Approaches to Efficient Visual Homing of Mobile Robots in Rough Terrain

Annett Stelzer

Technische Fakultät
Albert-Ludwigs-Universität Freiburg im Breisgau

UNI FREIBURG

Dissertation zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften

Betreuer: Prof. Dr. Wolfram Burgard

Approaches to Efficient Visual Homing of Mobile Robots in Rough Terrain

Annett Stelzer

Dissertation zur Erlangung des akademischen Grades Doktor der Ingenieurwissenschaften
Technische Fakultät, Albert-Ludwigs-Universität Freiburg im Breisgau

Dekan:	Prof. Dr. Georg Lausen
Erstgutachter:	Prof. Dr. Wolfram Burgard
	Albert-Ludwigs-Universität Freiburg im Breisgau
Zweitgutachter:	Prof. Dr. Alin Albu-Schäffer
	Technische Universität München
	DLR Institut für Robotik und Mechatronik
Tag der Disputation:	16.12.2016

Zusammenfassung

Mobile Roboter sind in Form von Staubsaugrobotern und Rasenmährobotern längst in unseren Haushalten angekommen und erleichtern uns die Arbeit. Doch auch in anderen Bereichen sind mobile robotische Helfer gefragt, vor allem für Aufgaben, die für den Menschen zu gefährlich wären. Als Beispiele seien Rettungsaktivitäten nach Großunfällen oder Naturkatastrophen, die Erkundung von fremden Planeten und das Erforschen von Höhlen genannt. Roboter für solche Einsatzgebiete müssen in der Lage sein, sich in unbekannten und unstrukturierten Umgebungen sicher zu bewegen. Dabei ist im ersten Schritt keine volle Autonomie erforderlich, vielmehr sollten die mobilen Roboter unterstützende Werkzeuge für Rettungskräfte oder Wissenschaftler sein, um ihnen einen Überblick über die Situation zu verschaffen, auf deren Basis weitere Aufgaben geplant und ausgeführt werden können. Teilautonomie für das Lösen von Aufgaben, für die keine menschliche Interpretationsfähigkeit nötig ist, reicht aus. Dies beinhaltet zum Beispiel das selbstständige Umfahren von bzw. Warnen vor Hindernissen, oder das eigenständige Zurückkehren zum Ausgangspunkt, wenn der Bediener das robotische Werkzeug nicht länger benötigt, oder wenn Objekte zu einer Basis zurückgebracht werden sollen.

Roboter für solche Einsatzzwecke haben in der Regel nur sehr begrenzte Rechen- und Speicherressourcen, entweder, weil sie klein und agil sein müssen um sich in stark unstrukturiertem Gelände bewegen zu können, oder, weil sie in planetaren Explorationsszenarien weltraumqualifizierte Hardware benötigen. Aus diesem Grund sollten die autonomen Fähigkeiten des Systems ressourcensparend implementiert werden.

In der vorliegenden Arbeit wird eine Navigationsmethode für mobile Roboter in unebenem Gelände entwickelt, die das autonome Zurückkehren zum Ausgangspunkt entlang eines zuvor zurückgelegten Pfades beinhaltet. Der Roboter hat vorab keinerlei Informationen über die Umgebung, sondern ist allein auf seine Wahrnehmung der Umgebung (exterozeptiv) und seines eigenen Zustandes (propriozeptiv) beschränkt. Exterozeptive Sensoren sind eine Stereokamera, welche die Erstellung von dichten Tiefenbildern der nahen Umgebung vor dem Roboter ermöglicht, und eine omnidirektionale Kamera, die ein 360° Panorama der Umwelt liefert. Propriozeptive Sensoren sind eine inertiale Messeinheit (IMU), die die Richtung des Gravitationsvektors sowie Beschleunigungen und Drehraten in 6 Freiheitsgraden misst, und Odometriesensoren, welche Rückschlüsse auf die Eigenbewegung des Roboters erlauben. Im Fall von Laufrobotern sind dies Momenten- und Drehwinkelsensoren in den Beingelenken, bei fahrenden Robotern kommen hierfür Radencoder zum Einsatz. Auf Basis dieser Wahrnehmungen soll der Roboter in der Lage sein, aus großer Entfernung (mehr als das hundertfache der Roboterlänge) entlang des zuvor zurückgelegten Pfades zum Aus-

gangspunkt zurückzukehren, ohne sich dabei selbst in Gefahr zu bringen. Um diese Aufgabe recheneffizient zu lösen, wird das Problem in zwei Teilaspekte zerlegt: Die lokale Hindernisvermeidung auf der einen Seite, und die globale Zurückverfolgung des Pfades auf der anderen Seite.

Für die lokale Navigation nutzt der Roboter seine propriozeptiven Sensoren und eine visuelle Odometrie aus der Stereokamera, um seine momentane Position und Orientierung in Bezug auf den Ausgangspunkt zu schätzen. Darauf aufbauend wird aus den Tiefenbildern der Stereokamera eine dichte metrische Höhenkarte der unmittelbaren Umgebung erstellt, auf deren Basis der Roboter die Passierbarkeit des Geländes ermitteln kann. Somit wird eine lokale Hindernisvermeidung erzielt, welche unabhängig von der Länge des insgesamt zurückgelegten Pfades konstante Rechenressourcen benötigt.

Für die globale Navigation speichert der Roboter den zurückgelegten Pfad als eine Sequenz von sogenannten Viewframes. Diese beinhalten die Erscheinung der Umgebung an den jeweiligen Positionen in Form der Landmarkenkonfiguration, die aus den omnidirektionalen Kamerabildern gewonnen wird. Die Erkenntnis, dass sich während der Bewegung des Roboters der Winkel zu weit entfernten Landmarken und Landmarken in Bewegungsrichtung des Roboters nur sehr wenig ändern, wohingegen nahe Landmarken ihre Richtung in Bezug auf den Roboter stärker ändern, wird genutzt, um die Sequenz der Viewframes nichtredundant und hierarchisch in der sogenannten Trail-Map (*Tra*nslation *I*nvariance *L*evel *Map*) zu speichern. Diese Form der Speicherung erlaubt weiterhin die einfache Skalierung der Karte im Fall von Speichermangel. Die Sequenz der Viewframes erlaubt dem Roboter die Berechnung von Richtungsvektoren, die die aktuell wahrgenommene Landmarkenkonfiguration in die abgespeicherte Konfiguration überführt, und so den Roboter von Viewframe zu Viewframe zurück zum Ausgangspunkt leitet. Dadurch benötigt auch die globale Navigation entlang des zuvor gelernten Pfades konstante Rechenzeit unabhängig von der Länge des zurückgelegten Weges, was den Betrieb von ressourcenlimitierten Systemen über sehr lange Strecken ermöglicht.

Abstract

Mobile robots, such as vacuum cleaning robots and robotic lawn mowers, have become part of our daily lives. While they work fully autonomously in well-defined environments, the demand for mobile robots in unstructured and unforeseeable areas, such as search-and-rescue scenarios or planetary exploration, is growing. These robots are not required to have full autonomy, but should rather be tools which support researchers or rescue workers by providing information about a remote environment. For this, they should offer basic autonomy functions, for example obstacle avoidance or autonomous returning. Robots for such tasks often have to be small and agile, which prevents them from carrying heavy sensors and batteries, and thus also limits their computational resources. Therefore, the implemented autonomous capabilities have to work efficiently.

This thesis focusses on the task of robot homing, which is the ability of a robot to return to its starting position after moving away. Since the method should work in unknown, unstructured terrain, it is divided into a local navigation task, which aims at detecting and avoiding obstacles, and a global navigation task, which uses only bearing angles to landmarks to memorize and retrace a path. The method is applicable to ground-based robots equipped with an inertial measurement unit (IMU), a stereo camera, an omnidirectional camera and odometry sensors.

Local obstacle avoidance is accomplished by creating a moving geometric grid map of the immediate surroundings of the robot. For this, the robot computes disparity images from the stereo image pairs and combines them to a dense grid map using the robust and accurate pose estimates obtained by fusing IMU data, visual odometry measurements and robot odometry data. From that, the robot estimates the traversability of the terrain and computes a cost map, which it uses to plan safe paths in a given direction. In contrast, no metric distance information is required for the global path learning and homing task. Instead, the robot only records landmark bearing angle configurations at certain locations along its path. The landmark observations are stored hierarchically by their degree of translation invariance in an efficient and scalable, novel data structure called *Trail-Map* (*Tra*nslation *I*nvariance *L*evel *Map*). For retracing this path back to the home position, the robot computes homing vectors by comparing the current landmark configuration with the stored reference configuration. By combining the local and global navigation approach, a visual homing method for unstructured terrain is achieved, which has very low memory requirements and offers runtimes constant with respect to the length of the traversed path.

Acknowledgments

This thesis was written during my employment at the Institute of Robotics and Mechatronics at the German Aerospace Center (DLR) in Oberpfaffenhofen, Germany. I would like to thank all the people who have supported me throughout the past years and without whom this PhD thesis would not have been possible.

First, I would like to thank my supervisor Prof. Wolfram Burgard for his advice and for the opportunity to pursue my PhD project at the Department of Computer Science at Albert-Ludwigs-University in Freiburg as an external PhD student.

I would like to thank Prof. Gerd Hirzinger and Prof. Alin Albu-Schäffer for giving me the opportunity to work at the DLR Institute of Robotics and Mechatronics. I have been enjoying working with all the people who are not only enthusiastic about robots, but also have the power to bring them into being.

Many thanks to Michael Suppa for being a great head of the Department of Perception and Cognition and for guiding and encouraging me over the past years. I am very grateful for his support, which persists even beyond his time at DLR.

I want to say thanks to my colleagues in the Department of Perception and Cognition and in the mobile robotics group. Many people directly supported my thesis work. First of all, I am grateful to Heiko Hirschmüller for providing a great stereo and visual odometry library and for always being ready to listen to problems. Many thanks to Martin Görner for creating the DLR Crawler and providing it as a test platform for my algorithms, for fruitful discussions, brilliant drawings, as well as for being a great office mate. It has always been fun to work with him. Thanks are due to Christoph Brand for his help with the Pioneer robots. Many thanks to Wolfgang Stürzl for providing the omnidirectional cameras and the image remapping functions that I used. Thanks to Klaus Strobl for supporting me in calibrating the stereo cameras. I am grateful to Tim Bodenmüller for providing the SensorNet framework and for supporting my experiments with the ART tracking system and the Pioneer robots. Furthermore, I would like to thank Thomas Jost and Franz Andert for letting me use their tachymeters for my outdoor experiments, and Iris Grixa for getting me started on how to use them. Many thanks are due to Mallikarjuna Vayugundla for supporting me in the outdoor experiments. I am also thankful to Markus Nowak for sharing his toolchain for bringing Matlab plots to Latex.

I am grateful to all the people who were ready to discuss topics related to this thesis with me. I would like to thank Elmar Mair for illuminating discussions and sharing his great idea which led to the development of the LT-Map, as well as to Marcus Augustine and Bastian

Jäger for advancing this method. I want to thank Martin Schuster for sharing his expertise about SLAM. Special thanks go to Martin Lingenauber for insightful discussions, for asking nasty questions and for constantly pushing me. Thanks are due to Korbinian Schmid for discussions about data fusion. I also want to thank my former office mates Florian Petit and Jörn Vogel for the fruitful discussions, the motivation and the enjoyable time.

Furthermore, I am very thankful to all the people at the institute who keep things running: The IT administrators, the people working at the workshops and our service team.

Finally, I want to thank my family and friends for their support and encouragement. Multiple thanks go to my husband Martin, who is also a wonderful colleague and supported me in the implementation of the navigation method. Many thanks to my children Hannah and Tim for their patience and for reminding me from time to time, that there are more important things in life than a PhD thesis.

Contents

1. Introduction

In recent years, mobile robots have become part of our daily lives. They vacuum our floors, mow our lawns and clean our windows. While these robots work in well-defined environments and perform well-defined, simple tasks fully autonomously, mobile robots are also demanded to work in unstructured and unpredictable areas. For example, the exploration of foreign planets, mines or disaster sites poses high risks to humans, which drives the demand for mobile robots as robust tools to support scientists or rescue workers. These robots must be able to safely traverse unstructured terrain. Full robot autonomy is neither required nor desired, because the decision of what places are interesting and what information to record is better to be made by a human, who can rapidly respond to unforeseeable events [71]. Rather, the robot should be a tool which supports the human operators by providing an overview of the situation, which is necessary for further decisions and task planning. For this, the robot should provide basic autonomy, such as obstacle avoidance, autonomous navigation to a given goal location, and autonomous homing, i.e. returning to the start location once the operator has finished its task or in case objects or samples have to be returned to a base.

Planetary surfaces, mines and disaster sites are not only characterized by unstructured terrain, but also by the absence of any external infrastructure such as GPS, artificial landmarks or beacons. Often there is no usable magnetic field information and only coarse or no a priori maps of the areas exist. Nevertheless, the robot must be able to move through the environment without putting itself to any risk. Robots which are applicable to this task usually have very limited computational resources and limited memory, because they have to be small and agile for rough terrain locomotion, or they have to use space-qualified hardware for planetary exploration. For this reason, the robot's autonomous skills have to be implemented in a very efficient way.

This thesis aims at the development of a method for visual homing in unknown, rough terrain. The term *homing* is borrowed from biology, where it describes the ability of insects to return to their nests after foraging. This special navigation task comprises learning and retracing a path. The robot memorizes a path while either being remotely controlled by an operator, or while autonomously navigating to intermediate waypoints given by the operator. Once the operator commands the robot to return to its base, it uses the memorized information for retracing the path. During homing and autonomous waypoint navigation, the robot has to detect and avoid obstacles. For this, we assume that the environment is static and free of deformable obstacles such as high grass or bushes.

The targeted robotic platforms are wheel-driven and multi-legged mobile robots. Each robot is equipped with lightweight, passive proprioceptive and exteroceptive sensors. Proprioceptive sensors are an inertial measurement unit (IMU) and sensors providing a leg or wheel odometry. Exteroceptive sensors are a stereo camera for visual odometry and for perceiving the geometrical structure of the terrain, and an omnidirectional camera for observing the surrounding visual panorama. The robot should efficiently combine the measurements of the available proprioceptive and exteroceptive sensors to safely navigate to given goal coordinates, to memorize the traveled path and to perform homing along the learned path, even over very long distances (more than a hundred times the robot length). Considering the limited computational resources of the robot, it is important that the computational requirements of the navigation method do not increase with respect to the length of the traveled path.

1.1. State of the Art

For navigation, the robot needs a representation of the environment that allows to plan motions. For this, often metric maps are used. Their main advantage is that they are human-readable because they are Cartesian. Metric maps can be divided into dense grid maps and sparse landmark maps. Dense grid maps represent geometric structure in form of a regular grid and can be 2D [21], 2.5D [53, 102, 47] or 3D [69, 89]. They usually have a fixed resolution and need much storage space because they also model uninteresting regions with the same resolution. Since metric grid maps can represent the geometric structure of the terrain, they are used for obstacle detection and avoidance. In contrast, sparse metric maps only store the coordinates of landmarks in the environment, either in 2D or in 3D. Thus, they consume less memory, but usually do not give information about the geometric structure of the environment. Therefore, they do not allow for obstacle avoidance, but are well suited for robot localization. The main problem of metric maps is keeping them consistent over a large range. The robot always encounters uncertainties in its odometry readings and environment observations, which must be corrected, especially when the robot comes back to a previously visited location. Usually, methods for simultaneous localization and mapping (SLAM) [20, 3] are used for this task. Examples for computationally tractable SLAM algorithms are FastSLAM [68], SEIF SLAM [90], iSAM [44] or RatSLAM [67].

In case a representation of the structure of the environment is not needed and the maps do not have to be human-readable, non-metric maps are a good choice. They resemble a topological representation of the environment, in which only the appearance of important places and the order of their visit is stored [106, 7, 25]. The robot never knows exact coordinates of its location, but it knows which place it is close to and in which direction other locations can be reached. Hence, for each place different information can be stored, and, by only storing what is needed, memory is saved. Since non-metric maps only represent the topology, sensor uncertainties do not lead to inconsistencies and no extra computation is needed. This makes them very attractive to systems with limited resources operating in large-scale environments.

Hybrid maps combine the best of the two worlds by having a topological map on the top level connecting local metric maps [28, 5, 49]. The edges usually carry information about the transformation between the submaps. Since the local maps are small and limited, uncertainty within the submaps can be ignored or corrected using SLAM in constant runtime. Furthermore, path planning is efficient, but the map still has a high level of detail. However, for the whole map to be consistent, also SLAM has to be performed. For the special case of retracing a previously traveled path, several authors have noted that no consistent global representation is required when using local metric submaps [80, 27, 56]. The robot must only be able to subsequently switch to the previously recorded submaps and localize within those. However, even though computational resources can be saved by omitting global SLAM updates, the overlapping metric submaps used in these state-of-the-art methods mean a significant memory overhead.

Several works focus on retracing learned paths [93, 81, 27], and some of them also implement obstacle avoidance methods [51, 72, 12]. Furgale and Barfoot [27] build local metric submaps of the landmark positions and use those for localization during repeating the path. Ostafew et al. [72] use this method with obstacle avoidance and apply iterative learning for improving the homing performance each time the path is repeated. The approach by Krajník et al. [51] heavily relies on odometry measurements, because the robot learns straight line sequences, where each segment is associated with a landmark map, the initial orientation of the robot and the segment length. For homing, the robot uses the landmark map only for correcting its current heading, but then moves straight until it has traversed the segment according to its odometry measurements. Šegvić et al. [81] propose to organize key-images acquired during learning in an adjacency graph, where the arcs store the two-view geometries of the connected images, which contain the rotation and translation between the corresponding image positions and a 3D reconstruction of the common features. For homing, a visual servoing approach is used. Cherubini and Chaumette [12] represent the robot's path as a sequence of key images, so that successive images contain common static features. The robot is then controlled to align the centroids of the matched common features in the current and the goal image. Obstacle avoidance is achieved by a laser range scanner.

1.2. Approach

In this thesis, the homing problem is divided into a local and a global navigation task, which have very different requirements. The local navigation task aims at detecting and avoiding obstacles, which requires accurate metric information about the surrounding terrain. Since these configurations can change and will be accessible when the place is visited again, the robot does not have to retain the local terrain properties for later reuse. Rather, in its global map the robot only needs to store the information necessary for retracing the previously traversed path. This task is solved without metric knowledge of the environment by only memorizing the appearance of the environment along the path. Thus, in contrast to the existing works mentioned above, our approach decouples the metric information used in the local navigation task from the global navigation task. Global path learning and homing is only based on bearing angles to landmarks and does not use any distance information, neither robot odometry nor landmark range measurements. Therefore, the proposed global navigation method is computationally very efficient and also applicable to robots which do not provide pose estimates, but employ a reactive obstacle avoidance strategy. An illustration of the proposed navigation scheme is presented in Fig. 1.1.

Fig. 1.2 shows an overview of the proposed navigation method. The local navigation layer serves for detecting and avoiding obstacles in the immediate surroundings of the robot. For this, the robot builds a grid map from stereo disparity images using local pose estimates to register the single-view maps. These pose estimates are derived by fusing visual odometry, IMU and odometry measurements to be robust against slip and visual disturbances. Since in rough terrain there is no binary distinction between obstacles and free space, the robot

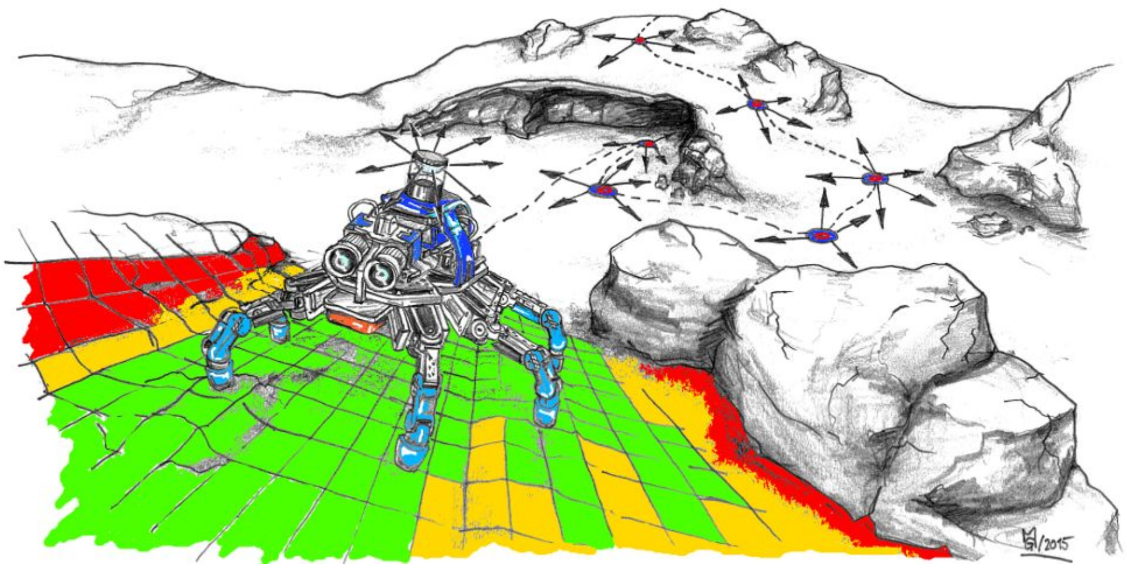


Figure 1.1.: An artist's view of path learning for visual homing in unstructured terrain: The robot is equipped with an omnidirectional camera providing a 360° panorama of the environment. The robot extracts landmarks from the panorama images and stores their bearing angle configurations at certain locations along the path. Additionally, it maintains a moving metric grid map created from its stereo camera to estimate the terrain traversability and avoid obstacles (red cells). Drawing courtesy to Martin Görner.

assesses the traversability of the terrain as a continuous cost value. For this, the robot analyses the perceived terrain geometry with respect to its kinematic abilities. The path planner takes the estimated traversability costs into account for planning short and safe paths in a given direction. Based on the path, the local navigation method generates appropriate motion commands and sends them to the robot's motion layer.

In contrast, the task of the global navigation layer is to provide information for the robot to know where it came from and how it can find its way back to where it started or to any other important place it has visited before. For this, the robot extracts landmarks from its omnidirectional camera and stores them in so-called *viewframes* along its path, which are unique configurations of landmarks with their bearing angles and descriptors at a certain location in space[1]. Since global information can grow unlimited with the length of the traversed path,

[1]To point out the difference to the term *keyframe*, which is also commonly used in robot navigation methods, we should mention that a keyframe usually stands for a full camera image recorded at a certain location. In contrast, a viewframe represents the configuration of extracted landmarks and is, thus, a more compressed description of a specific place in the environment.

