon, Y. Guan, and B. Kulis. “Weighted graph cuts without eigenvectors: A multilevel approach.” In: *Transactions on Pattern Analysis and Machine Intelligence* 29 (11), 2007. DOI: 10.1109/TPAMI.2007.1115.

[35] D. Differt. “Real-time rotational image registration.” In: *International Conference on Advanced Robotics*. IEEE. 2017, pp. 1–6. DOI: 10.1109/ICAR.2017.8023488.

[36] D. Differt and R. Möller. “Spectral skyline separation: Extended landmark databases and panoramic imaging.” In: *Sensors* 16 (10), 2016, p. 1614. DOI: 10.3390/s16101614.

[37] E. W. Dijkstra. “A note on two problems in connexion with graphs.” In: *Numerische Mathematik* 1 (1), 1959, pp. 269–271.

[38] V. Egido et al. “A door lintel locator sensor for mobile robot topological navigation.” In: *Intelligent Data Acquisition and Advanced Computing Systems*. IEEE. 2005, pp. 173–178. DOI: 10.1109/IDAACS.2005.282965.

[39] M. Ester et al. “A density-based algorithm for discovering clusters in large spatial databases with noise.” In: *International Conference on Knowledge Discovery and Data Mining*. American Association for Artificial Intelligence. 1996, pp. 226–231.

[40] R. M. Eustice, H. Singh, and J. J. Leonard. “Exactly sparse delayed-state filters for view-based SLAM.” In: *Transactions on Robotics* 22 (6), 2006, pp. 1100–1114. DOI: 10.1109/TRO.2006.886264.

[41] C. Evans. *Notes on the OpenSURF library*. Tech. rep. CSTR-09-001. University of Bristol, 2009. URL: http://www.cs.bris.ac.uk/Publications/Papers/2000970.pdf (visited on 03/14/2018).

[42] T. Fawcett. “An introduction to ROC analysis.” In: *Pattern Recognition Letters* 27 (8), 2006, pp. 861–874. DOI: 10.1016/j.patrec.2005.10.010.

[43] J. Figat, T. Kornuta, and W. Kasprzak. “Performance evaluation of binary descriptors of local features.” In: *International Conference on Computer Vision and Graphics*. Springer, 2014, pp. 187–194. DOI: 10.1007/978-3-319-11331-9_23.

[44] M. A. Fischler and R. C. Bolles. “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography.” In: *Communications of the ACM* 24 (6), 1981, pp. 381–395. DOI: 10.1145/358669.358692.

[45] D. Fleer. *3D models of apartments*. 2017. URL: http://www.ti.uni-bielefeld.de/html/people/dfleer/apartmentmodels.html (visited on 03/14/2018).

[46] D. Fleer. “Human-like room segmentation for domestic cleaning robots.” In: *Robotics* 6 (4), 2017, p. 35. DOI: 10.3390/robotics6040035.

[47] D. Fleer. *Panoramic image database*. 2016. URL: http://www.ti.uni-bielefeld.de/html/people/dfleer/fisheyedb.html (visited on 03/14/2018).

[48] D. Fleer. "Visual tilt estimation for planar-motion methods in indoor mobile robots." In: *Robotics* 6 (4), 2017, p. 32. DOI: 10.3390/robotics6040032.

[49] D. Fleer and R. Möller. "Comparing holistic and feature-based visual methods for estimating the relative pose of mobile robots." In: *Robotics and Autonomous Systems* 89, 2017, pp. 51–74. DOI: 10.1016/j.robot.2016.12.001.

[50] M. O. Franz et al. "A robot system for biomimetic navigation: From snapshots to metric embeddings of view graphs." In: *Robotics and Cognitive Approaches To Spatial Mapping*. Berlin, Heidelberg, Germany: Springer, 2007, pp. 297–314. DOI: 10.1007/978-3-540-75388-9_18.

[51] M. O. Franz et al. "Learning view graphs for robot navigation." In: *Autonomous Agents*. Boston, MA, USA: Springer, 1998, pp. 111–125. DOI: 10.1007/978-1-4615-5735-7_9.

[52] M. O. Franz et al. "Where did I take that snapshot? Scene-based homing by image matching." In: *Biological Cybernetics* 79 (3), 1998, pp. 191–202. DOI: 10.1007/s004220050470.