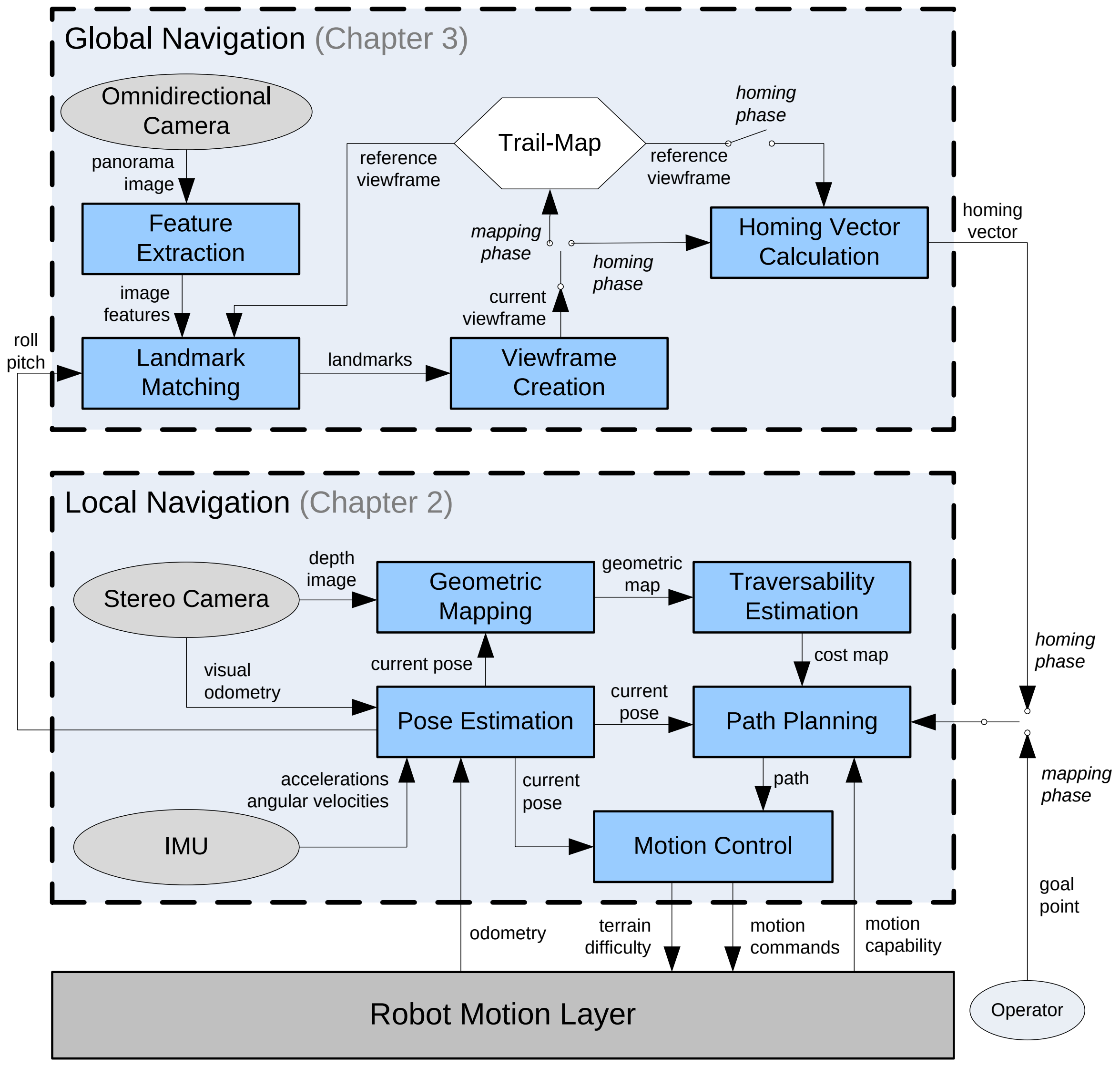


Figure 1.2.: Navigation Overview

the representation of the path must be scalable. Hence, the viewframes are stored in a novel data structure, called *Trail-Map* (*Tra*nslation *I*nvariance *L*evel *Map*), in an efficient way to avoid redundancies and to allow easy scaling of the map. For homing using the recorded path, the robot successively compares the current landmark configuration to the viewframes in the map and computes homing vectors, which give the direction for the robot to reach the home position.

1.3. Contributions

Robust Pose Estimation

We present a method for robust pose estimation by fusing visual odometry, leg odometry and IMU measurements using an indirect information filter. The method correctly considers the relative character of odometry measurements, so that the estimated covariance matrix of the pose estimate shows the correct behavior. Furthermore, the method takes visual odometry errors into account by differently weighting visual odometry measurements depending on the visual conditions. The performance of the data fusion process is evaluated in experiments with the DLR Crawler, a six-legged walking robot. We show that the pose estimate is robust against strong visual disturbances and against slip on loose terrain.

Dense Mapping from Disparity Images

We adapt of the locus method presented by Kweon and Kanade [53] to work for the creation of dense local metric maps from stereo disparity images. Using this method, dense elevation maps of arbitrary resolution can be created, irrespective of the resolution of the disparity images.

Development and Evaluation of a Complete Local Navigation Method for Unknown, Unstructured Terrain

The robust pose estimates and the dense elevation maps are used for creating a complete local navigation method for unknown, unstructured terrain. For this, they are combined with the terrain traversability estimation method, path planner and motion control method developed by Chilian [13]. In an experimental evaluation we show that it allows accurate local metric navigation in unknown, unstructured terrain, even under the presence of visual disturbances.

Development and Evaluation of the Trail-Map for Homing

We develop the Trail-Map, which is a novel data structure for storing landmark configurations for visual homing that allows easy scaling. The Trail-Map is inspired from the LT-Map

(Landmark-Tree Map) [2]. We show that the Trail-Map outperforms the LT-Map in terms of navigation performance, memory requirements and runtime, because the level structure of the Trail-Map is more appropriate for representing a map consisting of viewframes than the tree-structure of the LT-Map. The Trail-Map is created, used and scaled in constant time, independent of the length of the path, which makes it very attractive to mobile robots with limited computational resources. We present thorough evaluations of the Trail-Map in simulations with and without sensor noise, as well as in experiments with a wheeled robot in indoor and outdoor environments. Furthermore, we show that visual homing based on the Trail-Map requires magnitudes less memory than other state-of-the-art visual homing methods, while also offering runtimes constant with respect to the length of the path.

Improvement of the Difference Vector Method

We present improvements of the difference vector method for homing vector calculation presented by Lambrinos et al. [54] for the case of non-isotropic landmark distributions, which occur in strongly pruned Trail-Maps. The improved difference vector model results in straighter homing paths, and its normalized variant is additionally robust against false landmark matches.

Hybrid Navigation Method

We combine the local metric navigation method with the global topological navigation method based on the Trail-Map data structure and present a suitable obstacle avoidance method. The hybrid navigation performance is demonstrated in real-world experiments.

Implementation of the Whole Development Chain from Method Design to Experimental Validation

All methods which are developed in this thesis are experimentally evaluated with real robotic systems in real-world environments. The Trail-Map is additionally evaluated in different simulations, starting from perfect noise-free measurements to modelling image and odometry noise, before validating the method in real-world indoor and outdoor experiments.

1.4. Publications

There are several publications of which I am the first author or a co-author[2] and which are related to this thesis. The works are classified into publications that contain the main contributions of this thesis, and publications that contain related work.

Publications containing the key contributions of this thesis:

- A. Chilian, H. Hirschmüller, and M. Görner. Multisensor data fusion for robust pose estimation of a six-legged walking robot. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS) 2011*, pages 2497–2504, 2011 [15]: This paper contains the robust pose estimation method by fusing inertial measurements with visual odometry and leg odometry. Large parts of the text and figures created by myself are literally reused in this thesis in Section 2.1.
- A. Stelzer, H. Hirschmüller, and M. Görner. Stereo-vision-based navigation of a six-legged walking robot in unknown rough terrain. *The International Journal of Robotics Research*, 31(4):381–402, 2012 [86]: This paper describes the adaption of the locus method [53] for creating dense maps from disparity images. Furthermore, it shows the combination of pose estimation, dense metric mapping, traversability estimation and path planning, and includes the experimental evaluation of the overall navigation method. Large parts of the text and figures created by myself are literally reused in this thesis in the sections 2.2-2.6.
- A. Stelzer, E. Mair, and M. Suppa. Trail-Map: A scalable landmark data structure for biologically inspired range-free navigation. In *Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO) 2014, Bali, Indonesia*, pages 2138–2145, 2014 [87]: This paper contains the development of the Trail-Map data structure as improvement of the LT-Map (Landmark-Tree Map) [2, 63] and compares their performances in simulations in a noise-free environment. Large parts of the text and figures created by myself are literally reused in this thesis in the sections 3.4-3.6.

[2]Some of the publications appeared 2011 or earlier under my maiden name "Chilian".

- A. Stelzer, M. Suppa, and W. Burgard. Trail-Map-based homing under the presence of sensor noise. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2015*, pages 929–936, 2015 [88]: This paper contains the simulation of Trail-Map-based homing under the presence of sensor noise. Large parts of the text and figures created by myself are literally reused in this thesis in the sections 3.7-3.8.

Publications, of which I am an author and which are related to the topic of this thesis, but are not considered as key contributions of this thesis:

- A. Chilian and H. Hirschmüller. Stereo camera based navigation of mobile robots on rough terrain. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2009*, pages 4571–4576, 2009 [14]: This paper contains the straightforward mapping approach from disparity images, the traversability estimation, path planning and motion control method which is used in this work. Parts of the text and figures created by myself are literally reused in sections 2.2-2.5. These methods were developed during my diploma thesis work and are only restated briefly in this thesis for the sake of completeness.

- M. Görner, A. Chilian, and H. Hirschmüller. Towards an autonomous walking robot for planetary surfaces. In *Proceedings of the 10th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, September 2010 [34]: This paper contains the application of the local metric navigation method to the DLR Crawler, which could select the appropriate gaits according to the estimated traversability costs. Minor parts of the text and figures created by myself are literally reused in this thesis in sections 2.2-2.5.

- M. Augustine, E. Mair, A. Stelzer, F. Ortmeier, D. Burschka, and M. Suppa. Landmark-Tree Map: A biologically inspired topological map for long-distance robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO) 2012, Guangzhou, China*, 2012 [2]: This paper introduces the LT-Map as scalable data structure for viewframe-based homing.

- M. Görner and A. Stelzer. A leg proprioception based 6 DOF odometry for statically stable walking robots. *Autonomous Robots*, 34(4):311–326, 2013 [31]: This paper describes the calculation of the 6 degrees of freedom (DOF) leg odometry for the DLR Crawler, which is used for robust pose estimation.

- E. Mair, M. Augustine, B. Jäger, A. Stelzer, C. Brand, D. Burschka, and M. Suppa. A biologically inspired navigation concept based on the Landmark-Tree Map for efficient long-distance robot navigation. *Advanced Robotics*, 28(5):289–302, 2014 [63]: This paper describes the LT-Map as scalable data structure for viewframe-based homing and shows experimental results. Minor parts of the text created by myself are literally reused in Section 3.3.

1.5. Thesis Overview

This thesis is divided into three main parts. The first part is the description of the local metric navigation method presented in Chapter 2. It includes the robust pose estimation by fusing inertial measurements with visual odometry and leg odometry and the creation of dense local maps for traversability estimation and local path planning. The second part is the global topological navigation method described in Chapter 3. This chapter explains the viewframe-based navigation concept, introduces the novel data structure Trail-Map as improvement of the LT-Map and analyzes its behavior in simulations and real-world experiments. The third main part (Chapter 4) is the combination of the local metric and the global topological navigation methods to a complete navigation system for visual homing and analyses its performance in real-world experiments. Finally, Chapter 5 concludes this thesis and gives an overview of future perspectives.

2. Local Metric Navigation

The local part of the navigation algorithm is intended to work in a small area around the current position of the robot. It has the objective to lead the robot along a safe and obstacle-free path in the direction specified by the global navigation algorithm.

In semi-structured or structured environments, we can assume a flat ground plane. In that case, obstacles are objects with a certain extent above or below the ground plane. However, since this assumption does not hold in natural, rough outdoor terrain, we have to revise the definition of obstacles. In rough terrain a binary distinction between an obstacle and free space is inappropriate, because there are areas which can be traversed more easily than others. Thus, we should rather estimate the traversability of the terrain on a continuous scale between "most easily traversable" and "untraversable". For this, we have to geometrically model the robot's surroundings using the depth images computed by its stereo camera. Since a single camera view is often not sufficient to cover a region that is wide enough for the robot to pass, we combine multiple camera views to a local map. This, in turn, requires a good estimate of the robot's pose at the time of stereo image acquisition. We can achieve that by combining the visual odometry measurements of the stereo camera with inertial measurements and wheel or leg odometry readings. Once the robot has a geometrical representation of its environment, it can assess the traversability costs of the terrain regarding its kinematics and use the resulting cost map to plan paths that lead through safe terrain and take the robot's current motion abilities into account. The local navigation layer sends motion commands and the estimated terrain difficulty to the robot motion layer so that the robot follows the path. Fig. 2.1 shows an overview of the local navigation system.

Dense local mapping and traversability estimation can be computationally expensive processes which also require a large amount of memory. However, the robot requires metric information only in its vicinity. Thus, we can limit the size of the local map to a certain range. This also limits the computational complexity so that it does not grow with the size

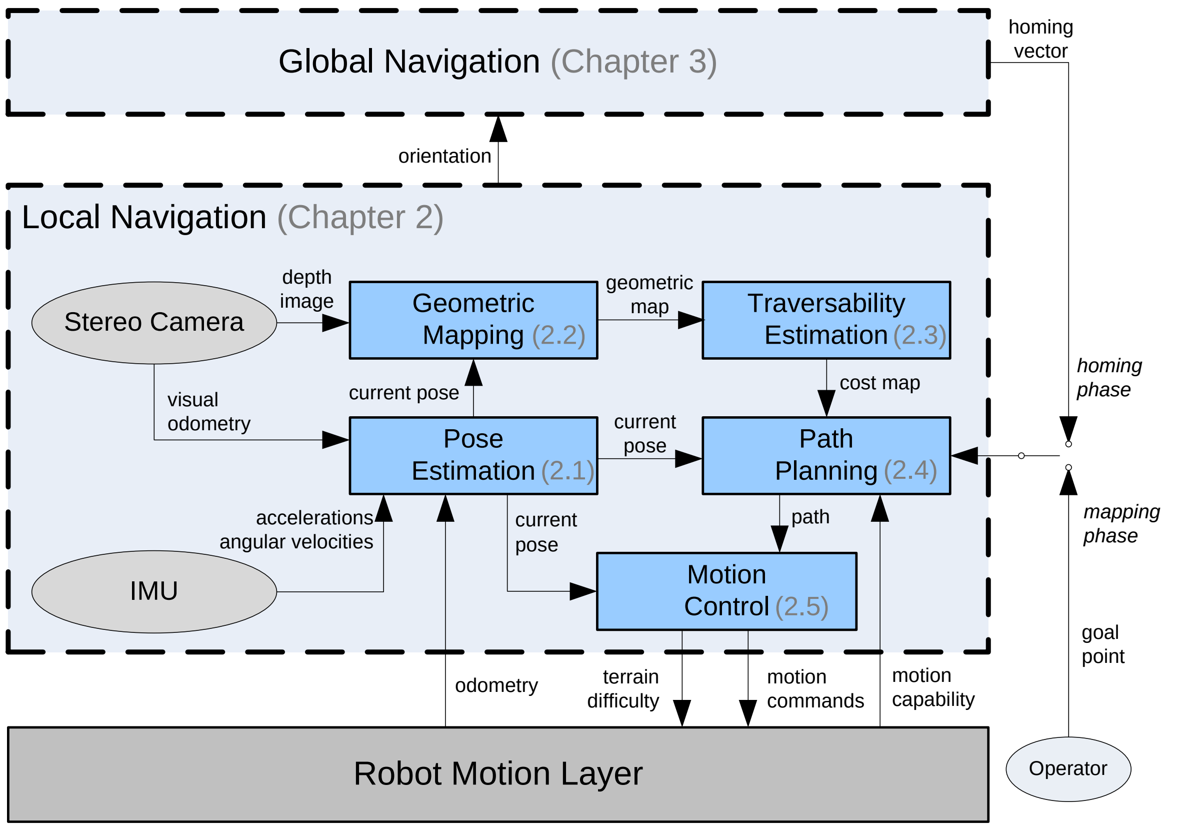


Figure 2.1.: Overview of the local navigation method (corresponding section numbers in brackets)

of the terrain explored by the robot. Hence, we can adjust the map resolution and size to the computational power available to the robot.

In this chapter, we describe a local navigation system for a six-legged walking robot equipped with a stereo camera and an inertial measurement unit (IMU). The walking robot provides a 6 degrees of freedom (DOF) leg odometry and is able to walk with different gait patterns that can be chosen according to the ground conditions. However, the navigation system is also applicable to any other ground-based robot which has a stereo camera and an IMU. In fact, in upcoming chapters we will use it for navigating a wheel-driven Pioneer 3-DX robot.

This chapter is organized as follows: Section 2.1 describes a multisensor data fusion method for calculating a robust pose estimate from IMU, visual odometry and leg odometry measurements. Section 2.2 gives details about local mapping from disparity images based on the estimated pose. In Section 2.3, we explain the process of traversability estimation. Path planning based on the terrain traversability is described in Section 2.4. Section 2.5 gives details about how the robot is commanded to move along the planned path and to explore its environment. We give the results of the experimental evaluation of the local navigation method in Section 2.6 and Section 2.7 concludes this chapter.

2.1. Robust Pose Estimation

Knowing the current pose with respect to a map or any other reference is essential in any navigation task. The robot is equipped with many sensors from whose measurements motion or other hints about the current pose can be inferred. The stereo camera allows the computation of a visual odometry by measuring the motion of corresponding features in subsequent image pairs. The IMU can sense accelerations and turn rates from which velocities, angles and positions can be derived. Leg joint sensors are able to measure the leg configurations from which a leg odometry can be calculated. Wheel encoders give information about the driving speed. All these sensors allow the robot to calculate estimates of its pose or parts of it, but none is free of shortcomings. Visual odometry can only work in well-textured environments and may fail when lighting conditions are bad. Wheel or leg odometry is subject to slip, which can happen frequently in rough terrain. Inferring poses from an IMU yields good results over short time periods, but is subject to unbounded drift in the long run. Hence, for a robust and accurate pose estimate, we have to fuse the information of all available sensors.

2.1.1. State of the Art

Many authors have addressed the problem of sensor data fusion. Often IMU data is combined with GPS readings because of their complementary properties. A multisensor Kalman filter for fusing IMU and GPS data was presented by Caron et al. [9]. Dissanayake et al. [19] presented an indirect information filter for fusing GPS, IMU and wheel encoder data. However, on planetary surfaces or in caves, GPS is not available.

Other authors do not use GPS data but combine inertial measurements with visual odometry and/or odometry information. Konolige et al. [48] showed a data fusion method using an extended Kalman filter for correcting visual odometry measurements by IMU roll and pitch angle measurements with respect to the gravity vector, and by IMU angular rate measurements for the yaw angle. Helmick et al. [37] used the indirect extended Kalman filter formulation to fuse IMU measurements with relative measurements from visual odometry and vehicle odometry. Lamon and Siegwart [55] used an extended information filter for fusing data from an inertial navigation system and a three-dimensional rover odometry. Apart from Helmick et al. [37], these works neglect that odometry measurements (visual as well as wheel or leg odometry) are relative measurements and therefore have to be treated carefully in Kalman filter frameworks. To overcome this problem, relative pose measurements can be treated as velocities. However, this only yields satisfactory results when the relative pose measurements have a very high rate and, hence, the velocity is approximated correctly. The correct solution is to augment the state vector in the Kalman filter framework by the states that are part of the relative measurements as described by Roumeliotis and Burdick [76] and termed *stochastic cloning*. Schmid et al. [79] used stochastic cloning for estimating the state of a highly-dynamic quadrotor with time-delayed relative sensor measurements in a Kalman filter framework.

2.1.2. Approach

In our application, we want to compute a pose estimate from the robot's IMU data, its stereo visual odometry and, if available, its leg odometry. For this, we chose to use an *indirect feedback information filter*. The *information filter* has the advantage that we can fuse measurements of multiple sensors at the same time very easily. The *indirect* or *error state* form works on an error state vector which contains the errors of the actual state rather than the state variables themselves. The advantage is that we do not need to model the usually nonlinear robot dynamics, but the filter is based on linear equations describing the error propagation in the inertial system. The *feedback* formulation means that the estimated error is fed back into the IMU navigation equations to correct the current position, velocity and orientation estimates. Hence, the estimated error states remain small and small angle approximations in the filter equations are possible. This also means that we can predict the error state as zero for each new filter step. Furthermore, the indirect filter formulation allows the filter to run at a lower frequency than the inertial navigation equations. Roumeliotis et al. [77] give a more detailed discussion of the different filter formulations.

The information filter is numerically equivalent to the Kalman filter but has inverse complexity properties. In particular, while the prediction step of the Kalman filter is computationally simple and the update step is complex, the information filter equations yield a complex prediction step and a computationally cheap update step. A good explanation of the information filter is given by Dissanayake et al. [19] and can be summarized as follows.