[53] F. Fraundorfer and D. Scaramuzza. "Visual odometry: Part II: Matching, robustness, optimization, and applications." In: *Robotics & Automation Magazine* 19 (2), 2012, pp. 78–90. DOI: 10.1109/MRA.2012.2182810.

[54] S. Friedman, H. Pasula, and D. Fox. "Voronoi random fields: Extracting topological structure of indoor environments via place labeling." In: *International Joint Conference on Artificial Intelligence*. Vol. 7. Burlington, MA, USA: Morgan Kaufmann Publishers, 2007, pp. 2109–2114.

[55] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha. "Visual simultaneous localization and mapping: A survey." In: *Artificial Intelligence Review* 43 (1), 2015, pp. 55–81. DOI: 10.1007/s10462-012-9365-8.

[56] E. Garcia-Fidalgo and A. Ortiz. "Vision-based topological mapping and localization methods: A survey." In: *Robotics and Autonomous Systems* 64, 2015, pp. 1–20. DOI: 10.1016/j.robot.2014.11.009.

[57] L. Gerstmayr et al. "A vision-based trajectory controller for autonomous cleaning robots." In: *Autonome Mobile Systeme*. Informatik Aktuell. Berlin, Heidelberg, Germany: Springer, 2009, pp. 65–72. DOI: 10.1007/978-3-642-10284-4_9.

[58] L. Gerstmayr-Hillen et al. "Parsimonious loop-closure detection based on global image-descriptors of panoramic images." In: *International Conference on Advanced Robotics*. IEEE. 2011, pp. 576–581. DOI: 10.1109/ICAR.2011.6088548.

[59] L. Gerstmayr-Hillen et al. "Dense topological maps and partial pose estimation for visual control of an autonomous cleaning robot." In: *Robotics and Autonomous Systems* 61 (5), 2013, pp. 497–516. DOI: 10.1016/j.robot.2012.12.006.

[60] T. Goedemé et al. "Feature based omnidirectional sparse visual path following." In: *International Conference on Intelligent Robots and Systems*. IEEE. 2005, pp. 1806–1811. DOI: 10.1109/IROS.2005.1545111.

[61] P. Graham, A. Philippides, and B. Baddeley. "Animal cognition: Multi-modal interactions in ant learning." In: *Current Biology* 20 (15), 2010, R639–R640. DOI: 10.1016/j.cub.2010.06.018.

[62] G. Guennebaud, B. Jacob, et al. *Eigen v3*. URL: http://eigen.tuxfamily.org/ (visited on 03/14/2018).

[63] B. M. Haralick et al. “Review and analysis of solutions of the three point perspective pose estimation problem.” In: *International Journal of Computer Vision* 13 (3), 1994, pp. 331–356. DOI: 10.1007/BF02028352.

[64] W. K. Härdle, S. Klinke, and B. Rönz. *Introduction to statistics*. Cham, Switzerland: Springer, 2015. DOI: 10.1007/978-3-319-17704-5.

[65] R. I. Hartley and F. Kahl. “Global optimization through rotation space search.” In: *International Journal of Computer Vision* 82 (1), 2009, pp. 64–79. DOI: 10.1007/s11263-008-0186-9.

[66] N. Hawes et al. “Home alone: Autonomous extension and correction of spatial representations.” In: *International Conference on Robotics and Automation*. IEEE. 2011, pp. 3907–3914. DOI: 10.1109/ICRA.2011.5980004.

[67] L. Hillen. “From local visual homing towards navigation of autonomous cleaning robots.” PhD thesis. Bielefeld University, 2013. URL: https://pub.uni-bielefeld.de/publication/2637265 (visited on 03/14/2018).

[68] A. Hoffmann and R. Möller. “Cloud-edge suppression for visual outdoor navigation.” In: *Robotics* 6 (4), 2017, p. 38. DOI: 10.3390/robotics6040038.

[69] M. Horst and R. Möller. “Visual place recognition for autonomous mobile robots.” In: *Robotics* 6 (2), 2017, p. 9. DOI: 10.3390/robotics6020009.

[70] C.-W. Hsu, C.-C. Chang, C.-J. Lin, et al. *A practical guide to support vector classification*. Tech. rep. Department of Computer Science, National Taiwan University, 2003-2016. URL: http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf (visited on 03/14/2018).

[71] A. S. Huang et al. “Visual odometry and mapping for autonomous flight using an RGB-D camera.” In: *International Symposium on Robotics Research*. Vol. 2. International Foundation for Robotics Research. 2011.

[72] W. Hübner and H. A. Mallot. “Metric embedding of view-graphs.” In: *Autonomous Robots* 23 (3), 2007, pp. 183–196. DOI: 10.1007/s10514-007-9040-0.

[73] J. Illingworth and J. Kittler. “A survey of the Hough transform.” In: *Computer Vision, Graphics, and Image Processing* 44 (1), 1988, pp. 87–116. DOI: 10.1016/S0734-189X(88)80033-1.

[74] B. Jähne, H. Scharr, and S. Körkel. “Principles of filter design.” In: *Handbook of Computer Vision and Applications*. Vol. 2. San Diego, CA, USA: Academic Press, 1999, pp. 125–151.

[75] J. L. Jones. “Robots at the tipping point: The road to iRobot Roomba.” In: *Robotics & Automation Magazine* 13 (1), 2006, pp. 76–78. DOI: 10.1109/MRA.2006.1598056.

[76] Y. Ke and R. Sukthankar. “PCA-SIFT: A more distinctive representation for local image descriptors.” In: *Conference on Computer Vision and Pattern Recognition*. Vol. 2. IEEE. 2004, pp. II–506. DOI: 10.1109/CVPR.2004.1315206.

[77] L. Kneip and P. Furgale. “OpenGV: A unified and generalized approach to real-time calibrated geometric vision.” In: *International Conference on Robotics and Automation*. IEEE. 2014, pp. 1–8. DOI: 10.1109/ICRA.2014.6906582.

[78] L. Kneip and S. Lynen. “Direct optimization of frame-to-frame rotation.” In: *International Conference on Computer Vision*. IEEE. 2013, pp. 2352–2359. DOI: 10.1109/ICCV.2013.292.

[79] K. Konolige and M. Agrawal. “FrameSLAM: From bundle adjustment to real-time visual mapping.” In: *Transactions on Robotics* 24 (5), 2008, pp. 1066–1077. DOI: 10.1109/TRO.2008.2004832.

[80] J. Košecká and W. Zhang. “Video compass.” In: *European Conference on Computer Vision*. Springer. 2002, pp. 476–490. DOI: 10.1007/3-540-47979-1_32.

[81] I. Kostavelis and A. Gasteratos. “Semantic mapping for mobile robotics tasks: A survey.” In: *Robotics and Autonomous Systems* 66, 2015, pp. 86–103. DOI: 10.1016/j.robot.2014.12.006.

[82] F. Labrosse. “Short and long-range visual navigation using warped panoramic images.” In: *Robotics and Autonomous Systems* 55 (9), 2007, pp. 675–684. DOI: doi:10.1016/j.robot.2007.05.004.

[83] J.-K. Lee and K.-J. Yoon. “Real-time joint estimation of camera orientation and vanishing points.” In: *Conference on Computer Vision and Pattern Recognition*. IEEE. 2015, pp. 1866–1874. DOI: 10.1109/CVPR.2015.7298796.

[84] S. Leutenegger, M. Chli, and R. Y. Siegwart. “BRISK: Binary robust invariant scalable keypoints.” In: *International Conference on Computer Vision*. IEEE. 2011, pp. 2548–2555. DOI: 10.1109/ICCV.2011.6126542.

[85] K. Levenberg. “A method for the solution of certain non-linear problems in least squares.” In: *Quarterly of Applied Mathematics* 2 (2), 1944, pp. 164–168.

[86] O. Linde and T. Lindeberg. “Object recognition using composed receptive field histograms of higher dimensionality.” In: *International Conference on Pattern Recognition*. Vol. 2. IEEE. 2004, pp. 1–6. DOI: 10.1109/ICPR.2004.1333965.

[87] Z. Liu and G. von Wichert. “Extracting semantic indoor maps from occupancy grids.” In: *Robotics and Autonomous Systems* 62 (5), 2014, pp. 663–674. DOI: 10.1016/j.robot.2012.10.004.

[88] J. Lobo and J. Dias. “Relative pose calibration between visual and inertial sensors.” In: *The International Journal of Robotics Research* 26 (6), 2007, pp. 561–575. DOI: 10.1177/0278364907079276.