For transforming the indirect Kalman filter into the information form, we define the information matrix $\boldsymbol{Y}$ and the error information vector $\Delta\boldsymbol{y}$ as

$$\boldsymbol{Y} = \boldsymbol{P}^{-1} \qquad \text{and} \qquad \Delta\boldsymbol{y} = \boldsymbol{Y} \cdot \Delta\boldsymbol{x}, \tag{2.1}$$

where $\boldsymbol{P}$ is the estimation covariance matrix and $\Delta\boldsymbol{x}$ is the error state vector. We can transform the Kalman filter equations such that $\boldsymbol{Y}$ and $\Delta\boldsymbol{y}$ are estimated, which results in the prediction step

$$\boldsymbol{Y}_t^- = (\boldsymbol{A}_t\boldsymbol{Y}_{t-1}^{-1}\boldsymbol{A}_t^T + \boldsymbol{Q}_t^p)^{-1} \tag{2.2}$$
$$\Delta\boldsymbol{y}_t^- = \boldsymbol{Y}_t^-(\boldsymbol{A}_t\boldsymbol{Y}_{t-1}^{-1}\Delta\boldsymbol{y}_{t-1}), \tag{2.3}$$

where $\boldsymbol{A}_t$ is the state transition matrix and $\boldsymbol{Q}_t^p$ is the process noise matrix. In the feedback form, we can simplify the prediction in Eq. (2.3) to $\Delta\boldsymbol{y}_t^- = 0$, because we correct the error after each filter step. The update step of the information filter becomes

$$\boldsymbol{Y}_t = \boldsymbol{H}_t^T(\boldsymbol{Q}_t^m)^{-1}\boldsymbol{H}_t + \boldsymbol{Y}_t^- \tag{2.4}$$
$$\Delta\boldsymbol{y}_t = \boldsymbol{H}_t^T(\boldsymbol{Q}_t^m)^{-1}z_t + \Delta\boldsymbol{y}_t^-, \tag{2.5}$$

where $\boldsymbol{H}_t$ is the measurement matrix and $\boldsymbol{Q}_t^m$ is the measurement noise matrix. In the indirect formulation, the measurement vector z_t is the difference between the IMU measurements and the measurements of an aiding sensor. We can write the update step as

$$\boldsymbol{Y}_t = \boldsymbol{I}_t + \boldsymbol{Y}_t^- \quad \text{with} \quad \boldsymbol{I}_t = \boldsymbol{H}_t^T(\boldsymbol{Q}_t^m)^{-1}\boldsymbol{H}_t, \tag{2.6}$$
$$\Delta\boldsymbol{y}_t = \boldsymbol{i}_t + \Delta\boldsymbol{y}_t^- \quad \text{with} \quad \boldsymbol{i}_t = \boldsymbol{H}_t^T(\boldsymbol{Q}_t^m)^{-1}z_t. \tag{2.7}$$

The term $\boldsymbol{I}_t$ is the amount of information in the measurement and $\boldsymbol{i}_t$ is the contribution of the measurement z_t to the state vector. If there are several measurements $z_{k,t}$ at a time step t we get

$$\boldsymbol{I}_t = \sum_{k=1}^{n} \boldsymbol{H}_{k,t}^T(\boldsymbol{Q}_{k,t}^m)^{-1}\boldsymbol{H}_{k,t} = \sum_{k=1}^{n} \boldsymbol{I}_{k,t} \tag{2.8}$$
$$\boldsymbol{i}_t = \sum_{k=1}^{n} \boldsymbol{H}_{k,t}^T(\boldsymbol{Q}_{k,t}^m)^{-1}z_{k,t} = \sum_{k=1}^{n} \boldsymbol{i}_{k,t}. \tag{2.9}$$

The simplicity of the update stage of the information filter originates from the fact that the measurements of the single sensors are conditionally independent. Hence, the information form of the Kalman filter has computational advantages for multisensor data fusion. The routines for computing $\boldsymbol{I}_{k,t}$ and $\boldsymbol{i}_{k,t}$ for each measurement are independent of each other and independent of $\boldsymbol{Y}_t^-$ and $\Delta\boldsymbol{y}_t^-$, which allows them to run in parallel and on distributed systems. The disadvantage is that we have to perform a matrix inversion to obtain the error state vector $\Delta\boldsymbol{x}_t$ from the information vector $\Delta\boldsymbol{y}_t$. However, the more external sensors are used, the higher the benefit of using the information filter.

2.1.3. State Vector and State Transition Model

For implementing the information filter we use a state vector consisting of 15 variables: The position $\boldsymbol{p}$ (3), the velocity $\boldsymbol{v}$ (3), the orientation Euler angles $\boldsymbol{\varphi}$ (3), the bias of the gyroscopes $\boldsymbol{b}_g$ (3) and the bias of the accelerometers $\boldsymbol{b}_a$ (3). In the indirect formulation we use the error state vector

$$\Delta\boldsymbol{x} = (\Delta\boldsymbol{p}, \Delta\boldsymbol{v}, \Delta\boldsymbol{\varphi}, \Delta\boldsymbol{b}_g, \Delta\boldsymbol{b}_a)^T. \tag{2.10}$$

The position $\boldsymbol{p}$ and velocity $\boldsymbol{v}$ variables are given in world coordinates with the origin located at the IMU origin at the beginning of the data fusion process. The Euler angles $\boldsymbol{\varphi}$ are the angles of the rotation matrix that turns a point from the IMU coordinate frame to the world coordinate frame. The bias values $\boldsymbol{b}_g$ and $\boldsymbol{b}_a$ are given in IMU coordinates.

The use of Euler angles for representing the orientation of the robot is valid in this application, because configurations which cause the Euler angle gimbal lock problem (such as 90° pitch) will not be reached by a ground-based robot during regular operation. We chose Euler angles because they provide an intuitive representation of orientation. For applications in which gimbal lock can occur, representations such as rotation vector or quaternions should be used. However, in the error state vector, the orientation error $\Delta\boldsymbol{\varphi}$ always contains small Euler angles, which are, thus, equivalent to the components of a rotation vector. This can easily be shown by applying small angle approximation when computing a rotation matrix from Euler angles and from a rotation vector.

The discrete time error state propagation originates from the inertial error dynamics [95] as

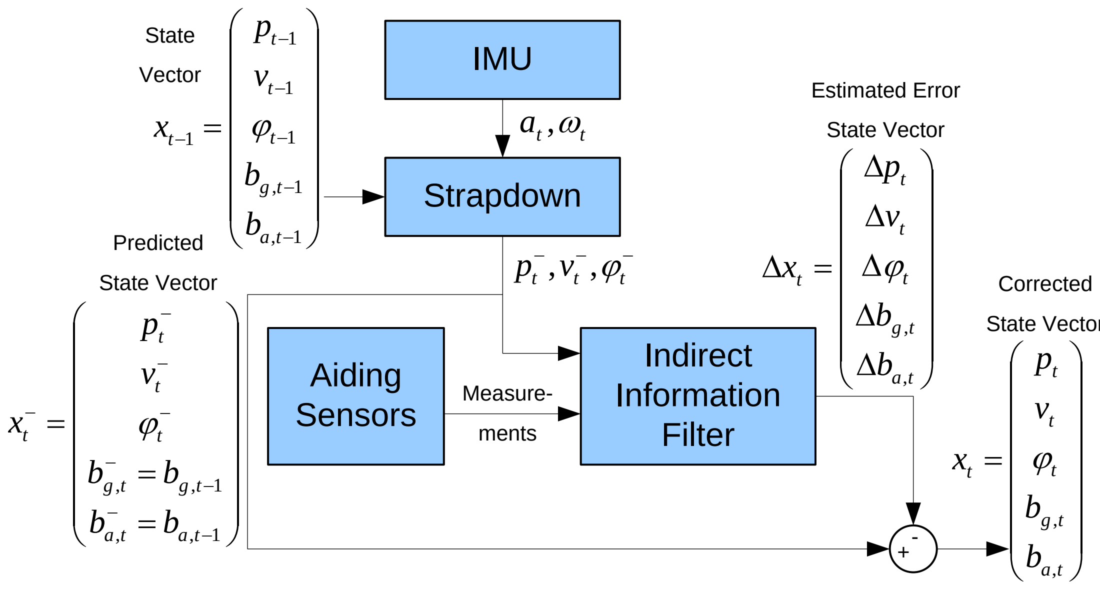
$$\Delta \boldsymbol{x}_t^- = \boldsymbol{A}_t \cdot \Delta \boldsymbol{x}_{t-1} \tag{2.11}$$

$$\boldsymbol{A}_t = \mathbf{I} - \begin{bmatrix} \mathbf{0} & -\mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{R}_t^- \lfloor (\boldsymbol{a}_t - \boldsymbol{b}_{a,t}^-) \times \rfloor & \mathbf{0} & \boldsymbol{R}_t^- \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{R}_t^- & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \Delta t \tag{2.12}$$

$$\lfloor \boldsymbol{o} \times \rfloor = \begin{bmatrix} 0 & -o_z & o_y \\ o_z & 0 & -o_x \\ -o_y & o_x & 0 \end{bmatrix}, \tag{2.13}$$

where $\mathbf{I}$ is the identity matrix (not to be confused with the information amount $\boldsymbol{I}_t$), $\boldsymbol{a}_t = (a_{tx}, a_{ty}, a_{tz})^T$ is the acceleration measured by the IMU, $\boldsymbol{b}_{a,t}^-$ is the predicted accelerometer bias, $\boldsymbol{R}_t^-$ is the propagated rotation from the IMU coordinate frame to the world coordinate frame and Δt is the time difference between $t-1$ and t.

2.1.4. The Multisensor Data Fusion Process



State
Vector
$x_{t-1} = (p_{t-1}, v_{t-1}, \varphi_{t-1}, b_{g,t-1}, b_{a,t-1})$
IMU
a_t, ω_t
Strapdown
$p_t^-, v_t^-, \varphi_t^-$
Predicted
State Vector
$x_t^- = (p_t^-, v_t^-, \varphi_t^-, b_{g,t}^- = b_{g,t-1}, b_{a,t}^- = b_{a,t-1})$
Aiding
Sensors
Measure-
ments
Indirect
Information
Filter
Estimated Error
State Vector
$\Delta x_t = (\Delta p_t, \Delta v_t, \Delta \varphi_t, \Delta b_{g,t}, \Delta b_{a,t})$
Corrected
State Vector
$x_t = (p_t, v_t, \varphi_t, b_{g,t}, b_{a,t})$
+
-

Figure 2.2.: Overview of the multisensor data fusion process. The measurements from the aiding sensors are the absolute roll and pitch angles derived from the IMU accelerations, and the visual odometry and/or leg odometry measurements.

Fig. 2.2 gives an overview of one time step of the data fusion process. First, we feed the accelerations $\boldsymbol{a}_t$ and angular velocities $\boldsymbol{\omega}_t$ measured by the IMU into the strapdown algorithm. Considering the state vector $\boldsymbol{x}_{t-1}$ from the previous filter step, this algorithm integrates the IMU measurements to velocity $\boldsymbol{v}_t^-$, position $\boldsymbol{p}_t^-$ and orientation Euler angles $\boldsymbol{\varphi}_t^-$. These values are the predicted state variables. We chose to predict the bias values $\boldsymbol{b}_{a,t}^-$ and $\boldsymbol{b}_{g,t}^-$ to be equal to the bias values of the last filter step, because they change only slowly. Every time one or more measurements of the aiding sensors are available, the indirect information filter computes an estimated error state vector $\Delta\boldsymbol{x}_t$. We subtract this error state vector from the predicted state vector $\boldsymbol{x}_t^-$ to feedback the error. The result is the corrected state vector $\boldsymbol{x}_t$. If no measurements of the aiding sensors are available, the error state vector is zero and the corrected state will be the predicted state. The following sections will describe the strapdown block and the information filter block in more detail.

The Strapdown Algorithm

The accelerations and angular velocities of the IMU are measured in the IMU coordinate frame. Since the IMU moves, we have to transform the accelerations into the world coordinate system before integrating them. For this, we need the rotation matrix $\boldsymbol{R}_t$, which turns a vector from the IMU coordinate frame to the world coordinate frame.

We compute the propagated rotation matrix $\boldsymbol{R}_t^-$ from the rotation matrix $\boldsymbol{R}_{t-1}$ of the last time step and a differential rotation $\boldsymbol{R}_{\Delta,t}$. Assuming a high sampling rate (Δt is small), we get $\boldsymbol{R}_{\Delta,t}$ from the gyroscope measurements $\boldsymbol{\omega}_t$ as follows [8]:

$$\boldsymbol{R}_t^- = \boldsymbol{R}_{t-1}\boldsymbol{R}_{\Delta,t} \tag{2.14}$$

$$\boldsymbol{R}_{\Delta,t} = \mathbf{I} + \frac{\sin|\boldsymbol{\phi}_t|}{|\boldsymbol{\phi}_t|}\lfloor\boldsymbol{\phi}_t\times\rfloor + \frac{1-\cos|\boldsymbol{\phi}_t|}{|\boldsymbol{\phi}_t|^2}\lfloor\boldsymbol{\phi}_t\times\rfloor^2 \tag{2.15}$$

$$\boldsymbol{\phi}_t = (\boldsymbol{\omega}_t - \boldsymbol{b}_{g,t}^-)\Delta t \tag{2.16}$$

$$|\boldsymbol{\phi}_t| = \sqrt{\phi_{x,t}^2 + \phi_{y,t}^2 + \phi_{z,t}^2}. \tag{2.17}$$

The variable $\boldsymbol{\phi}_t$ is the rotation vector.

Knowing the rotation matrix, we can compute the IMU velocity $\boldsymbol{v}_t^-$ and position $\boldsymbol{p}_t^-$. For this, we first have to compensate the acceleration measurements $\boldsymbol{a}_t$ for bias $\boldsymbol{b}_{a,t}^-$. After that, we can transform them into the world frame using $\boldsymbol{R}_t^-$ and compensate the gravity vector[1] $\boldsymbol{g} = (0, 0, -|\boldsymbol{g}|)^T$:

$$\boldsymbol{v}_t^- = \boldsymbol{v}_{t-1} + (\boldsymbol{R}_t^-(\boldsymbol{a}_t - \boldsymbol{b}_{a,t}^-) + \boldsymbol{g})\Delta t \tag{2.18}$$

$$\boldsymbol{p}_t^- = \boldsymbol{p}_{t-1} + \boldsymbol{v}_{t-1}\Delta t + \tfrac{1}{2}(\boldsymbol{R}_t^-(\boldsymbol{a}_t - \boldsymbol{b}_{a,t}^-) + \boldsymbol{g})\Delta t^2. \tag{2.19}$$

Stochastic Cloning

Within the indirect information filter we use relative and absolute measurements to compute the estimated error state vector. While absolute measurements only depend on the current state of the system, relative measurements contain a difference between the current system state and a previous state. Since Kalman filter theory assumes that a measurement only depends on the current state of the system, we have to treat relative measurements in a special way. When we ignore this fact, the data fusion filter might also give good results at the first glance. However, viewed more closely, the resulting estimated variances are not feasible: For example, when fusing IMU data only with relative position measurements, we would expect that the estimated position variance grows over time, because summing up relative position measurements results in a drifting position estimate. However, if we ignore the relative character of the position measurements, the resulting position variances are estimated small and constant over time. This can cause serious problems when using another position sensor such as GPS, which gives absolute but noisy position measurements. These measurements will not influence the estimated position to the expected extent because the position estimate after fusing only the relative measurements is overconfident. Even if no absolute measurement was available for a longer time and position drift is significant, the estimated position variance would be small.

To avoid this problem, we have to augment the state vector and covariance matrix to also contain the previous state which is part of the relative measurement. This approach was described by Roumeliotis and Burdick [76] and termed *stochastic cloning*. This method introduces the correlations between the current and the previous state and hence allows to estimate a correct covariance matrix, with growing variances over time if only relative measurements are available.

To keep the augmented covariance matrix small, we only clone the covariances associated to the states $\boldsymbol{p}_t$ and $\boldsymbol{\varphi}_t$, because we only use relative position and rotation measurements. At

[1]The value for $|\boldsymbol{g}|$ should be chosen according to the location.

each time $t = t_{\text{Start}}$ when at least one relative measurement starts, we augment the covariance matrix as follows:

$$\check{\boldsymbol{x}}_t = \begin{bmatrix} \boldsymbol{p}_t \\ \boldsymbol{\varphi}_t \end{bmatrix} \qquad \check{\boldsymbol{P}}_t = \mathbf{Cov}(\check{\boldsymbol{x}}_t, \check{\boldsymbol{x}}_t) \tag{2.20}$$

$$\boldsymbol{P}_t^{\text{aug}} = \begin{bmatrix} \check{\boldsymbol{P}}_{t_{\text{Start}}} & \mathbf{Cov}(\check{\boldsymbol{x}}_{t_{\text{Start}}}, \boldsymbol{x}_t) \\ \mathbf{Cov}(\boldsymbol{x}_t, \check{\boldsymbol{x}}_{t_{\text{Start}}}) & \boldsymbol{P}_t \end{bmatrix}, \tag{2.21}$$

where $\mathbf{Cov}(\boldsymbol{x}_t, \check{\boldsymbol{x}}_{t_{\text{Start}}})$ is the covariance between the states at time t and the cloned states at t_{Start}. Since the covariance $\check{\boldsymbol{P}}_{t_{\text{Start}}}$ must not change during prediction of the filter, the system matrix $\boldsymbol{A}_t^{\text{aug}}$ and the process noise matrix $\boldsymbol{Q}_t^{p,\text{aug}}$ become

$$\boldsymbol{A}_t^{\text{aug}} = \mathbf{blkdiag}\left[\mathbf{I}, \boldsymbol{A}_t\right] \tag{2.22}$$

$$\boldsymbol{Q}_t^{p,\text{aug}} = \mathbf{blkdiag}\left[\mathbf{0}, \boldsymbol{Q}_t^p\right], \tag{2.23}$$

where $\mathbf{blkdiag}\,[\boldsymbol{U}, \boldsymbol{V}]$ stands for a block diagonal matrix with the matrices $\boldsymbol{U}, \boldsymbol{V}$ on its main diagonal.

Since the information filter uses the inverse covariance, we must ensure that $\boldsymbol{A}_t^{\text{aug}} \boldsymbol{P}_t^{\text{aug}} (\boldsymbol{A}_t^{\text{aug}})^T + \boldsymbol{Q}_t^{p,\text{aug}}$ is invertible in the prediction step in Eq. (2.2). For that reason, if two different relative measurements start at the same time, we apply cloning only once to keep the covariance matrix full rank. If measurements start at different times, we also have to clone the covariances between the different previous states correctly. After processing a relative measurement, we delete the corresponding covariances from the augmented covariance matrix because they are not needed any longer. However, in our application, usually a relative measurement starts at the same time the previous measurement ends. Thus, after deleting a previous state, we clone the current state for augmenting the covariance matrix again. In our case, relative measurements from visual odometry and leg odometry are taken at different rates. Thus, the augmented state vector usually contains two different previous positions and orientations, each corresponding to the starting time of a relative measurement.

The Indirect Information Filter

Fig. 2.3 illustrates the data flow within the indirect information filter for multisensor data fusion. First, we predict the augmented information matrix $\boldsymbol{Y}_t^{\text{aug-}}$. Then, we compute the values for $\boldsymbol{i}_{k,t}^{\text{aug}}$ and $\boldsymbol{I}_{k,t}^{\text{aug}}$ for each available sensor measurement k at time step t using the

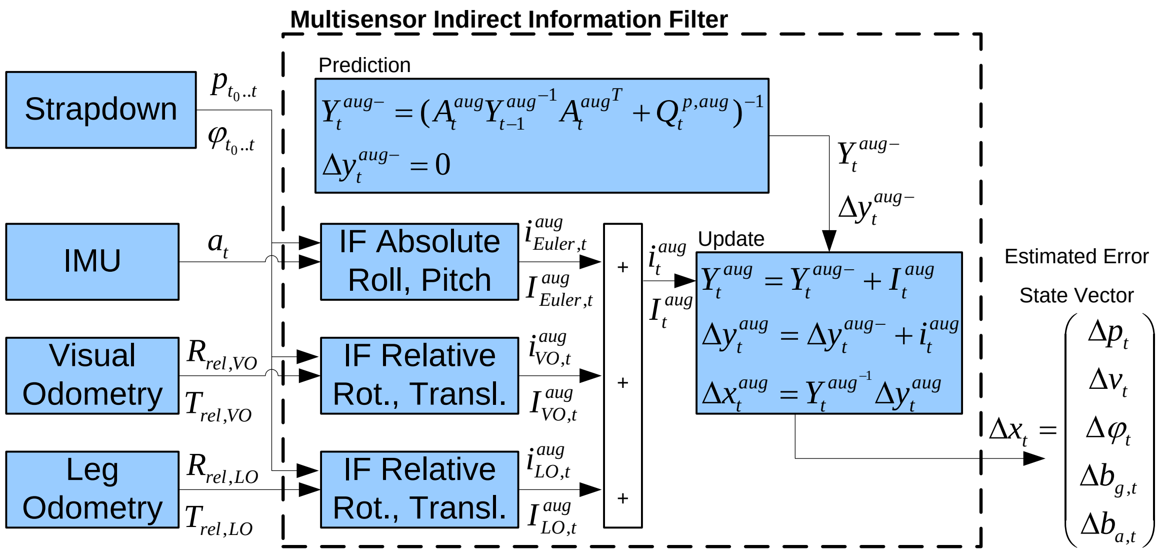

Multisensor Indirect Information Filter
Prediction
$Y_t^{aug-} = (A_t^{aug} Y_{t-1}^{aug^{-1}} A_t^{aug^T} + Q_t^{p,aug})^{-1}$
$\Delta y_t^{aug-} = 0$
Strapdown
$p_{t_0..t}$
$\varphi_{t_0..t}$
IMU
a_t
Visual Odometry
$R_{rel,VO}$
$T_{rel,VO}$
Leg Odometry
$R_{rel,LO}$
$T_{rel,LO}$
IF Absolute Roll, Pitch
IF Relative Rot., Transl.
IF Relative Rot., Transl.
$i_{Euler,t}^{aug}$
$I_{Euler,t}^{aug}$
$i_{VO,t}^{aug}$
$I_{VO,t}^{aug}$
$i_{LO,t}^{aug}$
$I_{LO,t}^{aug}$
+
+
+
i_t^{aug}
I_t^{aug}
Y_t^{aug-}
Δy_t^{aug-}
Update
$Y_t^{aug} = Y_t^{aug-} + I_t^{aug}$
$\Delta y_t^{aug} = \Delta y_t^{aug-} + i_t^{aug}$
$\Delta x_t^{aug} = Y_t^{aug^{-1}} \Delta y_t^{aug}$
Estimated Error
State Vector
$\Delta x_t = \begin{pmatrix} \Delta p_t \\ \Delta v_t \\ \Delta \varphi_t \\ \Delta b_{g,t} \\ \Delta b_{a,t} \end{pmatrix}$

Figure 2.3.: Overview of the multisensor data fusion information filter. IF: Information filter.

differences between the strapdown algorithm results and the sensor measurements. After that, we sum up all available information amounts $\boldsymbol{I}_{k,t}^{\text{aug}}$ and information contributions $\boldsymbol{i}_{k,t}^{\text{aug}}$ and perform the update equations. In the end, we transform the resulting information vector $\Delta\boldsymbol{y}_t^{\text{aug}}$ into the error state vector $\Delta\boldsymbol{x}_t^{\text{aug}}$, from which we delete the cloned states. The following sections describe these steps in more detail.

Prediction Using the state transition matrix $\boldsymbol{A}_t^{\text{aug}}$ as given in Eq. (2.12) and Eq. (2.22), we predict the information matrix $\boldsymbol{Y}_t^{\text{aug}-}$ as in Eq. (2.2). The prediction of the information vector simply becomes $\Delta\boldsymbol{y}_t^{\text{aug}-} = 0$ because in the indirect feedback information filter the error is corrected after each filter step.

Absolute Roll and Pitch Angle Measurements Since the accelerometers of the IMU sense the gravity, which is known in size and direction with respect to the world frame, we can determine the absolute roll and pitch angles γ_{abs} and β_{abs} of the acceleration measurement $\boldsymbol{a} = [a_x, a_y, a_z]^T$ as follows:

$$\gamma_{\text{abs}} = \text{atan2}(a_y, a_z), \tag{2.24}$$

$$\beta_{\text{abs}} = \text{atan2}(-a_x, a_y \sin\gamma_{\text{abs}} + a_z \cos\gamma_{\text{abs}}). \tag{2.25}$$

From the absolute roll and pitch angles, we can compute an absolute rotation matrix $\boldsymbol{R}_{\text{abs}}$ using

$$\boldsymbol{R}_{\text{abs}} = \begin{bmatrix} \text{c}\beta\text{c}\alpha & \text{s}\gamma\text{s}\beta\text{c}\alpha - \text{c}\gamma\text{s}\alpha & \text{c}\gamma\text{s}\beta\text{c}\alpha + \text{s}\gamma\text{s}\alpha \\ \text{c}\beta\text{s}\alpha & \text{s}\gamma\text{s}\beta\text{s}\alpha + \text{c}\gamma\text{c}\alpha & \text{c}\gamma\text{s}\beta\text{s}\alpha - \text{s}\gamma\text{c}\alpha \\ -\text{s}\beta & \text{s}\gamma\text{c}\beta & \text{c}\gamma\text{c}\beta \end{bmatrix},$$
$$\text{s}\varphi = \sin\varphi_{\text{abs}}, \qquad \text{c}\varphi = \cos\varphi_{\text{abs}}. \tag{2.26}$$

For this, we set the yaw angle α_{abs} to be equal to the yaw angle of the propagated rotation matrix $\boldsymbol{R}_t^-$ because it cannot be determined from the acceleration measurements.

The absolute roll and pitch angles obtained from the acceleration measurements contain a high level of noise. The noise is caused by additional accelerations that occur when the robot moves. Hence, we must fuse the absolute noisy angles with low-noise angular measurements. The propagated rotation matrix $\boldsymbol{R}_t^-$ as computed in Eq. (2.14)-Eq. (2.17) contains the roll and pitch angles from integrating the gyroscope measurements. These angles do not suffer from high noise but from a drift caused by integrating the sensor values. By fusing $\boldsymbol{R}_{\text{abs}}$ and $\boldsymbol{R}_t^-$, we can determine the roll and pitch Euler angles quite accurately without drift and high noise. We compute the difference rotation matrix between the propagated rotation $\boldsymbol{R}_t^-$ and the absolute rotation $\boldsymbol{R}_{\text{abs}}$ as

$$\boldsymbol{R}_{\text{diff}} = \boldsymbol{R}_t^- \cdot \boldsymbol{R}_{\text{abs}}^T. \tag{2.27}$$

Using the equations

$$\begin{aligned} \alpha &= \operatorname{atan2}(\boldsymbol{R}_{(2,1)}, \boldsymbol{R}_{(1,1)}) \\ \beta &= \operatorname{atan2}(-\boldsymbol{R}_{(3,1)}, \boldsymbol{R}_{(2,1)}\sin\alpha + \boldsymbol{R}_{(1,1)}\cos\alpha) \\ \gamma &= \operatorname{atan2}(\boldsymbol{R}_{(1,3)}\sin\alpha - \boldsymbol{R}_{(2,3)}\cos\alpha, -\boldsymbol{R}_{(1,2)}\sin\alpha + \boldsymbol{R}_{(2,2)}\cos\alpha) \end{aligned} \tag{2.28}$$

to extract Euler angles from the elements $\boldsymbol{R}_{(i,j)}$ of a rotation matrix, we calculate the angle differences γ_{diff} and β_{diff} from $\boldsymbol{R}_{\text{diff}}$, which give the measurement vector

$$z_{\text{Euler},t} = \begin{bmatrix} \gamma_{\text{diff}} \\ \beta_{\text{diff}} \end{bmatrix}. \tag{2.29}$$

The measurement matrix $\boldsymbol{H}_{\text{Euler},t}$ which projects the state vector $\boldsymbol{x}_t$ onto the measurement vector $z_{\text{Euler},t}$ is

$$\boldsymbol{H}_{\text{Euler},t} = \begin{bmatrix} \mathbf{0}_{2\times6} & \mathbf{I}_{2\times2} & \mathbf{0}_{2\times7}. \end{bmatrix} \tag{2.30}$$

For the augmented state vector, we have to augment the measurement matrix with zeros to

$$\boldsymbol{H}^{\text{aug}}_{\text{Euler},t} = \begin{bmatrix} \mathbf{0} & \boldsymbol{H}_{\text{Euler},t} \end{bmatrix}, \tag{2.31}$$

because the measurement does not depend on any previous states but is absolute. The measurement noise matrix $\boldsymbol{Q}^{m}_{\text{Euler},t}$ contains the variances of the absolute roll and pitch angle measurements and can be found by filter tuning.

Knowing $z_{\text{Euler},t}$, $\boldsymbol{H}^{\text{aug}}_{\text{Euler},t}$, $\boldsymbol{Q}^{m}_{\text{Euler},t}$, we compute the information contribution $\boldsymbol{i}^{\text{aug}}_{\text{Euler},t}$ and the information amount $\boldsymbol{I}^{\text{aug}}_{\text{Euler},t}$ using Eq. (2.6)-Eq. (2.7).

The use of absolute angles obtained by accelerometer data as measurements for the data fusion filter violates Kalman filter theory, which assumes that measurement noise and process noise are uncorrelated. Hence, the filter result is suboptimal. However, the suboptimal filter result is still better than not using absolute roll and pitch angle measurements for limiting the drift of the orientation estimates.

Relative Translation and Rotation Measurements We fuse the relative motion measurements with the relative rotations and translations computed by the strapdown algorithm within the same time period. Visual odometry, as well as leg odometry, provide relative position and orientation measurements between two consecutive images or robot poses, respectively. We fuse visual odometry and leg odometry in the same way as relative translation and rotation measurements. Thus, we will not distinguish them in the next paragraphs, but will refer to them as "odometry sensor".

A relative measurement has two timestamps t_{start} and t_{end} at the beginning and the end of the relative measurement. Furthermore, for fusing relative rotations and translations, all values must be represented in the same coordinate frame. That means, we have to transform the relative measurements of all sensors into relative measurements in the IMU coordinate frame in order to fuse them with IMU measurements. That implies that we know the transformations between the different sensor coordinate frames, either by design or by calibration.

The differences between the relative motion given by the strapdown algorithm in the time interval from t_{start} to t_{end} and the relative motion measured by the odometry sensor give the measurement vector $z_{\text{rel},t}$. In order to compute the difference between two relative rotations $\boldsymbol{R}^I_{\text{rel}}$ measured by the IMU and $\boldsymbol{R}^S_{\text{rel}}$ measured by an odometry sensor, we have to compute an absolute rotation matrix. To preserve the relative character of the measurements, we multiply both relative rotations with the same absolute rotation matrix $\boldsymbol{R}_{t_{\text{start}}}$ to get pseudo-absolute rotation measurements. This absolute rotation matrix $\boldsymbol{R}_{t_{\text{start}}}$ should be the best estimate of the rotation from the IMU into the world frame at time step t_{start}:

$$\boldsymbol{R}^I_{t_{\text{end}}} = \boldsymbol{R}_{t_{\text{start}}}\boldsymbol{R}^I_{\text{rel}}, \qquad \boldsymbol{R}^S_{t_{\text{end}}} = \boldsymbol{R}_{t_{\text{start}}}\boldsymbol{R}^S_{\text{rel}}. \tag{2.32}$$

Now we can calculate the rotational difference matrix as

$$\boldsymbol{R}_{\text{diff}} = \boldsymbol{R}^I_{t_{\text{end}}} \cdot (\boldsymbol{R}^S_{t_{\text{end}}})^T. \tag{2.33}$$

The measurement vector $z_{\text{rel},t}$ contains the differences $\boldsymbol{p}_{\text{diff}}$ between the two relative translations and the angle differences $\boldsymbol{\varphi}_{\text{diff}}$ computed from $\boldsymbol{R}_{\text{diff}}$ using Eq. (2.28):

$$z_{\text{rel},t} = \left[\boldsymbol{p}_{\text{diff}}, \boldsymbol{\varphi}_{\text{diff}}\right]^T. \tag{2.34}$$

The augmented measurement matrix $\boldsymbol{H}^{\text{aug}}_{\text{rel},t}$ which projects the augmented state vector $\Delta\boldsymbol{x}^{\text{aug}}_t$ onto the measurement vector $z_{\text{rel},t}$ is

$$\boldsymbol{H}^{\text{aug}}_{\text{rel},t} = \begin{bmatrix}-\boldsymbol{H}_{\text{rel},t_{\text{Start}}} & \boldsymbol{H}_{\text{rel},t}\end{bmatrix}. \tag{2.35}$$

$$\boldsymbol{H}_{\text{rel},t} = \begin{bmatrix}\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times6}\\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times6}\end{bmatrix}. \tag{2.36}$$

We have to represent the relative character of the measurements in the measurement matrix $\boldsymbol{H}^{\text{aug}}_{\text{rel},t}$. We achieve that by the matrix $\boldsymbol{H}_{\text{rel},t_{\text{Start}}}$, which contains an identity matrix in the columns corresponding to the location of the cloned covariance of time t_{Start} and zeros everywhere else.

For computing the measurement noise matrix $\boldsymbol{Q}^m_{\text{rel},t}$, we have to know the standard deviations of the relative position and rotation measurements. The measurement errors of the leg odometry depend on how much the feet of the robot slip on the ground. On homogeneous ground we can assume a constant amount of slippage and find the noise values by filter tuning. For visual odometry, assuming constant standard deviations for the relative motion measurements is not appropriate. The visual odometry errors depend on the number and on the distribution of feature points in the image, and these parameters are not constant. Thus, we compute an error estimate for each visual odometry measurement as proposed by Stelzer et al. [86]. This estimate provides valuable information for fusing the visual odometry measurements with the data of other pose sensors. We transform the estimated errors of the visual odometry measurements into the IMU coordinate frame using error propagation and feed them into the measurement noise matrix.

Once we know $\boldsymbol{z}_{\text{rel},t}$, $\boldsymbol{H}^{\text{aug}}_{\text{rel},t}$ and $\boldsymbol{Q}^{m}_{\text{rel},t}$, we can compute the information contribution $\boldsymbol{i}^{\text{aug}}_{\text{rel},t}$ and the information amount $\boldsymbol{I}^{\text{aug}}_{\text{rel},t}$ using Eq. (2.6)-Eq. (2.7).

Update At every time step, we calculate $\boldsymbol{i}^{\text{aug}}_{k,t}$ and $\boldsymbol{I}^{\text{aug}}_{k,t}$ of each available sensor measurement. In the final step of the multisensor information filter, we sum these values and use them to update the predicted information vector and information matrix using Eq. (2.6)-Eq. (2.7). Finally, we transform the resulting information vector $\Delta\boldsymbol{y}^{\text{aug}}_{t}$ into an error state vector $\Delta\boldsymbol{x}^{\text{aug}}_{t}$. We extract $\Delta\boldsymbol{x}_t$ from $\Delta\boldsymbol{x}^{\text{aug}}_{t}$, which contains the estimated errors of the single robot states. If required, we can extract the covariance matrix $\boldsymbol{P}_t$ from $\boldsymbol{P}^{\text{aug}}_{t}$ after inverting the resulting information matrix $\boldsymbol{Y}^{\text{aug}}_{t}$.

Error State Feedback

We subtract the corresponding error estimates from the error state vector $\Delta\boldsymbol{x}_t$ to correct the position, velocity and bias values of the predicted state vector $\boldsymbol{x}^{-}_{t}$. For feeding back the estimated rotation angle error $\Delta\boldsymbol{\varphi}_t$, we compute a rotation matrix $\boldsymbol{R}_{\text{corr}}$ from $\Delta\boldsymbol{\varphi}_t$ using

$$\boldsymbol{R}_{\text{corr}} = \mathbf{I} + \lfloor \Delta\boldsymbol{\varphi}_t \times \rfloor \tag{2.37}$$

and perform the correction as

$$\boldsymbol{R}_t = \boldsymbol{R}^{T}_{\text{corr}} \cdot \boldsymbol{R}^{-}_{t}. \tag{2.38}$$

From $\boldsymbol{R}_t$ we can extract the corrected Euler angles via Eq. (2.28).

Filter Initialization

In the beginning of the data fusion process the robot is motionless in its starting position. We use this phase for filter initialization.

We estimate the starting orientation $\boldsymbol{R}_{t_0}$ with respect to the gravity vector from the very first IMU acceleration measurements $\boldsymbol{a}_{t_0}$ as shown in Eq. (2.24)-Eq. (2.25). Furthermore, we initialize the bias estimates $\boldsymbol{b}_{a,t_0}$ and $\boldsymbol{b}_{g,t_0}$ using the starting orientation, the known gravity vector $\boldsymbol{g}$ and the gyroscope measurements $\boldsymbol{\omega}_{t_0}$ as

$$\boldsymbol{b}_{a,t_0} = \boldsymbol{a}_{t_0} + \boldsymbol{R}_{t_0}^T \cdot \boldsymbol{g} \tag{2.39}$$

$$\boldsymbol{b}_{g,t_0} = \boldsymbol{\omega}_{t_0}. \tag{2.40}$$

From the following IMU measurements, we refine the estimates of the bias values and the starting orientation by exploiting the fact that the robot does not move. Hence, we feed position, velocity and orientation measurements with the value of zero and small noise matrices into the information filter. As a result the bias value estimation stabilizes. Furthermore, we fuse the absolute roll and pitch angle measurements from the accelerations with the orientation measurements from the gyroscopes as described above. The initialization phase is finished when the change in the bias estimates drops below a threshold. This process usually takes a few seconds. Once the information filter is initialized, the robot can start moving and visual odometry and leg odometry measurements are used to compute pose estimates.

2.1.5. Experimental Evaluation

For evaluating the performance of the multisensor data fusion filter we used the DLR Crawler (ref. Fig. 2.4), which is a six-legged, actively compliant walking robot [32, 33] that was developed as a prototype of an autonomous robot for rough terrain. Its legs are equipped with joint angle sensors that allow the computation of a leg odometry in 6 DOF [31]. Furthermore, it has a stereo camera head for perceiving its environment and an Xsens MTi-10 IMU. We used the IMU measurements at a rate of 120Hz, the leg odometry measurements at a rate of 10Hz and the visual odometry data with error estimates at a rate of about 4.5Hz. We steered the Crawler manually along a rectangular path through a 2m $\times$ 2m testbed filled with gravel. For this, we used a 6 DOF space mouse generating the commands "walk forward", "turn left/right" and "walk sideways to the left/right". The walking speed was approx-

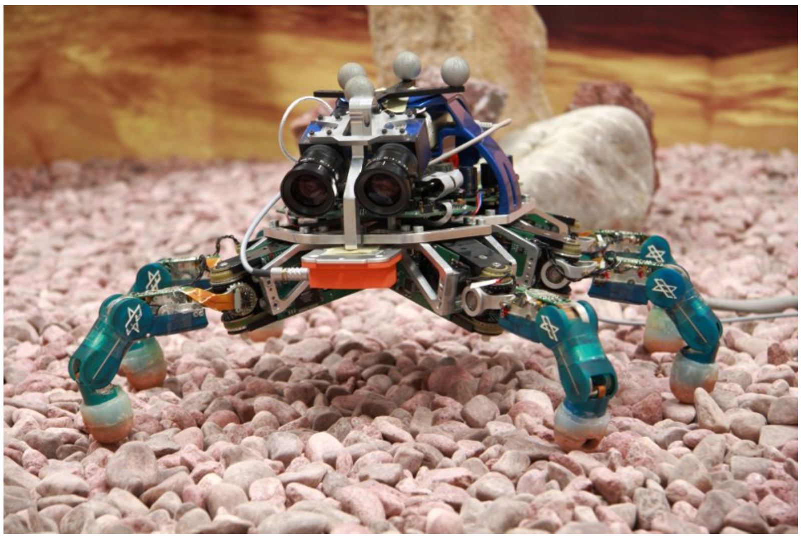
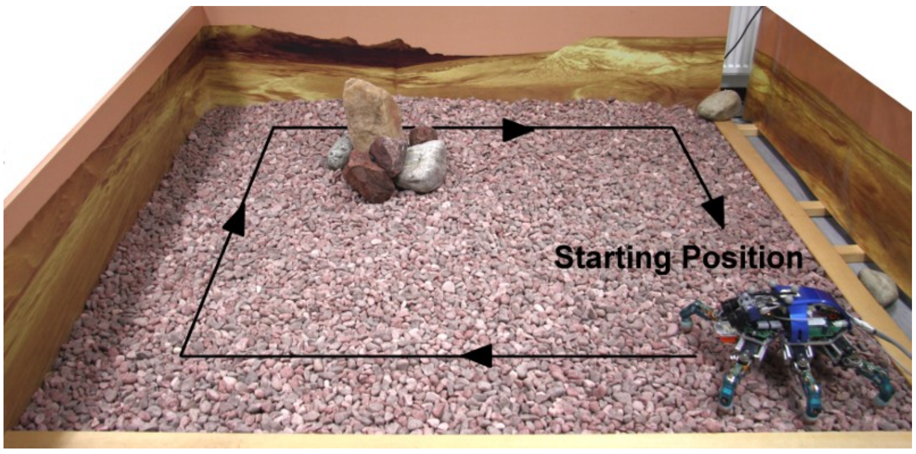


Figure 2.4.: The DLR Crawler as test platform for the multisensor data fusion method. It is equipped with a stereo camera, an IMU, joint angle sensors in the legs and a reflecting target body for ground truth measurements.

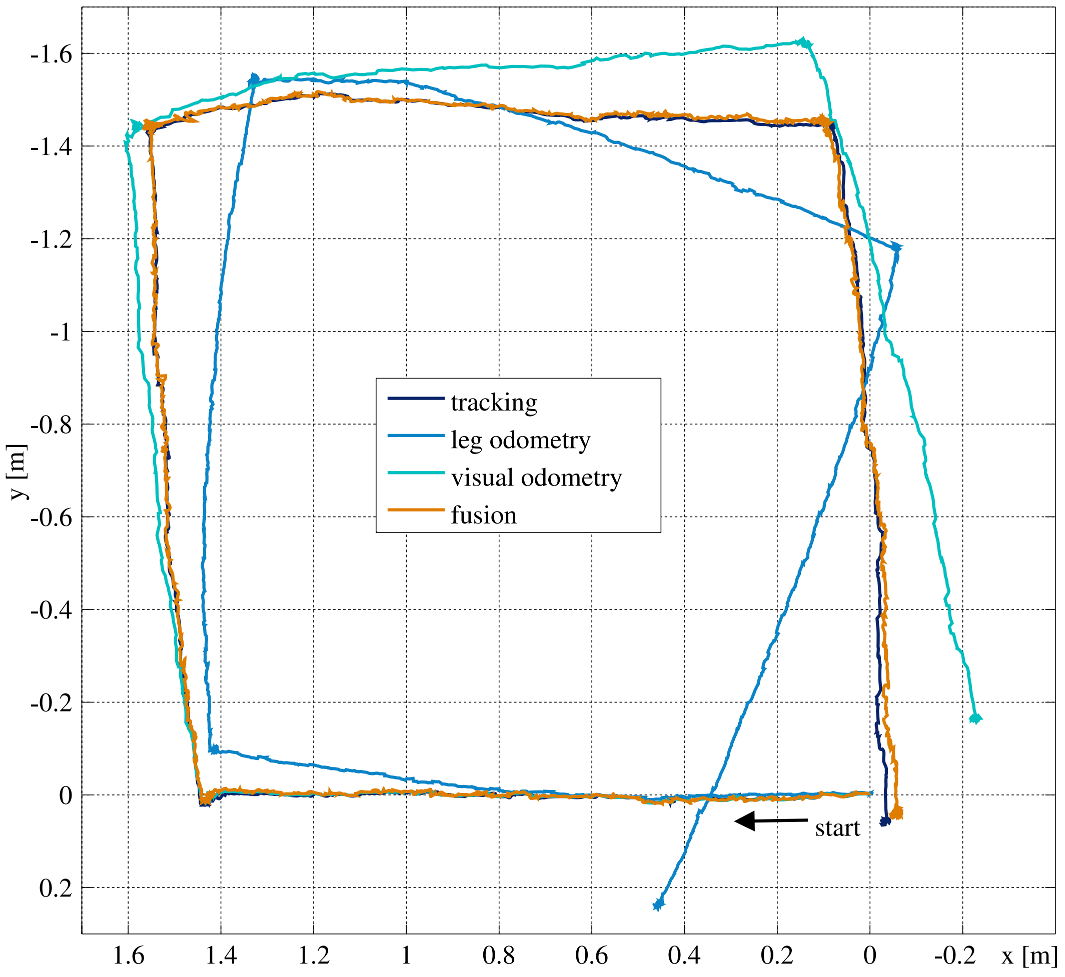
imately 0.04m/sec. We recorded the estimated trajectories measured by visual odometry and leg odometry, as well as the trajectory estimated by fusing inertial, visual and leg odometry data. Additionally, we mounted a reflecting target body on the Crawler and tracked it by an infrared tracking system. The trajectory of the target body provided a ground truth measurement.

We performed several runs with different lighting conditions. Fig. 2.5a shows the test setup with the robot in its starting pose and the approximately steered path. In this setup, the lighting conditions and the texture of the gravel were very good. Fig. 2.5b shows the ground truth trajectory measured by the tracking system, the fusion result and the different odometry trajectories which were obtained by summing the relative measurements of the respective sensors. The trajectory computed using only the IMU measurements is not shown here because its enormous drift led to an error of more than 100m after 60sec runtime.