[89] D. G. Lowe. “Distinctive image features from scale-invariant keypoints.” In: *International Journal of Computer Vision* 60 (2), 2004, pp. 91–110. DOI: 10.1023/B:VISI.0000029664.99615.94.

[90] S. Lowry et al. “Visual place recognition: A survey.” In: *Transactions on Robotics* 32 (1), 2016, pp. 1–19. DOI: 10.1109/TRO.2015.2496823.

[91] U. von Luxburg. “A tutorial on spectral clustering.” In: *Statistics and Computing* 17 (4), 2007, pp. 395–416. DOI: 10.1007/s11222-007-9033-z.

[92] M. J. Magee and J. K. Aggarwal. “Determining vanishing points from perspective images.” In: *Computer Vision, Graphics, and Image Processing* 26 (2), 1984, pp. 256–267. DOI: 10.1016/0734-189X(84)90188-9.

[93] E. Mair et al. “Adaptive and generic corner detection based on the accelerated segment test.” In: *European Conference on Computer Vision*. Springer. 2010, pp. 183–196. DOI: 10.1007/978-3-642-15552-9_14.

[94] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*. Cambridge, United Kingdom: Cambridge University Press, 2008. DOI: 10.1017/cbo9780511809071.

[95] D. W. Marquardt. “An algorithm for least-squares estimation of nonlinear parameters.” In: *Journal of the Society for Industrial and Applied Mathematics* 11 (2), 1963, pp. 431–441.

[96] E. Menegatti, T. Maeda, and H. Ishiguro. “Image-based memory for robot navigation using properties of omnidirectional images.” In: *Robotics and Autonomous Systems* 47 (4), 2004, pp. 251–267. DOI: 10.1016/j.robot.2004.03.014.

[97] R. Möller. “A model of ant navigation based on visual prediction.” In: *Journal of Theoretical Biology* 305, 2012, pp. 118–130. DOI: 10.1016/j.jtbi.2012.04.022.

[98] R. Möller. *A SIMD implementation of the minwarping method for local visual homing*. Tech. rep. University of Bielefeld, Faculty of Technology, Computer Engineering Group, 2016. URL: http://www.ti.uni-bielefeld.de/html/people/moeller/tsimd_warpingsimd.html (visited on 03/14/2018).

[99] R. Möller. *Column distance measures and their effect on illumination tolerance in minwarping*. Tech. rep. University of Bielefeld, Faculty of Technology, Computer Engineering Group, 2016. URL: http://www.ti.uni-bielefeld.de/html/people/moeller/tsimd_warpingsimd.html (visited on 03/14/2018).

[100] R. Möller. “Local visual homing by warping of two-dimensional images.” In: *Robotics and Autonomous Systems* 57 (1), 2009, pp. 87–101. DOI: 10.1016/j.robot.2008.02.001.

[101] R. Möller, M. Horst, and D. Fleer. “Illumination tolerance for visual navigation with the holistic min-warping method.” In: *Robotics* 3 (1), 2014, pp. 22–67. DOI: 10.3390/robotics3010022.

[102] R. Möller, M. Krzykawski, and L. Gerstmayr. “Three 2D-warping schemes for visual robot navigation.” In: *Autonomous Robots* 29 (3), 2010, pp. 253–291. DOI: 10.1007/s10514-010-9195-y.

[103] R. Möller and A. Vardy. “Local visual homing by matched-filter descent in image distances.” In: *Biological Cybernetics* 95 (5), 2006, pp. 413–430. DOI: 10.1007/s00422-006-0095-3.

[104] R. Möller et al. “Cleaning robot navigation using panoramic views and particle clouds as landmarks.” In: *Robotics and Autonomous Systems* 61 (12), 2013, pp. 1415–1439. DOI: 10.1016/j.robot.2013.07.011.

[105] R. Möller et al. “Visual homing in environments with anisotropic landmark distribution.” In: *Autonomous Robots* 23 (3), 2007, pp. 231–245. DOI: 10.1007/s10514-007-9043-x.

[106] J.-M. Morel and G. Yu. “ASIFT: A new framework for fully affine invariant image comparison.” In: *SIAM Journal on Imaging Sciences* 2 (2), 2009, pp. 438–469. DOI: 10.1137/080732730.

[107] O. M. Mozos et al. “Supervised semantic labeling of places using information extracted from sensor data.” In: *Robotics and Autonomous Systems* 55 (5), 2007, pp. 391–402. DOI: 10.1016/j.robot.2006.12.003.

[108] M. Muja and D. G. Lowe. “Fast approximate nearest neighbors with automatic algorithm configuration.” In: *International Conference on Computer Vision Theory and Applications*. Vol. 2. 2009, pp. 331–340.

[109] M. Muja and D. G. Lowe. “Fast matching of binary features.” In: *Conference on Computer and Robot Vision*. IEEE. 2012, pp. 404–410. DOI: 10.1109/CRV.2012.60.

[110] A. C. Murillo et al. “Visual door detection integrating appearance and shape cues.” In: *Robotics and Autonomous Systems* 56 (6), 2008, pp. 512–521. DOI: 10.1016/j.robot.2008.03.003.

[111] R. M. Murray, Z. Li, and S. S. Sastry. *A mathematical introduction to robotic manipulation*. Boca Raton, FL, USA: CRC Press, 1994.

[112] A. Narendra, S. Gourmaud, and J. Zeil. “Mapping the navigational knowledge of individually foraging ants, Myrmecia croslandi.” In: *Proceedings of the Royal Society of London B: Biological Sciences* 280 (1765), 2013. DOI: 10.1098/rspb.2013.0683.

[113] U. Nehmzow. *Scientific methods in mobile robotics: Quantitative analysis of agent behaviour*. London, United Kingdom: Springer, 2006. DOI: 10.1007/1-84628-260-8.

[114] A. Y. Ng, M. I. Jordan, and Y. Weiss. “On spectral clustering: Analysis and an algorithm.” In: *Neural Information Processing Systems Conference*. Vol. 14. 2. Neural Information Processing Systems Foundation. 2001, pp. 849–856.

[115] D. Nistér. “Preemptive RANSAC for live structure and motion estimation.” In: *Machine Vision and Applications* 16 (5), 2005, pp. 321–329. DOI: 10.1007/s00138-005-0006-y.

[116] D. Nistér, O. Naroditsky, and J. Bergen. “Visual odometry.” In: *Conference on Computer Vision and Pattern Recognition*. Vol. 1. IEEE. 2004, pp. I–652. DOI: 10.1109/CVPR.2004.1315094.

[117] D. Nistér, O. Naroditsky, and J. Bergen. “Visual odometry for ground vehicle applications.” In: *Journal of Field Robotics* 23 (1), 2006, pp. 3–20. DOI: 10.1002/rob.20103.

[118] E. E. Osuna, R. Freund, and F. Girosi. *Support vector machines: Training and applications*. A.I. Memo 1602. Massachusetts Institute of Technology, 1997. URL: http://hdl.handle.net/1721.1/7290 (visited on 03/14/2018).

[119] J. C. Platt. “Probabilities for SV machines.” In: *Advances in Large Margin Classifiers*. Ed. by A. J. Smola et al. Cambridge, MA, USA: MIT Press, 2000, pp. 61–74.

[120] E. Prassler et al. “A short history of cleaning robots.” In: *Autonomous Robots* 9 (3), 2000, pp. 211–226. DOI: 10.1023/A:1008974515925.

[121] A. Pronobis and P. Jensfelt. “Hierarchical multi-modal place categorization.” In: *European Conference on Mobile Robots*. 2011, pp. 159–164. URL: http://aass.oru.se/Agora/ECMR2011/proceedings/papers/ECMR2011_0058.pdf (visited on 03/14/2018).

[122] A. Pronobis and P. Jensfelt. “Large-scale semantic napping and reasoning with heterogeneous modalities.” In: *International Conference on Robotics and Automation*. IEEE. 2012, pp. 3515–3522. DOI: 10.1109/ICRA.2012.6224637.

[123] A. Pronobis et al. “Multi-modal semantic place classification.” In: *The International Journal of Robotics Research* 29 (2-3), 2010, pp. 298–320. DOI: 10.1177/0278364909356483.

[124] R. Raguram, J.-M. Frahm, and M. Pollefeys. “Exploiting uncertainty in random sample consensus.” In: *International Conference on Computer Vision*. IEEE. 2009, pp. 2074–2081. DOI: 10.1109/ICCV.2009.5459456.

[125] A. Ranganathan. “PLISS: Detecting and labeling places using online change-point detection.” In: *Robotics: Science and Systems*. Cambridge, MA, USA: MIT Press, 2010. URL: http://www.roboticsproceedings.org/rss06/p24.pdf (visited on 03/14/2018).