The visual odometry trajectory was accurate apart from slightly underestimating the yaw angle. The leg odometry trajectory shows that yaw angles were overestimated because of slip in the gravel (ref. Fig. 2.6b). The fusion trajectory was close to the ground truth path. Fig. 2.6a shows plots of the z-coordinates. While visual odometry and leg odometry drifted due to roll and pitch angle errors, the estimated z-coordinate of the fusion result remained close to the ground truth curve because the absolute roll and pitch angle measurements from the accelerometers stabilized the pose estimate. Fig. 2.6c shows the standard deviations of

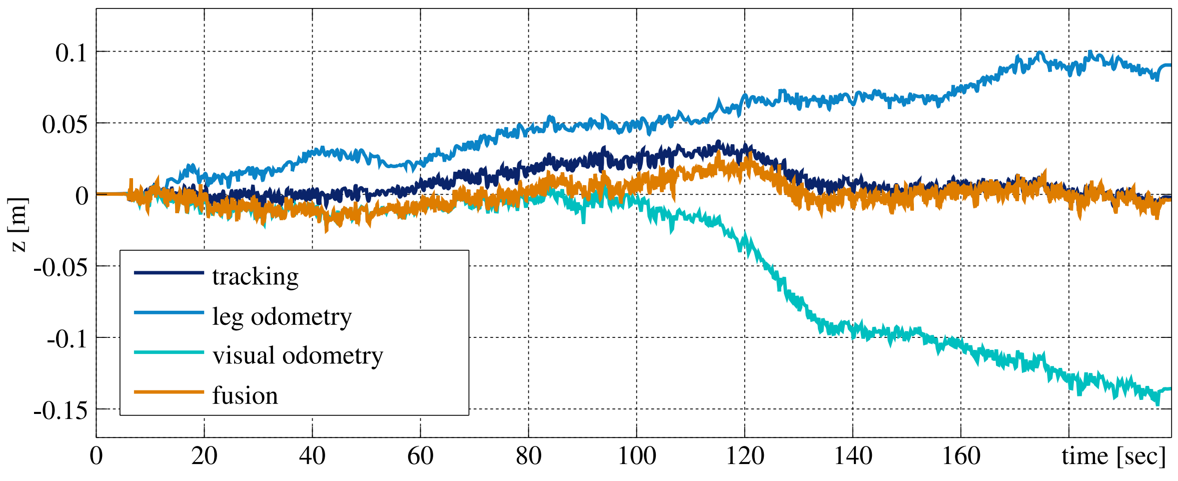


(a) Test setup and steered trajectory

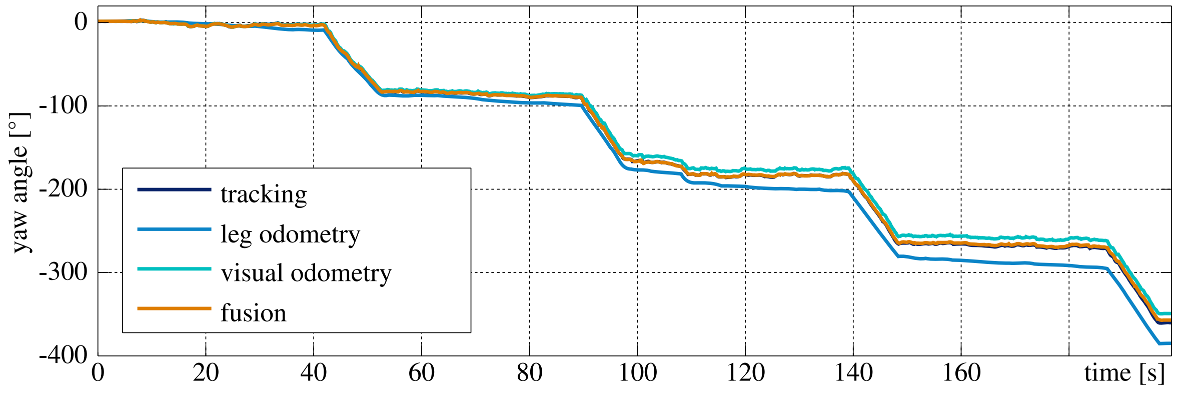


(b) Recorded trajectories

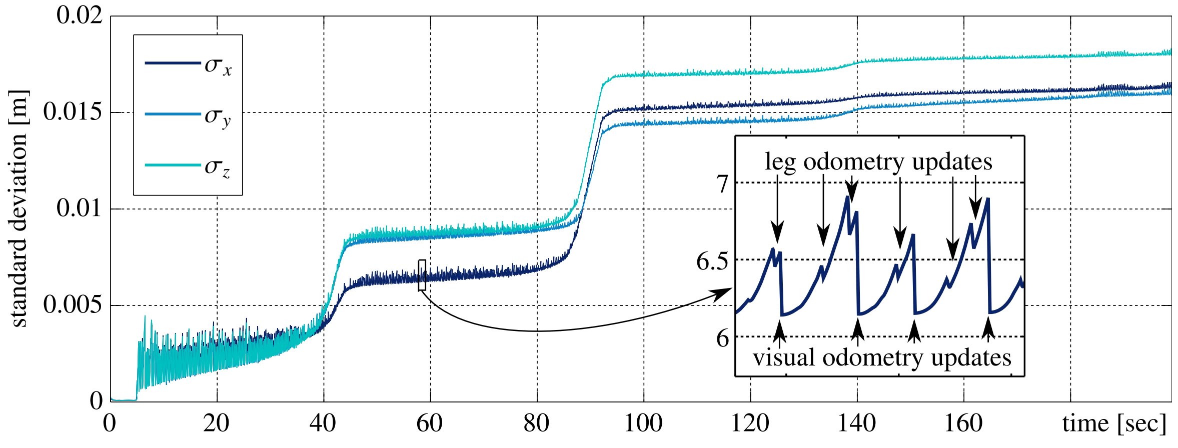
Figure 2.5.: Setup and recorded trajectories of a test run with good visual conditions



(a) z-coordinates



(b) Yaw angles



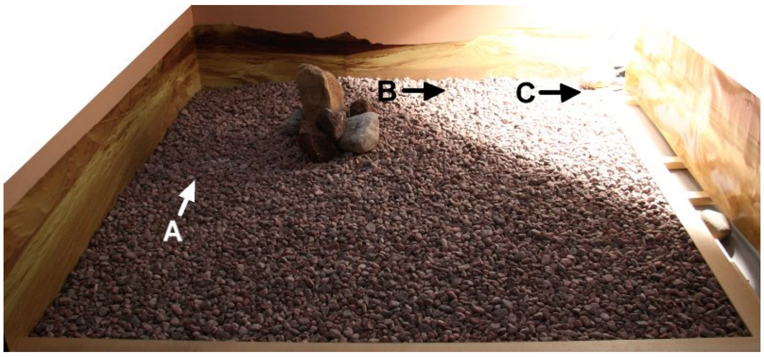
(c) Standard deviations computed from the estimation covariance matrix

Figure 2.6.: z-coordinates, yaw angles and position standard deviations of a test run with good visual conditions

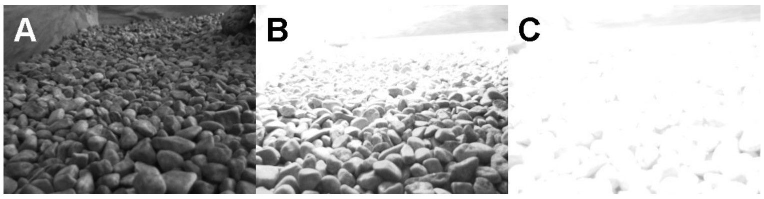
the position estimates computed from the estimation covariance matrix. As can be seen, the standard deviations grew with time since no absolute position measurements were available. The detailed plot in this figure illustrates the influence of the relative odometry measurements on the covariance: Without odometry measurements, the standard deviation of the position would grow quadratically because of integrating inertial measurements. Every time an odometry measurement was available, the uncertainty of the robot position decreased. In this application, visual odometry measurements usually had lower uncertainties than leg odometry measurements and, thus, had a stronger influence on the estimation covariance. However, during turning in the corners of the testbed, the estimated errors of the visual odometry measurements were higher, which led to increasing covariances during these periods. The reason for that was the texture of the testbed walls, which was worse than the texture of the gravel. However, since the robot additionally had the leg odometry and yaw angular velocity measurements, it could fuse all these information to an accurate yaw angle estimate.

In some of the test runs we simulated poor visual conditions by illuminating one corner of the testbed by a very bright light source (ref. Fig. 2.7a) and setting the camera to fixed exposure. As a result, the images taken from the illuminated area were overexposed and nearly white while all other images were well-exposed. Sample images taken by the Crawler's camera are shown in Fig. 2.7b. Fig. 2.7c and Fig. 2.8 show the recorded trajectories. Leg odometry suffered from a yaw angle error (ref. Fig. 2.8b) and overestimated its forward translation due to leg slip. Visual odometry was very accurate in areas with good lighting. However, large visual odometry errors occurred in the illuminated corner. These errors caused the visual odometry trajectory to continue in a wrong direction because of the summation of relative measurements. Since the estimated error of the visual odometry measurements was also higher in this region, the data fusion filter put a lower weight on these measurements. Thus, the leg odometry and inertial measurements had more influence on the pose estimate, so that the fusion result was not affected by the large visual odometry errors.

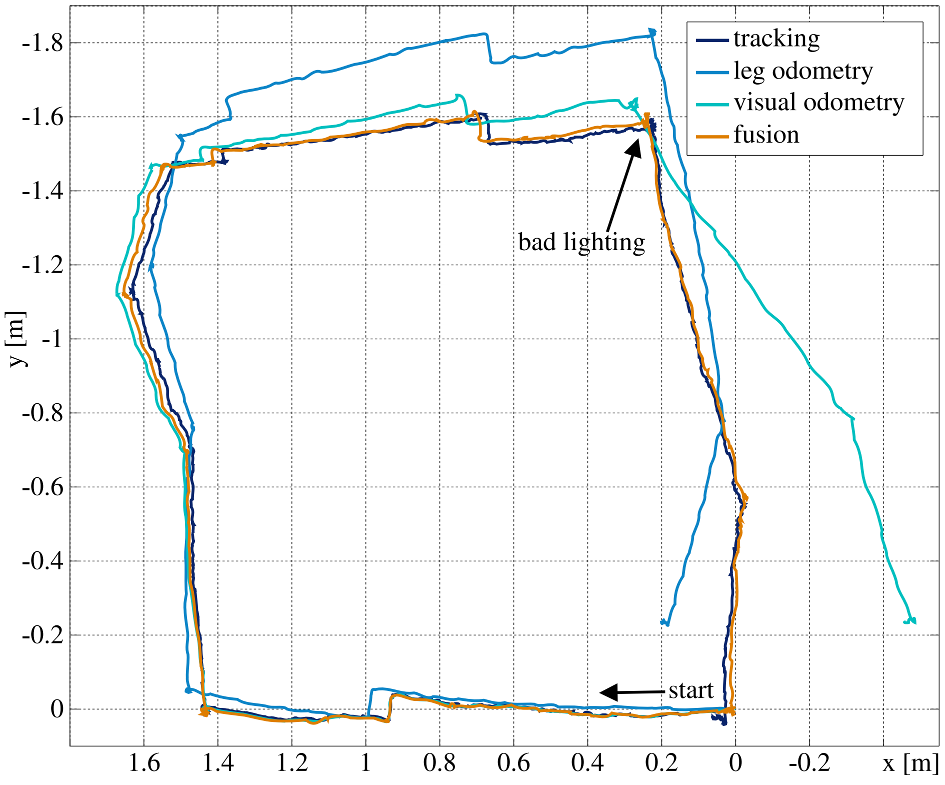
To achieve more general information about the performance of the data fusion filter, we computed the end point errors of the recorded trajectories as the distances to the ground truth trajectory end points. The results for 9 runs along a rectangular path in the gravel testbed with different lighting conditions are shown in Fig. 2.9. As can be seen, the fusion result was always significantly better than using only visual odometry or leg odometry. The average endpoint error of the fusion trajectory was 1.1% in relation to the average path length of 5.6m.



(a) Test setup

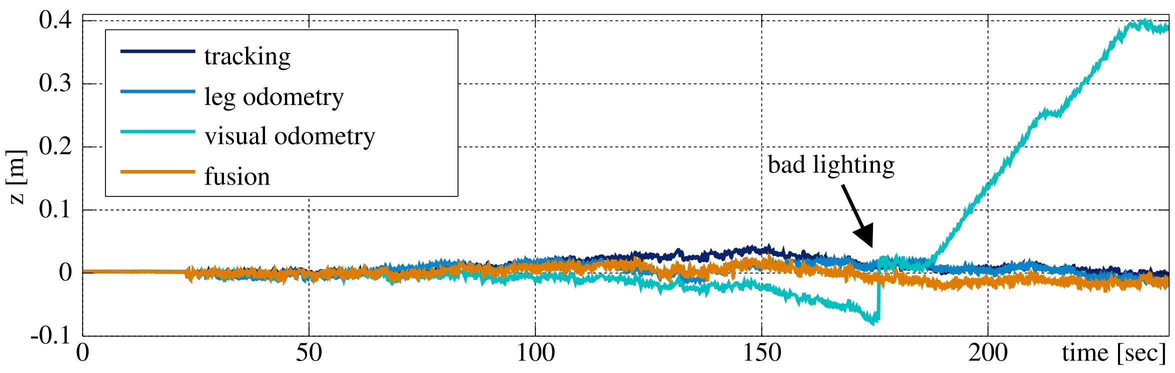


(b) Sample images

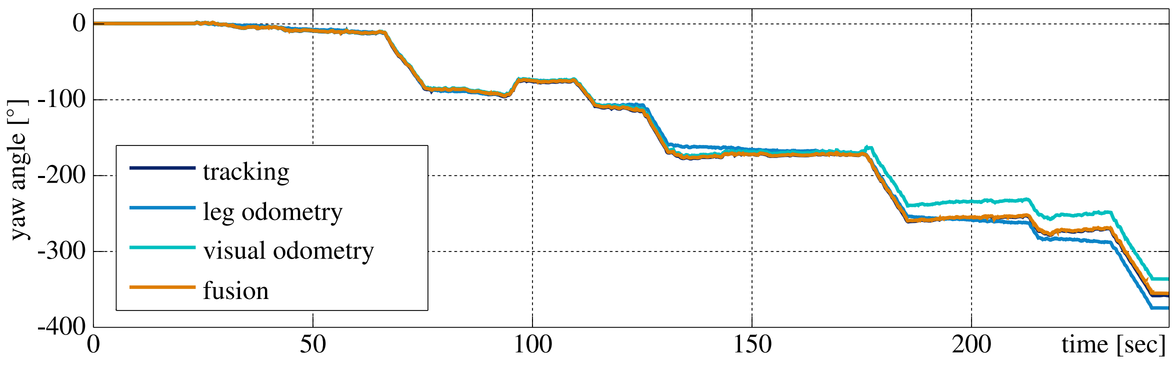


(c) Recorded trajectories

Figure 2.7.: Setup, sample images and recorded trajectories of a test run with poor lighting conditions



(a) z-coordinates



(b) Yaw angles

Figure 2.8.: z-coordinates and yaw angles of a test run with poor lighting conditions

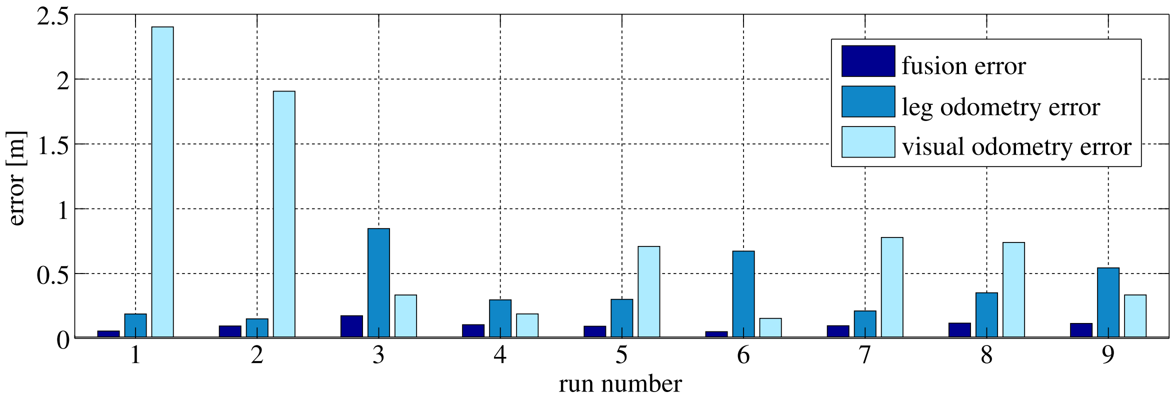


Figure 2.9.: End point errors of fusion, visual and leg odometry trajectories. Good visual conditions only in runs 4, 6 and 9.

These experimental results show that the data fusion algorithm improves robustness, as well as accuracy of the pose estimate compared to using only a single sensor. Based on the gravity vector, we can derive absolute roll and pitch angle measurements from the IMU data, which limit the drift in the orientation estimate. The IMU further allows for short-term stable position and yaw angle estimates, which are supported and corrected by the aiding odometry measurements. The visual odometry error estimate reliably detects ill-conditioned measurements as caused by bad visual conditions and the data fusion filter puts a lower weight on those measurements. Using the leg odometry measurements, the robot is able to overcome visually poor areas with a high accuracy.

However, it can also be seen that the position estimate still suffers from a drift caused by summing up translational and yaw angle errors in the relative motion estimates. This error can only be eliminated by using absolute measurements such as compass or GPS data (where available) or landmark positions. Nevertheless, since we only require locally accurate pose estimates, the presented data fusion method is sufficient for our application.

2.2. Local Mapping

The robot requires a local map for obstacle detection and avoidance and for local path planning. For this, the local map must represent the geometric properties of the terrain, which the robot can evaluate to find regions that are safe for traversal.

2.2.1. State of the Art

We can find different types of metric maps representing 3D terrain geometry in literature.

Raw point clouds are, for example, used by Cole and Newman [16], Henry et al. [38] and Howard et al. [40]. Point clouds make the registration of scans from different view points easy, but they suffer from a high memory consumption.

A *Digital Elevation Map (DEM)* is a 2D grid with fixed resolution that stores a single height value in each cell, which results in a 2.5D representation of the terrain surface. They need only little storage space, but they cannot represent overhangs or multiple height values per grid cell. The resolution of the DEM has to be chosen appropriately. If the resolution is too low, highly complex terrain cannot be modeled. If the resolution is too high, much storage space is required and holes can occur in the map if the sensor for perceiving the environment does not provide such high-resolution data. Digital elevation maps are widely used in robotic applications where no full 3D model is required, for example in the works by Kolter et al.

[47], Kweon and Kanade [53] and Ye and Borenstein [102]. Triebel et al. [92] proposed an extension of an elevation map to multiple levels.

A *3D Voxel Occupancy Grid* is a full 3D grid with fixed resolution where each grid cell stores a probability value of that cell being occupied. 3D voxel maps require more storage space than DEMs, but they can represent the full 3D structure of the terrain. They are used by Moravec [69] and Suzuki et al. [89]. Their memory consumption can be decreased by using tree-like structures as shown by Payeur et al. [73] and Wurm et al. [100].

A *Triangle Mesh* is composed of connected triangles which approximate the surface of the terrain. The resolution of the mesh can vary and it is possible to represent the full 3D structure of the environment. Triangle meshes are used by Huber and Hebert [41], Früh et al. [26], Hähnel et al. [36] and Rusu et al. [78]. Similar to the raw point clouds, they also require much memory.

Continuous terrain functions can be created using statistical learning processes. Plagemann et al. [74] treat terrain mapping as a regression problem and recover a continuous elevation function from noisy range measurements using a Gaussian Process model with different lengthscales (smaller for densely sampled parts, higher for sparsely sampled regions). This method provides an explicit model of uncertainty. Hadsell et al. [35] learn continuous terrain functions using a kernel based approach where the lengthscale is chosen according to the sensor uncertainty at a specific point.

2.2.2. Approach

In this thesis, we chose to use a Digital Elevation Map (DEM) as local map, since it requires only little memory and is sufficient for the task of local metric mapping. In case the robot moves through a multi-level environment, the small size of the local map ensures that not more than one level is modelled in the same map. We perform mapping in two steps. First, we create a single-view DEM from a single depth image of the stereo camera, which in our application is the only sensor that gives dense information about the shape of the terrain. For small robots with a low point of view (ref. Fig. 2.10) a single camera image pair usually only covers a small region which is not wide enough to allow for meaningful obstacle detection. Thus, we have to combine several single-view DEMs to a local DEM using the accurate pose estimates obtained by the multisensor data fusion process described in the previous section. We choose the resolution of the DEM according to the robot properties. For a walking robot,

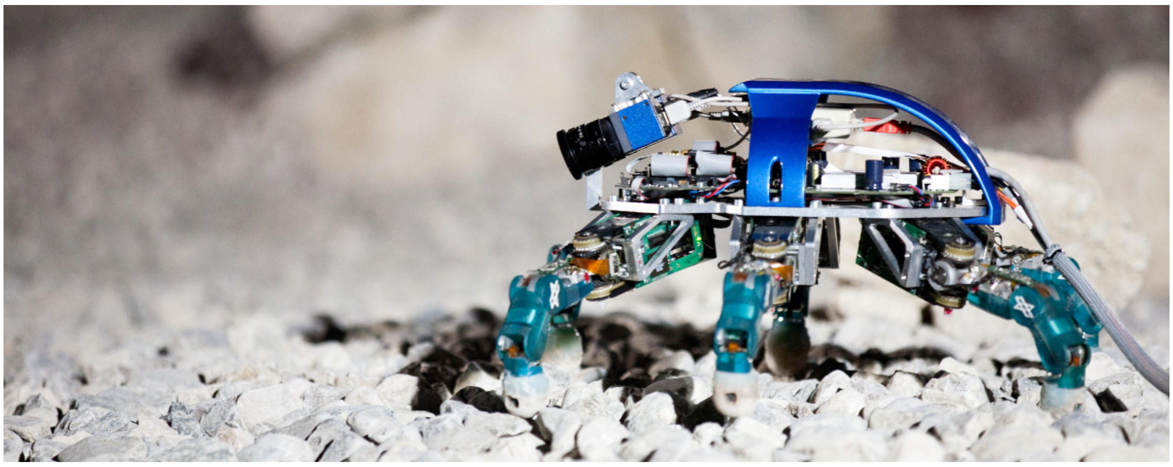


Figure 2.10.: The DLR Crawler as example of a robot with a low point of view

we chose the resolution to be equal to the size of a foot. For a driving robot, the resolution was chosen to be equal to the width of a wheel.

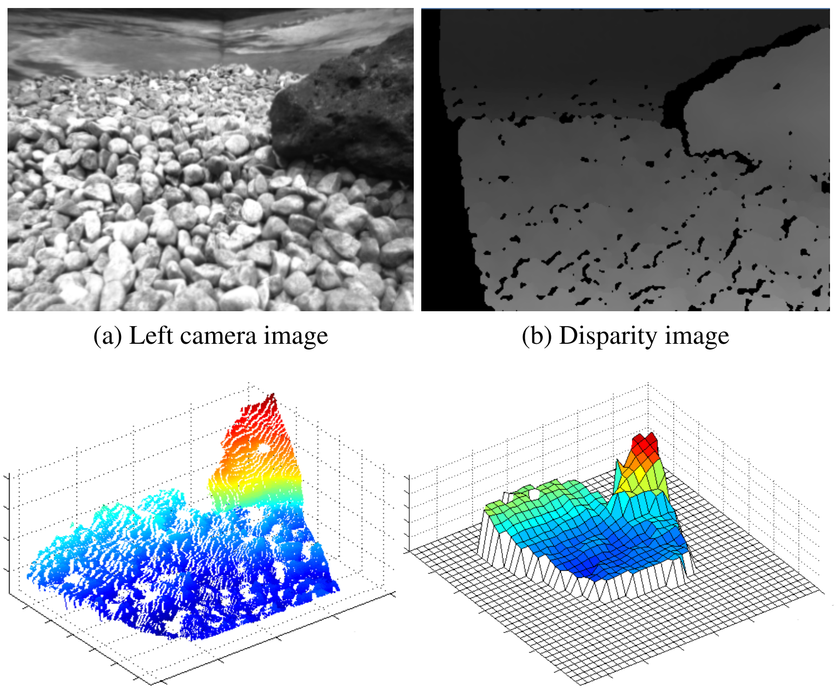
2.2.3. Creating Single-View Maps from Disparity Images

This section will present two approaches for creating a DEM from a single stereo image pair and discuss their performances and computational complexities. Furthermore, we will discuss the influence of stereo reconstruction errors on the map range and resolution.

Point Cloud Solution

A straightforward method for creating a DEM from a disparity image is to compute a point cloud from the disparity image and then to fill these points into a height grid. This approach is visualized in Fig. 2.11 and was presented by Chilian and Hirschmüller [14], where it gave good results for the chosen camera and map resolution.