[126] A. Ranganathan and J. Lim. “Visual place categorization in maps.” In: *International Conference on Intelligent Robots and Systems*. IEEE. 2011, pp. 3982–3989. DOI: 10.1109/IROS.2009.5354164.

[127] E. Rosten and T. Drummond. “Machine learning for high-speed corner detection.” In: *European Conference on Computer Vision*. Springer. 2006, pp. 430–443. DOI: 10.1007/11744023_34.

[128] E. Rosten, R. Porter, and T. Drummond. “Faster and better: A machine learning approach to corner detection.” In: *Transactions on Pattern Analysis and Machine Intelligence* 32 (1), 2010, pp. 105–119. DOI: 10.1109/TPAMI.2008.275.

[129] E. Rublee et al. “ORB: An efficient alternative to SIFT or SURF.” In: *International Conference on Computer Vision*. IEEE. 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[130] D. Scaramuzza. “Performance evaluation of 1-point-RANSAC visual odometry.” In: *Journal of Field Robotics* 28 (5), 2011, pp. 792–811. DOI: 10.1002/rob.20411.

[131] D. Scaramuzza and F. Fraundorfer. “Visual odometry.” In: *Robotics & Automation Magazine* 18 (4), 2011, pp. 80–92. DOI: 10.1109/MRA.2011.943233.

[132] D. Scaramuzza, A. Martinelli, and R. Siegwart. “A toolbox for easily calibrating omnidirectional cameras.” In: *International Conference on Intelligent Robots and Systems*. IEEE. 2006, pp. 5695–5701. DOI: 10.1109/IROS.2006.282372.

[133] G. Schindler and F. Dellaert. “Atlanta world: An expectation maximization framework for simultaneous low-level edge grouping and camera calibration in complex man-made environments.” In: *Conference on Computer Vision and Pattern Recognition*. Vol. 1. IEEE. 2004, pp. I-203 –I-209. DOI: 10.1109/CVPR.2004.1315033.

[134] A. Schmidt et al. “Comparative assessment of point feature detectors in the context of robot navigation.” In: *Journal of Automation, Mobile Robotics and Intelligent Systems* 7 (1), 2013, pp. 11–20.

[135] J. Shi and J. Malik. “Normalized cuts and image segmentation.” In: *Transactions on Pattern Analysis and Machine Intelligence* 22 (8), 2000, pp. 888–905. DOI: 10.1109/34.868688.

[136] L. Shi, S. Kodagoda, and G. Dissanayake. “Application of semi-supervised learning with Voronoi graph for place classification.” In: *International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 2991–2996. DOI: 10.1109/IROS.2012.6385549.

[137] T. Spexard et al. “BIRON, where are you? Enabling a robot to learn new places in a real home environment by integrating spoken dialog and visual localization.” In: *International Conference on Intelligent Robots and Systems*. IEEE. 2006, pp. 934–940. DOI: 10.1109/IROS.2006.281770.

[138] H. Stewenius, C. Engels, and D. Nistér. “Recent developments on direct relative orientation.” In: *ISPRS Journal of Photogrammetry and Remote Sensing* 60 (4), 2006, pp. 284–294. DOI: 10.1016/j.isprsjprs.2006.03.005.

[139] T. Stone et al. “Skyline-based localisation for aggressively manoeuvring robots using UV sensors and spherical harmonics.” In: *International Conference on Robotics and Automation*. IEEE. 2016, pp. 5615–5622. DOI: 10.1109/ICRA.2016.7487780.

[140] W. Stürzl and H. A. Mallot. “Efficient visual homing based on Fourier transformed panoramic images.” In: *Robotics and Autonomous Systems* 54 (4), 2006, pp. 300–313. DOI: 10.1016/j.robot.2005.12.001.

[141] W. Stürzl and J. Zeil. "Depth, contrast and view-based homing in outdoor scenes." In: *Biological Cybernetics* 96 (5), 2007, pp. 519–531. DOI: 10.1007/s00422-007-0147-3.

[142] J.-P. Tardif. "Non-iterative approach for fast and accurate vanishing point detection." In: *International Conference on Computer Vision*. IEEE. 2009, pp. 1250–1257. DOI: 10.1109/ICCV.2009.5459328.

[143] J.-P. Tardif, Y. Pavlidis, and K. Daniilidis. "Monocular visual odometry in urban environments using an omnidirectional camera." In: *International Conference on Intelligent Robots and Systems*. IEEE. 2008, pp. 2531–2538. DOI: 10.1109/IROS.2008.4651205.

[144] S. Thrun. "Learning metric-topological maps for indoor mobile robot navigation." In: *Artificial Intelligence* 99 (1), 1998, pp. 21–71. DOI: 10.1016/S0004-3702(97)00078-7.

[145] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. Cambridge, MA, USA: MIT Press, 2005.

[146] R. Toldo and A. Fusiello. "Robust multiple structures estimation with J-linkage." In: *European Conference on Computer Vision*. Vol. 1. Springer, 2008, pp. 537–547. DOI: 10.1007/978-3-540-88682-2_41.

[147] B. J. Tordoff and D. W. Murray. "Guided-MLESAC: Faster image transform estimation by using matching priors." In: *Transactions on Pattern Analysis and Machine Intelligence* 27 (10), 2005, pp. 1523–1535. DOI: 10.1109/TPAMI.2005.199.

[148] P. H. Torr and A. Zisserman. "MLESAC: A new robust estimator with application to estimating image geometry." In: *Computer Vision and Image Understanding* 78 (1), 2000, pp. 138–156. DOI: 10.1006/cviu.1999.0832.

[149] E. Tretyak et al. "Geometric image parsing in man-made environments." In: *International Journal of Computer Vision* 97 (3), 2012, pp. 305–321. DOI: 10.1007/s11263-011-0488-1.

[150] I. Ulrich and J. Borenstein. "VFH+: Reliable obstacle avoidance for fast mobile robots." In: *International Conference on Robotics and Automation*. Vol. 2. IEEE. 1998, pp. 1572–1577. DOI: 10.1109/ROBOT.1998.677362.

[151] A. Vardy and R. Möller. "Biologically plausible visual homing methods based on optical flow techniques." In: *Connection Science* 17 (1-2), 2005, pp. 47–89. DOI: 10.1080/09540090500140958.

[152] S. Vasudevan et al. "Cognitive maps for mobile robots: An object based approach." In: *Robotics and Autonomous Systems* 55 (5), 2007, pp. 359–371. DOI: 10.1016/j.robot.2006.12.008.

[153] K. Weber, S. Venkatesh, and M. Srinivasan. "Insect-inspired robotic homing." In: *Adaptive Behavior* 7 (1), 1999, pp. 65–97. DOI: 10.1177/105971239900700104.

[154] J. Weickert and H. Scharr. "A scheme for coherence-enhancing diffusion filtering with optimized rotation invariance." In: *Journal of Visual Communication and Image Representation* 13 (1-2), 2002, pp. 103–118. DOI: 10.1006/jvci.2001.0495.

[155] H. Wildenauer and M. Vincze. "Vanishing point detection in complex man-made worlds." In: *International Conference on Image Analysis and Processing*. IEEE. 2007, pp. 615–622. DOI: 10.1109/ICIAP.2007.4362845.

[156] X. Yang and Y. Tian. “Robust door detection in unfamiliar environments by combining edge and corner features.” In: *Computer Vision and Pattern Recognition Workshops*. IEEE. 2010, pp. 57–64. DOI: 10.1109/CVPRW.2010.5543830.

[157] W. J. Youden. “Index for rating diagnostic tests.” In: *Cancer* 3 (1), 1950, pp. 32–35. DOI: 10.1002/1097-0142(1950)3:1<32::AID-CNCR2820030106>3.0.CO;2-3.

[158] J. Zeil, M. I. Hofmann, and J. S. Chahl. “Catchment areas of panoramic snapshots in outdoor scenes.” In: *Journal of the Optical Society of America A* 20 (3), 2003, pp. 450–469. DOI: 10.1364/JOSAA.20.000450.

[159] H. Zender et al. “Conceptual spatial representations for indoor mobile robots.” In: *Robotics and Autonomous Systems* 56 (6), 2008, pp. 493–502. DOI: 10.1016/j.robot.2008.03.007.

[160] Z. Zivkovic, O. Booij, and B. Kröse. “From images to rooms.” In: *Robotics and Autonomous Systems* 55 (5), 2007, pp. 411–418. DOI: 10.1016/j.robot.2006.12.005.