For creating the point cloud, we have to reconstruct the 3D coordinates of the image points. The stereo algorithm finds corresponding image points $p_l = (p_{lx}, p_{ly})$ in the left and $p_r = (p_{rx}, p_{ry})$ in the right rectified image. Due to the epipolar geometry, $p_{ly} = p_{ry}$ and $p_{lx} = p_{rx} + p_d$, where p_d is the disparity value. The values p_{lx}, p_{ly} and p_d are stored in the disparity image (ref. Fig. 2.11b).



(a) Left camera image (b) Disparity image

(c) Point cloud (d) DEM

Figure 2.11.: Creating a single-view DEM from a stereo image using a point cloud

We can reconstruct the object point $\boldsymbol{P}^c = (x^c, y^c, z^c)$ in the camera coordinate frame c from the image points $p^i_{lx}, p^i_{ly}, p^i_d$ in the image coordinate frame i using the standard stereo equations

$$x^c = \frac{t \cdot p^i_{lx}}{p^i_d} = \frac{t \cdot p^i_{lx}}{p^i_{lx} - p^i_{rx}}, \tag{2.41}$$

$$y^c = \frac{t \cdot p^i_{ly}}{p^i_d} = \frac{t \cdot (p^i_{ly} + p^i_{ry})}{2 \cdot (p^i_{lx} - p^i_{rx})}, \tag{2.42}$$

$$z^c = \frac{t \cdot f}{p^i_d} = \frac{t \cdot f}{p^i_{lx} - p^i_{rx}}, \tag{2.43}$$

where t is the stereo camera base line and f is the focal length. Then, we can transform the point $\boldsymbol{P}^c$ into the world coordinate frame using the estimated camera pose at the time of image acquisition. Fig. 2.11c shows the resulting point cloud. Now, we can store the mean z-value of all points whose (x, y)-coordinates fall into the same grid as the height value of that cell (ref. Fig. 2.11d).

The complexity of this method is linear in the number of pixels in the image and independent of the map resolution. The size of the local map has little influence on the computation time, because it only affects the number of cell means but not the number of 3D coordinates to be computed.

Resolution Issues

The main disadvantage of the point cloud approach is that the maps become sparse at a certain distance from the camera. The robot takes the stereo images from a low viewpoint and at a shallow angle, as shown in Fig. 2.10. Hence, the constant resolution of the disparity image leads to non-uniform sampling of the terrain, which is illustrated in Fig. 2.12. Areas that are viewed at a very shallow angle or that are far away from the camera will have a low resolution. Additionally, higher objects, which occlude parts of the terrain behind, cause range shadows that have to be considered carefully. Thus, we should use a different mapping approach that pays stronger attention to the resolution of the resulting map.

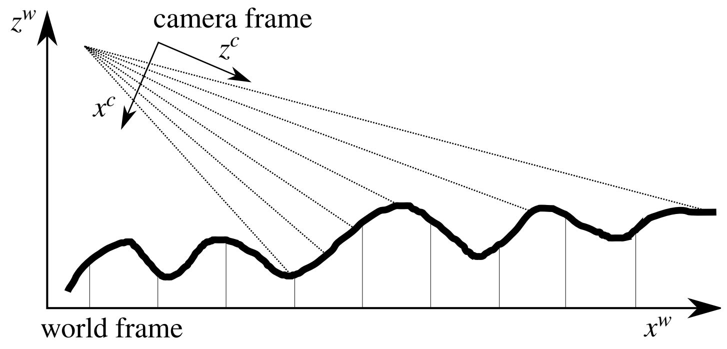


Figure 2.12.: Non-uniform terrain sampling for a camera with a low point of view (adapted from Kweon and Kanade [53], ©1990 IEEE)

Different methods have been proposed which address the problems of range shadows and holes in the maps. In the approach of Kolter et al. [47], gaps occurring due to range shadows and occlusions are filled using a texture synthesis method. For this, the method considers geometric constraints such as that the height of the missing terrain points must lie below the line of sight of the camera connecting the last and the next visible region of the terrain. This method requires a priori knowledge of height maps of similar terrain types. Other approaches use interpolation in the world coordinate frame, for example Früh et al. [26] apply horizontal and vertical interpolation to fill range shadows for urban modelling. Another way to create dense height maps is to triangulate adjacent pixels in the disparity image [41] if their distance in the camera frame is below a certain threshold. The resulting mesh can be transformed into the world coordinate system. This method has the advantage that the surface topology is preserved. The *locus method* proposed by Kweon and Kanade [53] is an effective way to create dense height maps from range images and will be described in the following section.

The Locus Method

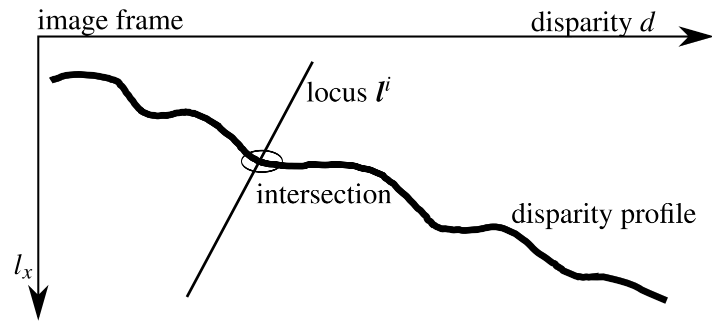


Figure 2.13.: Intersection of locus and disparity profile (adapted from Kweon and Kanade [53], ©1990 IEEE)

In the original publication [53] the locus method was derived for range data like laser scans, which give direct range measurements under certain horizontal and vertical scanning angles. In the next section we will derive how to apply this method to disparity images.

The locus method finds the elevation z^w at a point (x^w, y^w) of a reference plane by computing the intersection of the terrain with a vertical line at (x^w, y^w). The projection of the vertical line on the image is called *locus*. The method computes the intersection point with the terrain in image space rather than in Cartesian space (ref. Fig. 2.13). In image space, the terrain is represented by a depth profile.

For each cell in the DEM we compute a vector representing a vertical line $\boldsymbol{l}$ as

$$\boldsymbol{l} = (\boldsymbol{u}, \boldsymbol{v}) = \left([x^w, y^w, z^w]^T, \left[v_x, v_y, v_z\right]^T\right), \tag{2.44}$$

where $\boldsymbol{u}$ is a point and $\boldsymbol{v}$ is a unit directional vector. Each point $\boldsymbol{r}$ on the line is then represented by

$$\boldsymbol{r} = \boldsymbol{u} + \lambda \boldsymbol{v}. \tag{2.45}$$

For a vertical line in world coordinates the values of $\boldsymbol{u}$ and $\boldsymbol{v}$ become

$$\boldsymbol{u} = [x^w, y^w, 0]^T \tag{2.46}$$

$$\boldsymbol{v} = [0, 0, 1]^T \,. \tag{2.47}$$

We can transform this line into camera coordinates using the rotation matrix $\boldsymbol{R}_c^w$, which turns a point from the camera coordinate frame c to the world coordinate frame w, and the translation $\boldsymbol{t}^w$ from the world coordinate frame to the camera coordinate frame:

$$\boldsymbol{l}^c = (\boldsymbol{u}, \boldsymbol{v}) = \left((\boldsymbol{R}_c^w)^T(\boldsymbol{u} - \boldsymbol{t}^w), (\boldsymbol{R}_c^w)^T\boldsymbol{v}\right). \tag{2.48}$$

By projecting $\boldsymbol{l}^c$ onto the disparity image, we obtain a generalized locus $\boldsymbol{l}^i$. The projections $\boldsymbol{r}^i = (r^i_{lx}, r^i_{ly}, r^i_d)^T$ of all points $\boldsymbol{r}^c$ of line $\boldsymbol{l}^c$ onto the image also form a line $\boldsymbol{l}^i$ with

$$r^i_d = \frac{f \cdot t}{r^c_z} \tag{2.49}$$

$$r^i_{lx} = \frac{r^i_d \cdot r^c_x}{t} = \frac{f \cdot r^c_x}{r^c_z} \tag{2.50}$$

$$r^i_{ly} = \frac{r^i_d \cdot r^c_y}{t} = \frac{f \cdot r^c_y}{r^c_z}. \tag{2.51}$$

Now, we have to compute the intersection of $\boldsymbol{l}^i$ with the disparity profile of the image. Due to the orientation of the camera with respect to the world coordinate frame, it is beneficial to parameterize the locus line by the image row coordinate r^i_{ly}, because a vertical line in world coordinates will most likely run through all rows in the image, but will only appear in few columns. Hence, we can create line equations for the column $r^i_{lx} = f(r^i_{ly}) = a_x r^i_{ly} + n_x$ and disparity $r^i_d = f(r^i_{ly}) = a_d r^i_{ly} + n_d$ by transforming two distant points $\boldsymbol{r}^c_1$ and $\boldsymbol{r}^c_2$ on $\boldsymbol{l}^c$ into image coordinates $(r^i_{lx1}, r^i_{ly1}, r^i_{d1})$ and $(r^i_{lx2}, r^i_{ly2}, r^i_{d2})$ and determining the line equation parameters slope a and offset n for disparity (index d) and column (index x).

$$a_x = \frac{r^i_{lx1} - r^i_{lx2}}{r^i_{ly1} - r^i_{ly2}} \qquad n_x = r^i_{lx1} - a_x \cdot r^i_{ly1} \tag{2.52}$$

$$a_d = \frac{r^i_{d1} - r^i_{d2}}{r^i_{ly1} - r^i_{ly2}} \qquad n_d = r^i_{d1} - a_d \cdot r^i_{ly1} \tag{2.53}$$

The first step for finding the intersection is to search two sample points $(r^i_{lx1}, r^i_{ly1}, r^i_{d1})$ and $(r^i_{lx2}, r^i_{ly2}, r^i_{d2})$ on the line which fulfil the condition

$$r^i_{d1} < p^i_d(r^i_{lx1}, r^i_{ly1}) \tag{2.54}$$

$$r^i_{d2} > p^i_d(r^i_{lx2}, r^i_{ly2}). \tag{2.55}$$

For doing so, we search for the first sample point by starting at the bottom row of the image, searching for the second sample point starts at the top row. For each row, we can compute the disparity and column values using the line equations above.

After finding those two sample points, we apply binary search between these points to find the intersection $r^i_{ilx}, r^i_{ily}, r^i_{id}$ with $r^i_{id} = p^i_d(r^i_{ilx}, r^i_{ily})$. We should note that there are image pixels which do not have a valid disparity value. If the search reaches such a pixel, the search terminates and a linear search is performed between the current interval boundaries. If the search interval cannot be reduced any further and the disparity difference between the boundary values $p^i_d(r^i_{lx1}, r^i_{ly1})$ and $p^i_d(r^i_{lx2}, r^i_{ly2})$ is within a certain threshold, the intersection is found and calculated as

$$r^i_{ilx} = \frac{r^i_{lx1} + r^i_{lx2}}{2} \tag{2.56}$$

$$r^i_{ily} = \frac{r^i_{ly1} + r^i_{ly2}}{2} \tag{2.57}$$

$$r^i_{id} = \frac{r^i_{d1} + r^i_{d2}}{2}. \tag{2.58}$$

The threshold is necessary to avoid filling holes that are caused by range shadows. If the linear search finds multiple intersection points, we only consider the intersection with the highest elevation. We can calculate the elevation value from the intersection point using the reconstruction equations in Eq. (2.41)-Eq. (2.43).

The complexity of the locus method is linear in the number of cells in the DEM and in the best case logarithmic in the number of rows of the disparity image (worst case: linear).

Comparison of the Point Cloud Approach and the Locus Method

A comparison of DEMs created by the point cloud approach and by the locus method is shown in Fig. 2.14. The maps have a resolution of 2cm and a size of 3m×3m. They were computed from a stereo image pair of the size 1292 × 964 pixels recorded with wide angle lenses.

The map resulting from the point cloud approach (ref. Fig. 2.14c) is dense in regions close to the camera and becomes sparse in the distance, due to the non-uniform sampling of the terrain. Computation time for creating this map was 0.5sec on a standard CPU with 2.6GHz.

The map computed from the locus method (ref. Fig. 2.14d) is dense everywhere except for regions where range shadows were caused by larger objects in the foreground. Computation time for this map was 0.7sec. However, since the computation time of the locus method is linear in the size of the map, it will decrease for coarser and for smaller maps. For a slightly lower resolution of 3cm, the computation time of the locus method is already lower than for the point cloud approach (ref. Fig. 2.15). For smaller maps, the time will decrease further.

Map Range and Stereo Errors

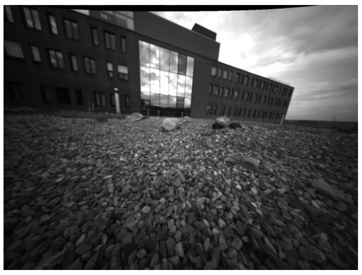
In stereo vision the error of reconstructing the distance of an object point grows quadratically with the distance of that point from the camera. To prevent erroneous range measurements from being inserted into the terrain map, the size of the local map should be limited to the range where the assumed disparity error ϵ_d results in distance reconstruction errors that are still lower than the map resolution r. We can compute this range z^c_{crit} from the focal length f and the camera baseline t as

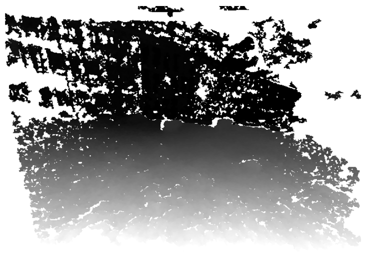
$$z^c_{crit} = \frac{t \cdot f}{p^i_{d_1}} \tag{2.59}$$

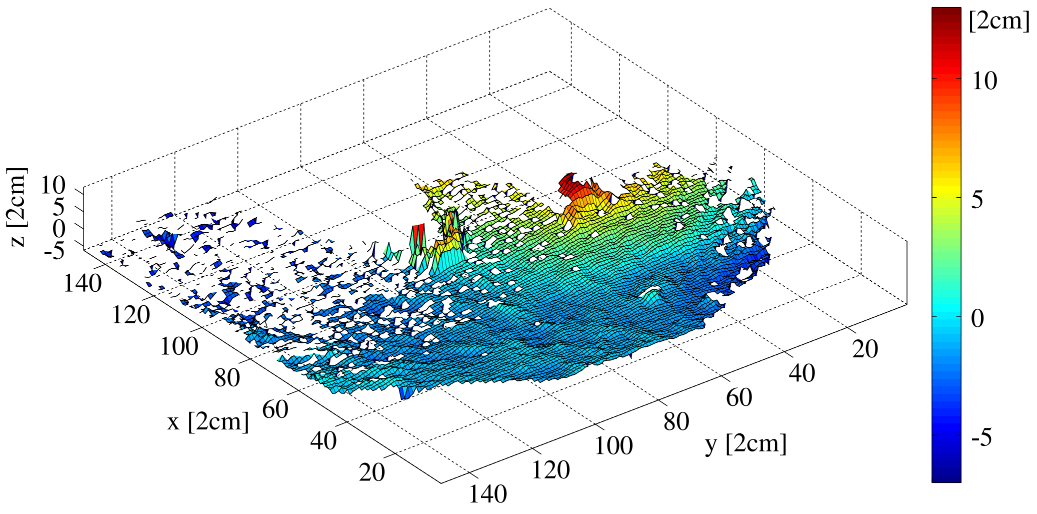
$$z^c_2 = z^c_{crit} - r = \frac{t \cdot f}{p^i_{d_2}} = \frac{t \cdot f}{\epsilon_d + p^i_{d_1}} \tag{2.60}$$

$$\rightarrow z^c_{crit} = \frac{r}{2} + \sqrt{\frac{r^2}{4} + \frac{r \cdot t \cdot f}{\epsilon_d}}. \tag{2.61}$$

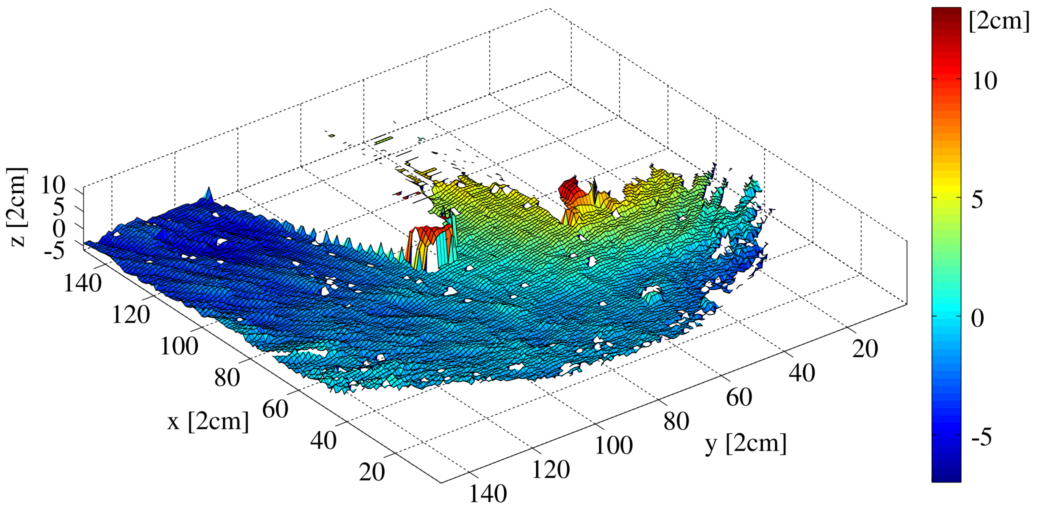
Usually, a disparity error of $\epsilon_d = 1$px is assumed. For the cameras with wide angle lenses ($f = 635$px) that were used for creating the DEMs in the previous section, Fig. 2.16 visualizes the relation between the critical range z^c_{crit} and the map resolution for different baselines. The plot shows that for the used baseline of $t = 9$cm, a resolution of $r = 4$cm is necessary to reconstruct reliably within a range of $z^c_{crit} = 1.5$m. Longer baselines, as well as longer focal lengths, will increase this value.


(a) Left camera image


(b) Disparity image



(c) DEM from point cloud approach



(d) DEM from locus method

Figure 2.14.: Comparison of the point cloud approach and the locus method for creating a local map with 2cm resolution and a size of 3m×3m

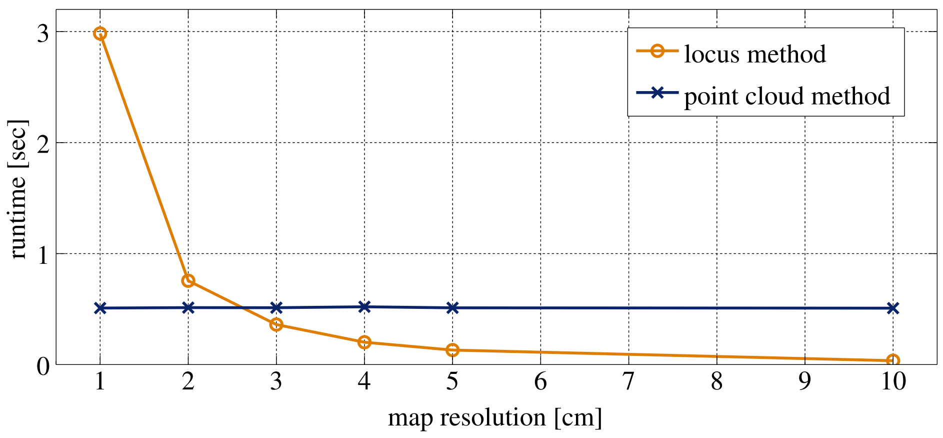


Figure 2.15.: Runtime comparison for a 3m×3m map at different resolutions

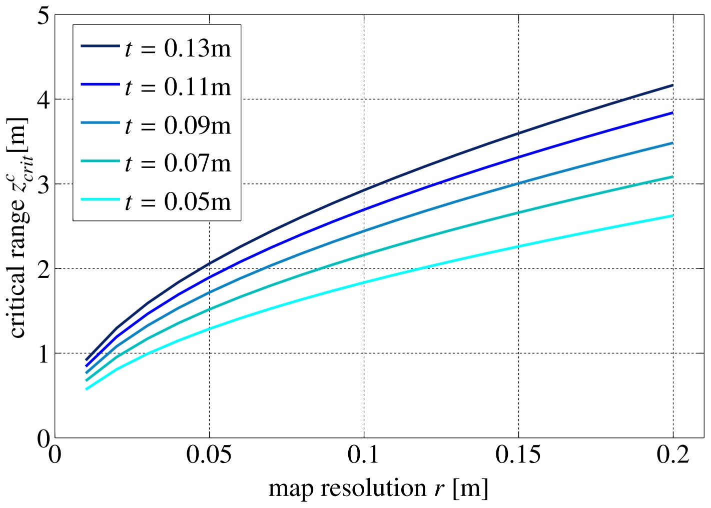


Figure 2.16.: Relation between the map resolution and the critical range for different baselines t (focal length $f = 635$px)

2.2.4. Combining Single-View Maps

Small robots with a low point of view can usually only cover a narrow region of the terrain by a single stereo image pair, which is not sufficient for obstacle detection. Thus, we have to combine terrain measurements from different points of view to a map. We assume that the camera poses at the time of image acquisition are known with a very high accuracy from the pose estimation step. Thus, we can use this knowledge to attach each single-view DEM from a depth image to the local DEM. Existing height values are overwritten by newer values. This approach is subject to errors from pose estimation, which can cause artifacts in the DEM. However, these errors remain small for small scale maps and can be considered in the traversability estimation process by taking the time into account when the height value was inserted into the local map [14]. In other cases, registration methods such as Iterative Closest Point (ICP) [105] have to be applied.

2.3. Traversability Estimation

The detection of obstacles or untraversable areas is crucial for safe navigation in unknown terrain. As already mentioned, in rough terrain there is no clear distinction between untraversable obstacles and free space. Rather, the robot has to estimate the traversability of the terrain on a continuous scale between "most easily traversable" and "untraversable" and write this value to a cost map, which has the same size and resolution as the local geometric DEM, but stores the traversability costs instead of the height values. In some works a traversability cost map is derived directly from range measurements of the environment, e.g. from stereo vision [82] or from Kinect data [6]. Other approaches estimate the terrain traversability based on a geometrical model of the environment, e.g. in form of a 2.5D elevation map [101]. As explained in the previous section, in our approach the robot geometrically models its surroundings for traversability estimation, because a single view does not always capture a region wide enough for reliable obstacle detection.

In this thesis, we use the traversability estimation method described by Chilian [13], which is based on plane fitting as in the GESTALT system [30]. We will briefly summarize this method here for completeness.

Each cell of the cost map is assigned a danger value d ($d \in \{[0, 1], \infty\}$) describing the terrain difficulty. A cell is traversable if the robot is not exposed to critical terrain hazards irrespective of its orientation given its center is located in that cell. Thus, we can treat the robot as a point and do not have to consider its orientation in the path planning process, which simplifies computation. A danger value of $d = 0$ stands for completely flat, smooth terrain, which can be traversed by the robot most easily. Higher danger values are assigned to areas which are harder to pass. A value of $d = 1$ describes terrain which is just barely traversable for the robot. Untraversable regions are assigned $d = \infty$. We assume that unknown areas are traversable but assign a high danger value of $d = 1$.

Based on the DEM, we estimate three potential hazards, which are steep slopes, high terrain roughness and high steps. If one of the criteria exceeds the corresponding critical value, the cell is marked as untraversable. The critical values are the maximum slope s_{crit}, maximum roughness r_{crit} and maximum step height h_{crit} which the robot can traverse without tipping over or getting stuck.

Similar to the traversability estimation in GESTALT [30], we calculate the slope s of a cell by fitting a plane in a circular region around the cell with a diameter corresponding to the maximum diameter of the robot. The angle between the plane normal and the z-axis of the global coordinate frame gives the slope inclination s. The terrain roughness r is calculated as the standard deviation of the terrain height values from the computed plane in the circular region around the cell.

We compute the step height h in two steps. First, we calculate local height differences within a square window of several grid cells (corresponding to approximately one third of the robot diameter) for all cells in the circular region. If the maximum height difference between any cell in that window and the center cell of the window is greater than the critical step height h_{crit}, and the slope between the corresponding two terrain points is higher than the critical slope s_{crit}, we store the maximum height difference as the temporary step height of the central cell of the window. Second, we compute the step height of the central cell of the circular region as

$$h = \min(h_{\max}, h_{\max} \cdot \frac{n_{st}}{n_{\text{crit}}}), \tag{2.62}$$

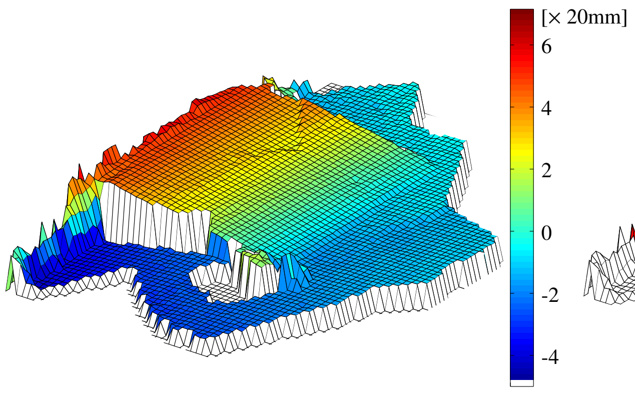
where $h_{\max}$ is the maximum temporary step height in the circular region, n_{st} is the number of cells in the circular region whose temporary step heights are higher than the critical step height and n_{crit} is the valid number of cells with a temporary step height higher than the critical step height. This method for calculating the step height also detects small steep slopes as steps and is robust against missing terrain information, which distinguishes it from the step height calculation method in GESTALT [30].

For traversable cells we compute the danger value from the three types of hazards as

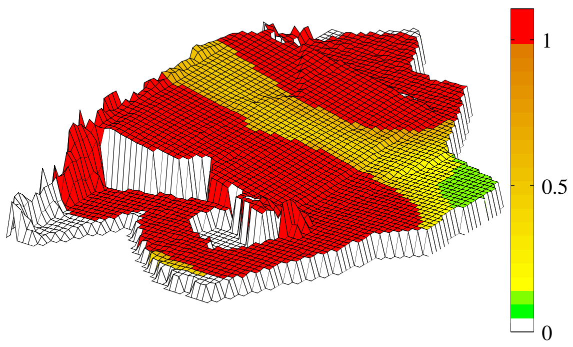
$$d = \alpha_1 \frac{s}{s_{\text{crit}}} + \alpha_2 \frac{r}{r_{\text{crit}}} + \alpha_3 \frac{h}{h_{\text{crit}}}, \tag{2.63}$$

where α_1, α_2 and α_3 are weight parameters which sum up to 1.

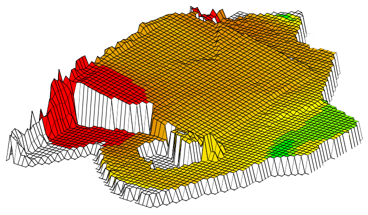
As Fig. 2.17 shows, the step height is well suited for detecting whether a cell is traversable or not, but it provides little information about the difficulty of the traversable cells. In contrast, the slope and roughness criteria can fail to detect untraversable cells but are better suited for estimating the difficulty of traversable cells.



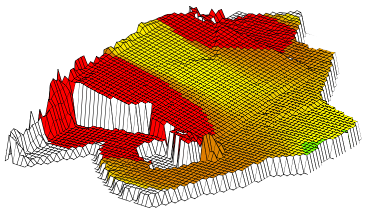
(a) DEM



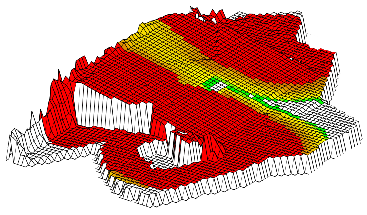
(b) Cost map



(c) Slope



(d) Roughness



(e) Step height

Figure 2.17.: Danger value computation from the criteria slope, roughness and step height ($s_{\text{crit}} = 20°$, $r_{\text{crit}} = 30\text{mm}$, $h_{\text{crit}} = 50\text{mm}$, $\alpha_1 = 0.5$, $\alpha_2 = 0.25$, $\alpha_3 = 0.25$)

The traversability of a cell is only computed if a sufficient number of height values is present in the circular robot-sized region around the cell. In addition to the danger value, we calculate a certainty value for each cell as the percentage of available height values in the circular region. The robot uses this value later to decide whether the active exploration of an area is necessary. The certainty and danger values are stored in the cost map. For more detailed information about the traversability estimation method please refer to Chilian [13] or Chilian and Hirschmüller [14].

2.4. Path Planning

For moving along a safe route in a given direction or towards an intermediate goal point, the robot has to plan a path based on the traversability cost map. The GESTALT planner [30] uses arc votes to command steering angles to the rover. From a bunch of driving arcs leading through safe terrain, GESTALT chooses the one that contributes most to moving towards the goal point. Selecting steering angles is suitable for wheel-driven robots, but not for walking robots. Thus, we make use of the more general path planning approach developed by Chilian [13]. This method uses a D* Lite [46] path planner, because the robot's knowledge about the terrain changes over time. Thus, the path planner must be able to adapt the path to changes in the map in an efficient way.

As for the A* algorithm, we have to implement a cost function and a heuristic distance function for the D* Lite path planner. The cost function $c(N, N')$ describes the cost for moving from vertex N to its neighbor N'. The heuristic distance function $h(N, S)$ is an estimate of the costs remaining to reach the start vertex from the current vertex N, and must not overestimate the true costs. Thus, we use the direct distance between two nodes for the heuristic distance function. The formulation of the cost function defines the optimality of a path. Often, a path is optimal if it is the shortest path to the goal. In the present work, not only the path length but also the traversability of the path cells should be taken into account. Thus, the cost function for going from vertex N to its neighbor N' is

$$c(N, N') = \sqrt{(N_x - N'_x)^2 + (N_y - N'_y)^2} + \beta \cdot d(N'). \tag{2.64}$$

The first term describes the distance between the vertexes and the second term denotes the danger value of the destination vertex weighted by $\beta > 1$. The bigger the value of β is chosen the longer paths are accepted if they go through safer cells (ref. Fig. 2.18). The costs of going to an untraversable cell are ∞.

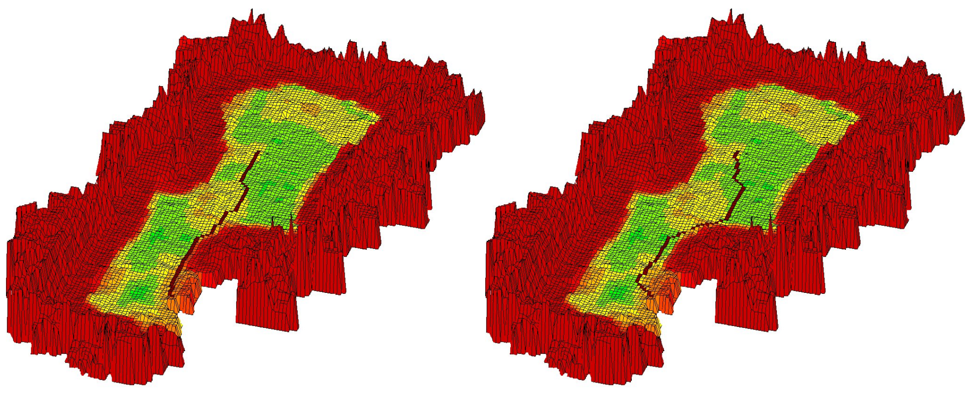

(a) $\beta = 3$ (b) $\beta = 10$

Figure 2.18.: Paths planned with different values of β

The path is replanned whenever the robot gets new information about terrain cells that belong to the planned path.

From the definition of the cost function follows that this method plans paths which are approximately β times longer than the shortest path, if their average danger value is less than $\frac{1}{\beta}$ of the danger of the shortest path. That means that only the relation between path length and path safety is considered but not the absolute danger value of a path. However, if the robot is carrying a heavy load or if its hardware is damaged, the path planner must adapt the path to the changed motion abilities of the robot. To avoid reassessing the traversability of the whole terrain map, a danger value threshold $0 \leq d_{\text{max}} \leq 1$ can be set in the path planner [34]. If the danger value of a cell is higher than d_{max}, we set the costs of moving to that cell to ∞. This increases the safety of the planned path.

2.5. Motion Control

Path following and active exploration are performed as described by Chilian [13]. Path following is achieved by a simple proportional controller which sends the motion commands "move forward", "turn left" and "turn right" as well as the maximum danger value of the upcoming path cells to the robot motion layer. Thus, the robot can adapt its motion strategy to the estimated terrain traversability.

Depending on the horizontal opening angle of the stereo camera, it might be possible that the robot is not able to perceive enough information about the upcoming terrain to calculate the traversability with high certainty. Furthermore, the path could lead into a region that is currently outside the view of the robot but could be perceived if the robot would turn. In these cases, actively exploring the environment of the path is necessary. For this, the motion controller can command exploration turns to the robot. An exploration turn is necessary, if the certainty value of a path cell which is in range of the cameras is lower than 1. During an exploration turn the robot turns over an angular range of 2ϵ so that the cameras cover the robot-sized circular region around the path cell being explored. Since a certainty value of 1 is hard to reach in practice, a set of rules about when exploration turns are permitted has been established [13]:

Between two exploration turns

- the distance between the path cell to be explored and the previously explored path cell must be at least l

and one of the following conditions must hold:

- The robot must have passed a distance of at least l.
- The path cells to be explored in two subsequent exploration turns must be at an angle of at least ϵ given the current robot position is the pivot point.
- The path must have been replanned.

These rules are necessary to avoid that the robot repeats exploration turns when the certainty value does not reach 1. When the camera is mounted on a pan unit, the exploration turns can be performed by turning only the camera instead of turning the whole robot. For the hardware used in the experiments the values were chosen to be $\epsilon = \pi/8$ and $l = 0.2\,\text{m}$.

2.6. Experimental Evaluation

To evaluate the performance of the local navigation method, we used the DLR Crawler, which we already described in Section 2.1.5. In this set of experiments, it was equipped with a wide-angle stereo camera pair, an IMU and it provided 6 DOF leg odometry measurements. We used an external optical tracking system for tracking a target body mounted on the Crawler, which provided ground truth pose measurements. The DLR-Crawler could choose between a simple and fast tripod gait for easy and smooth terrain, and a computationally more expensive adaptive gait with elevator reflexes for overcoming rough terrain [33]. We chose the traversability estimation parameters such that a cell was traversable, if it could be passed by the Crawler using the adaptive gait. The Crawler used the tripod gait by default, but was able to switch the gait pattern according to the estimated traversability of the upcoming terrain.

We created the indoor test environment shown in Fig. 2.19. A gentle slope led into a testbed filled with gravel. We used large stones as untraversable obstacles. Most of the gravel area was easy to pass for the robot. In a smaller region, we increased the terrain difficulty so that the robot could pass this area only using the adaptive gait with the elevator reflexes. We set the goal coordinates $x = 2.8$m and $y = 0.3$m relative to the starting position of the robot.

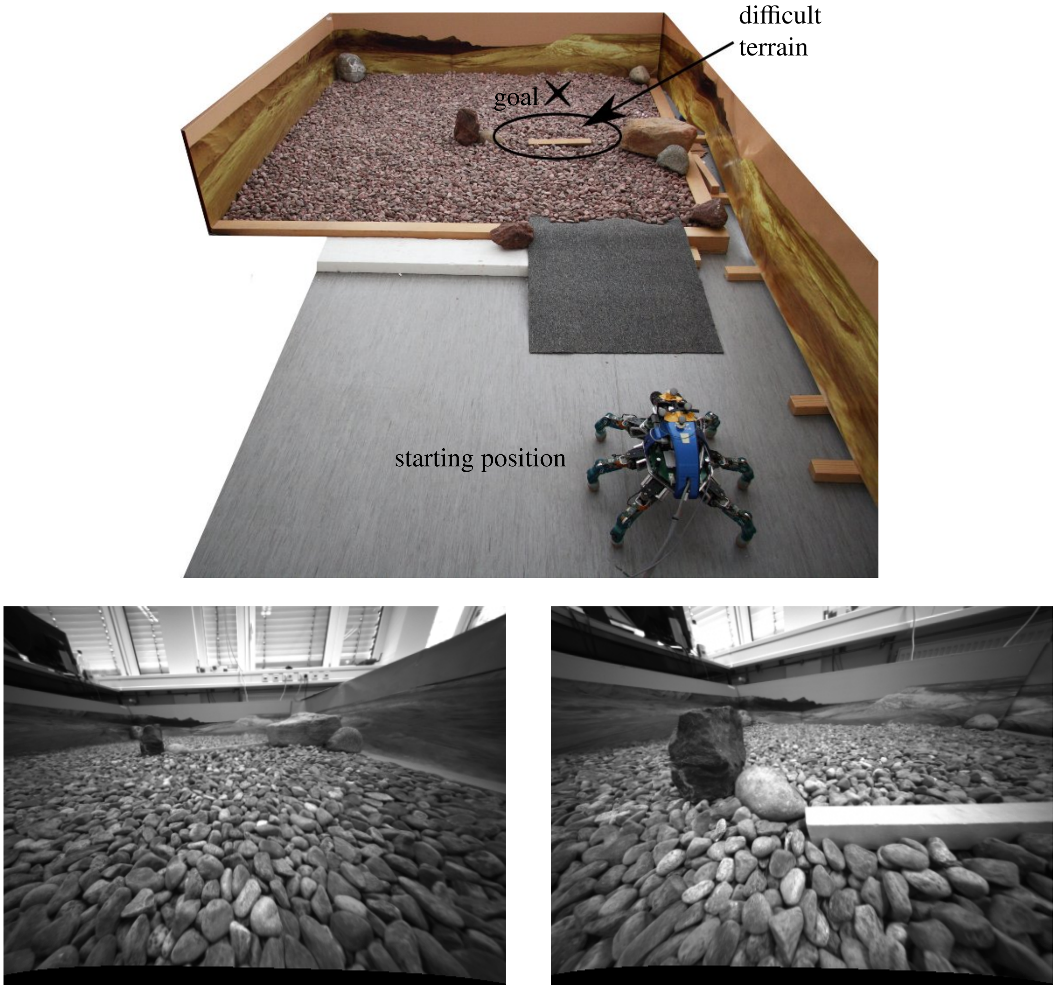


Figure 2.19.: Test setup and sample images as viewed by the stereo camera of the Crawler

In a first experiment, the robot should reach the goal point autonomously without any external disturbances or limitations in its motion capabilities. The resulting cost map, the trajectory recorded by the tracking system and the trajectory estimated by the robot are shown in Fig. 2.20. The map has a resolution of 20mm per grid cell. The colors indicate the traversability of the cells. Red cells are untraversable, green cells are easily traversable and from green to orange the difficulty of traversing a cell increases. As the cost map shows, the robot detected the big stones and the testbed walls as untraversable obstacles. The method for estimating the terrain traversability labels a cell as traversable only if the robot is safe when its center is located on that cell. Thus, a region of half of the robot diameter around each obstacle is also marked as untraversable. This allows the path planner to neglect the size and orientation of the robot but to only plan a path for the center of the robot. Furthermore, the traversability estimation method assigned higher danger values to the slope and the difficult region highlighted in Fig. 2.19 than to the flat gravel areas. These danger values were also sent to the motion layer of the robot, so that the robot could choose an appropriate gait according to the estimated danger of the upcoming terrain. Hence, the robot chose the fast tripod gait for all areas which were estimated to have low danger values and switched to the computationally more expensive adaptive gait for crossing the slope and the difficult terrain. The locations where the robot switched its gait pattern are labeled in Fig. 2.20. This demonstrates the benefits of assessing the terrain traversability at a continuous scale instead of only distinguishing between obstacles and traversable regions.

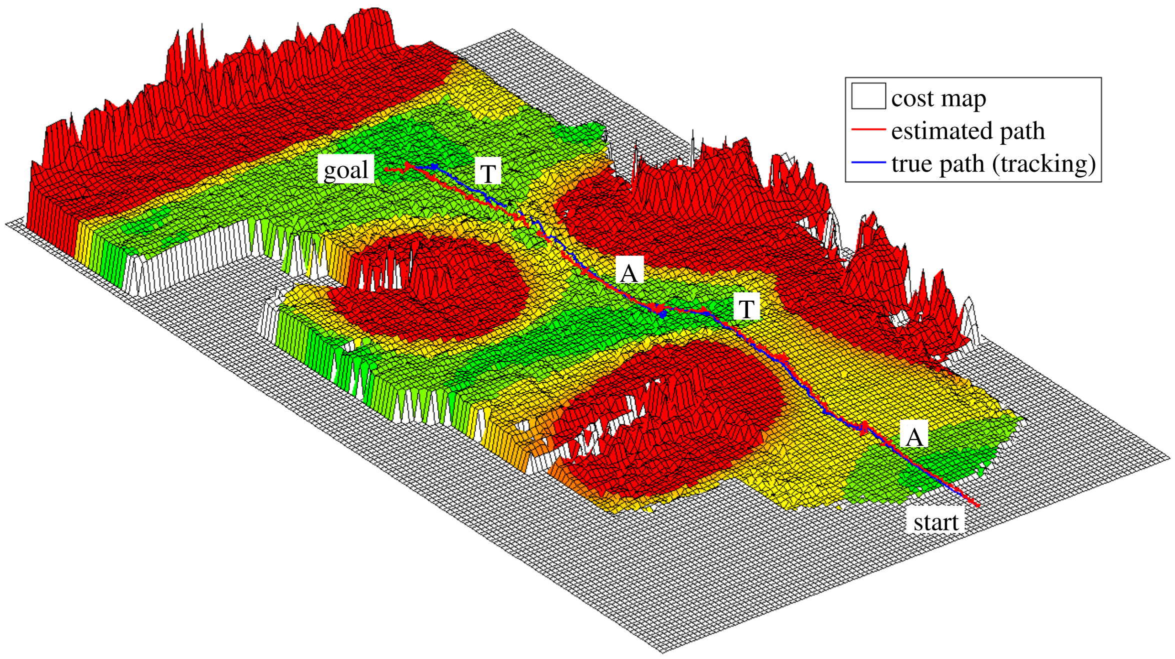
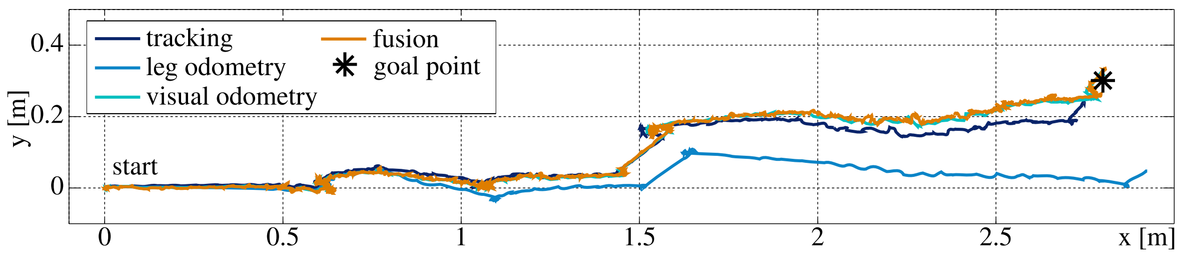
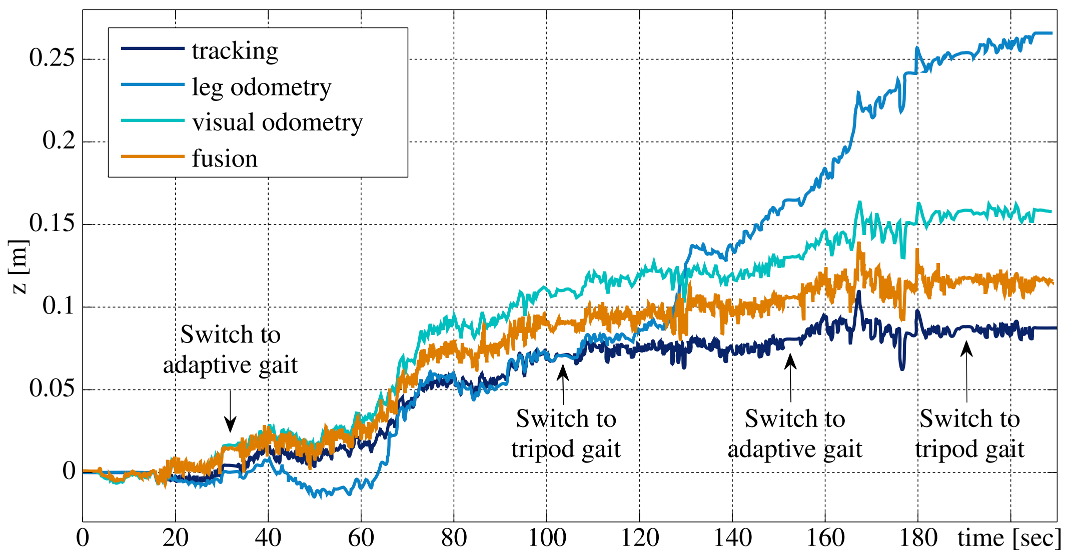


Figure 2.20.: Cost map for run 1 with full motion capabilities and without disturbances. A: switch to adaptive gait. T: switch to tripod gait.

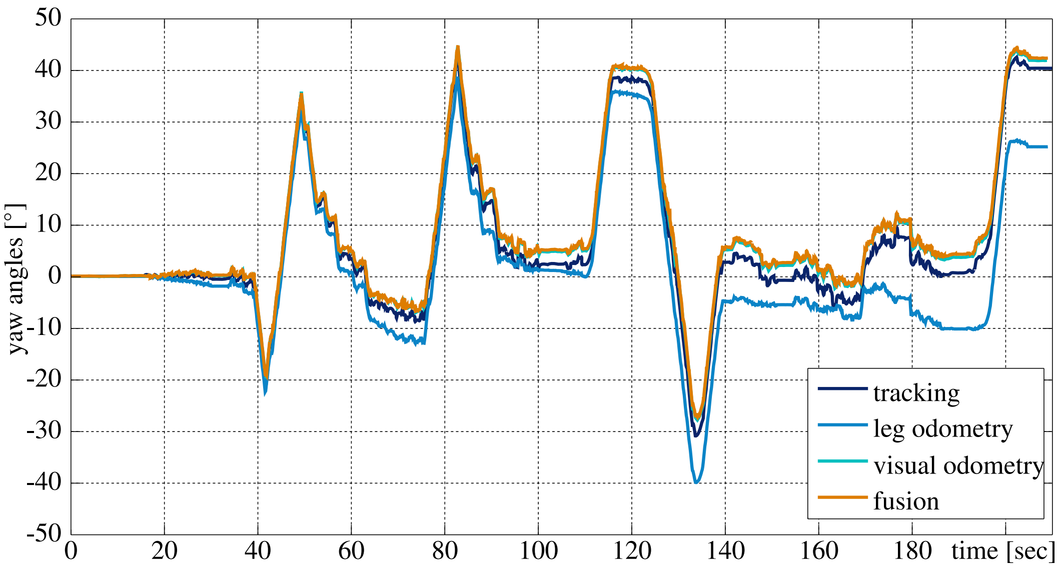
Fig. 2.21 compares the trajectories measured by visual odometry and leg odometry with the true trajectory given by the tracking system and the estimated trajectory obtained by fusing all motion measurements. The trajectory estimated by sensor data fusion is close to the visual odometry trajectory because the error of visual odometry was estimated to be very low. However, both visual odometry and leg odometry suffer from a drift in the z-coordinate as well as yaw angle errors. Due to absolute roll and pitch angle measurements, the error in the z-coordinate of the fusion trajectory is small. The yaw angle error cannot be corrected sufficiently since no absolute yaw angle measurements are available. The yaw angle plot also shows the exploration turns that were performed to gather more information about the upcoming terrain. The robot stopped when it had reached the goal location according to its



(a) Recorded trajectories



(b) z-coordinates



(c) Yaw angles

Figure 2.21.: Trajectories for run 1 with full motion capabilities and without disturbances

estimated pose. The true position of the robot at the goal point was $x = 2.75$m, $y = 0.24$m and $z = 0.09$m. This gives an endpoint error of 0.11m or 3.6% in relation to a path length of about 3.1m. This error is mainly caused by the deviation of the yaw angle. An overview of the experimental results is given in Table 2.1 on page 57.

We used the second test run to demonstrate the robustness of the navigation algorithm against visual disturbances. While the robot was walking through the test environment, we covered its cameras several times using a sheet of paper. All other test conditions remained equal to the previous run. Fig. 2.23 shows the resulting trajectories. The visual odometry trajectory (ref. Fig. 2.23a) had large errors caused by covering the cameras. The leg odometry trajectory also deviated from the true path due to slip. The path estimated by fusing all motion measurements was very accurate. The visual disturbances did not affect the fusion result since visual odometry errors were estimated to be high during these time periods and, thus, these erroneous measurements were given a very low weight in the data fusion process. This can also be seen in Fig. 2.24, because the estimated standard deviation of the position estimate increased strongly during these periods. The resulting cost map in Fig. 2.22 does not show any artifacts or obvious errors. At the goal point, the overall endpoint error was 0.05m.

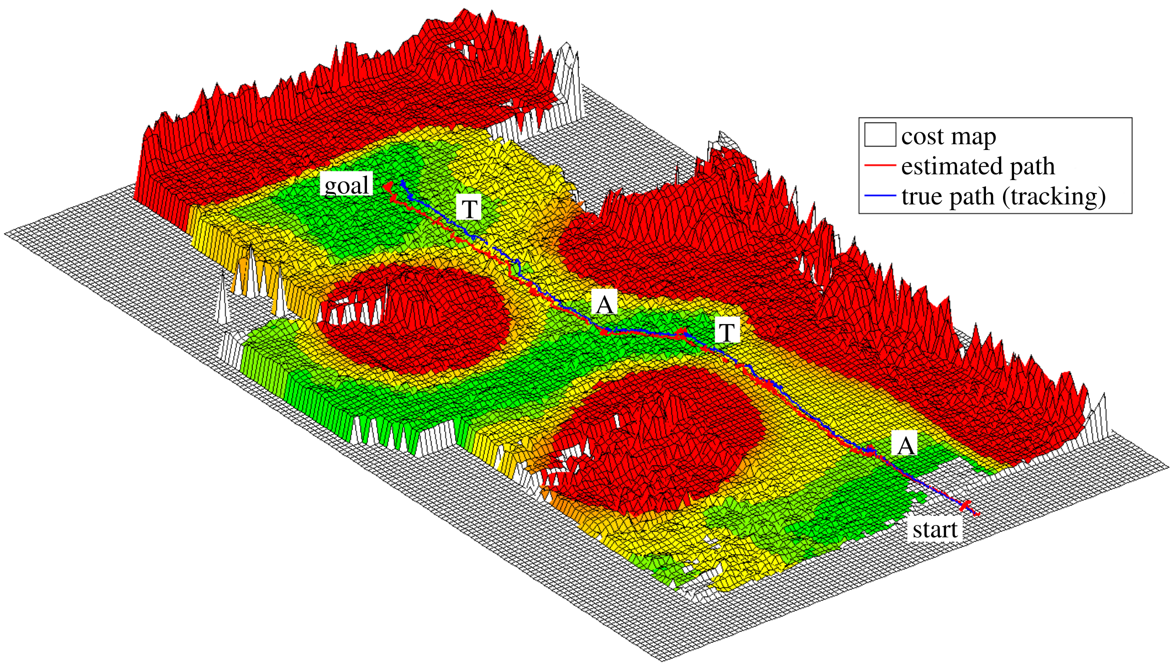
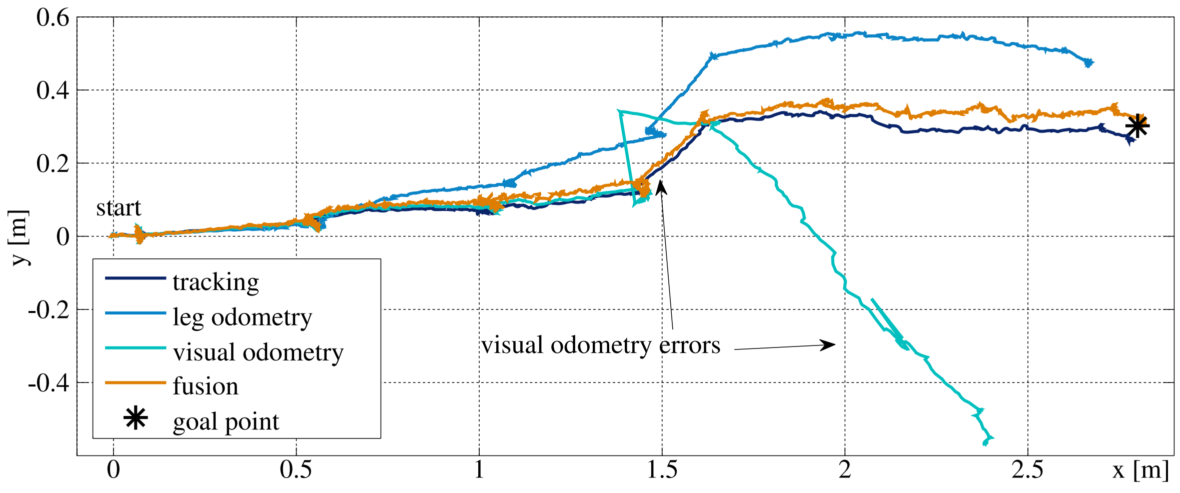
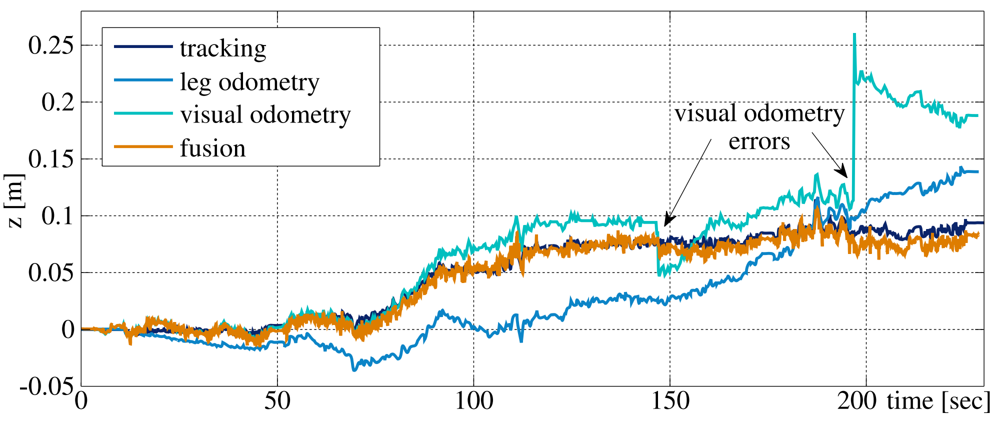


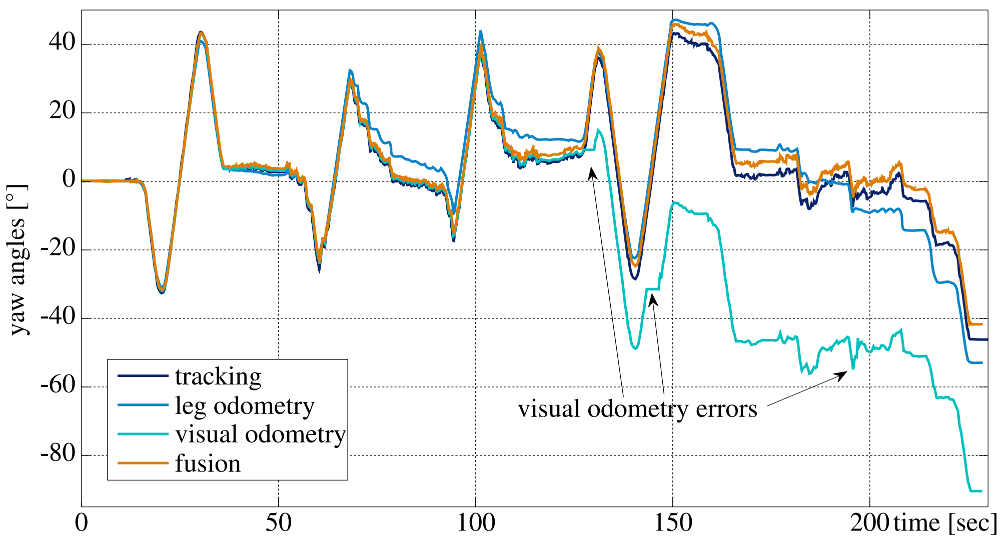
Figure 2.22.: Cost map and trajectories for run 2 after inducing visual odometry errors. A: switch to adaptive gait. T: switch to tripod gait.



(a) Recorded trajectories



(b) z-coordinates



(c) Yaw angles

Figure 2.23.: Trajectories for run 2 with visual odometry errors induced

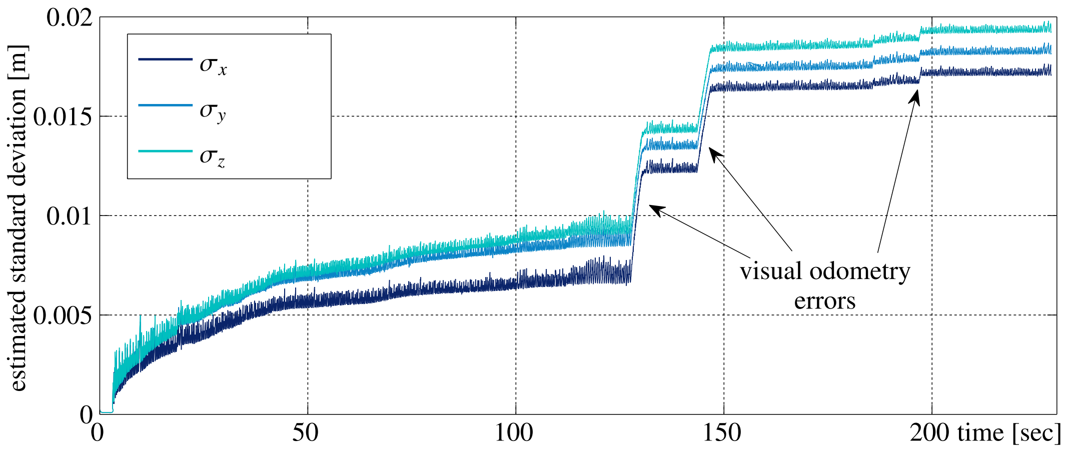
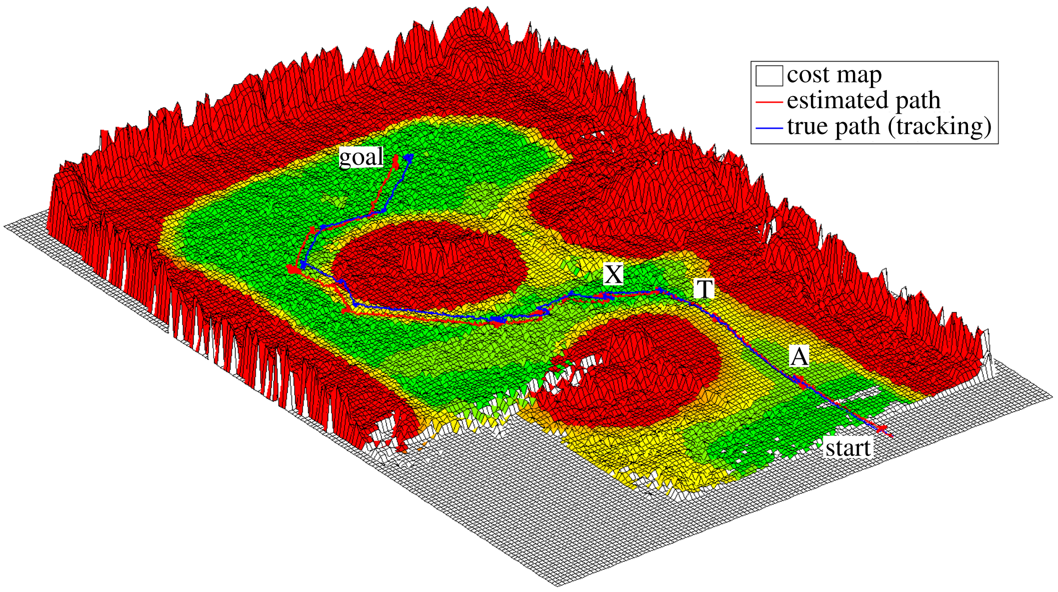


Figure 2.24.: Standard deviations computed from the estimation covariance matrix for run 2 with visual odometry errors induced

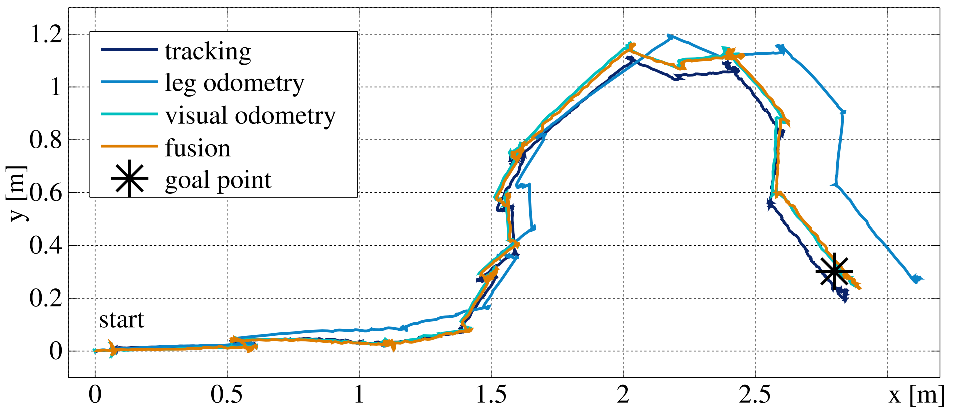
In a final experiment, we simulated that the Crawler picked up a heavy load at one point of its path and had to continue to the goal point with limited motion capabilities. Fig. 2.25 shows the resulting cost map and the trajectories. The robot started moving towards the goal point as in all previous test runs. After passing the slope, we set the danger value threshold to a low value of $d_{\text{max}} = 0.15$ to simulate that the robot was not able to traverse difficult terrain any longer. As a result, the Crawler avoided the difficult area of the testbed and chose the longer but safer path to the goal point. The endpoint error of the estimated position at the goal point was 0.06m. Table 2.1 gives an overview of the experimental results.

Table 2.1.: Results of the local navigation experiments

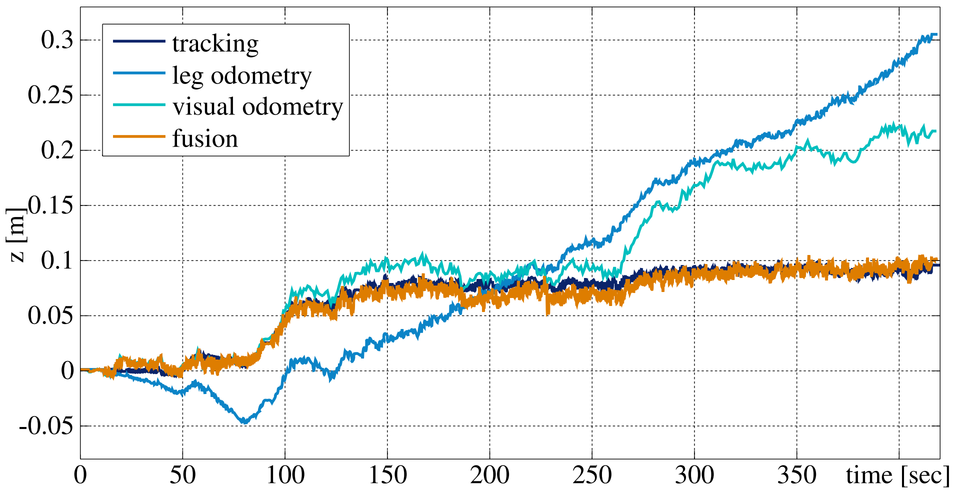
	run 1	run 2	run 3
visual disturbances	-	✓	-
limited motion capabilities	-	-	✓
path length	3.1m	3.1m	4.9m
estimated goal coordinate (x)	2.80m	2.81m	2.81m
estimated goal coordinate (y)	0.33m	0.30m	0.30m
estimated goal coordinate (z)	0.12m	0.08m	0.10m
true goal coordinate (x)	2.75m	2.78m	2.80m
true goal coordinate (y)	0.24m	0.27m	0.24m
true goal coordinate (z)	0.09m	0.09m	0.09m
endpoint pose estimation error	0.11m (3.6%)	0.05m (1.6%)	0.06m (1.2%)



(a) Cost map. A: switch to adaptive gait. T: switch to tripod gait. X: limitation of motion abilities.



(b) Recordered trajectories



(c) z-coordinates

Figure 2.25.: Cost map and trajectories for run 3 with simulated limited motion capabilities

2.7. Conclusion

In this chapter, we proposed a local metric navigation approach for small workspaces. The purpose of the local navigation method in this thesis is to lead the robot along a safe path in a given goal direction. The method solely relies on passive, light-weight onboard proprioceptive and exteroceptive sensors, which are a stereo camera for terrain perception and visual odometry, an IMU for sensing accelerations and turn rates, and sensors providing a robot odometry. We fuse measurements of all these sensors in an indirect information filter to obtain robust and accurate pose estimates. We use the estimated visual odometry errors in the filter framework, so that visual odometry measurements which have a low confidence also have a lower influence on the filter result. We could show in experiments with the DLR Crawler, which provided a 6 DOF leg odometry, that this method gives accurate pose estimates even in the presence of strong visual disturbances. Based on the pose estimates, we build a local geometric model of the surrounding terrain in form of a digital elevation map. For this, we adapted the locus method [53] to create dense elevation maps of arbitrary resolution from stereo disparity images. Furthermore, we applied a traversability estimation method for generating cost maps with continuous traversability costs, which are used to plan short and safe paths through unstructured terrain. We evaluated the complete local navigation method in experiments with the DLR Crawler. We showed that the method allows the robot to reach given goal coordinates on a short and safe path in unknown, unstructured terrain. Even leg slip in loose gravel and visual disturbances did not affect the performance. Thus, the experiments showed that the method is suitable for short-range metric navigation in unknown rough terrain.

3. Global Topological Navigation

The global navigation task aims at enabling the robot to follow a previously traveled path back to its starting position, which is called *homing*. For this, the robot needs to collect information about the traveled path and to store it in a global map. The robot will observe its immediate surroundings again when it reaches a previously visited location, so it does not need to store information about obstacles, but only information that is useful for moving in the correct directions. For this, a topological map is the obvious choice. Since the robot should be able to cover large distances with an onboard map, the global map should be scalable and the computational costs for building and maintaining the map, as well as for navigating using the map, should be small and constant with respect to the traveled distance.

The next section will summarize previous work on visual homing methods. Section 3.2 sketches the approach for solving the global navigation task. Section 3.3 explains the basic principles of viewframe-based navigation, in particular it will introduce and analyze different dissimilarity measures and methods for homing vector calculation. Section 3.4 will review the LT-Map and Section 3.5 will introduce the Trail-Map, an improved data structure for scalable viewframe-based homing. In Section 3.6, we compare the performances of both maps. Section 3.7 will analyze the performance of Trail-Map-based navigation under the presence of realistic sensor noise. A comparison of Trail-Map-based and SLAM-based navigation is given in Section 3.8. Section 3.9 describes the necessary steps for adapting the global navigation method for a real robot. In Section 3.10 we evaluate the performance of the global navigation method on a wheel-driven mobile robot in indoor and long-range outdoor experiments. Section 3.11 concludes this chapter.

3.1. State of the Art

In literature, techniques for retracing learned paths can be divided into appearance based and feature based navigation approaches. In *appearance based* navigation methods, the robot memorizes full images or special image properties in a topological map during a training phase and then navigates by matching the stored information with the current view. Matsumoto et al. [65] introduced a model for route representation called View-Sequenced Route Representation (VSRR). It contains an image sequence of a route along with directional information to the next view. The images are downscaled to meet the memory resources of the system. For creating the VSRR and for localization along the route, images are compared to each other using a correlation method. Winters and Santos-Victor [99] proposed a method for visual indoor navigation. In a training phase, omnidirectional images are recorded and form a topological representation of the environment. The large image database is compressed by Principal Component Analysis, such that only a few eigenimages corresponding to the highest eigenvalues of the image covariance matrix are kept. The image recorded at the current position of the robot is projected into the eigenspace to find the closest recorded image that gives the topological position of the robot. The robot is controlled by visual servoing on the corridor guideline extracted from ground-plane dewarped omnidirectional images. The method presented by Kosecka et al. [50] exploits the properties of man-made environments and uses gradient orientation histograms to describe different indoor locations. After learning, new images are classified using the nearest neighbor method. The method developed by Vardy [93] stores a sequence of snapshot images of the environment along with the odometry motion vectors and uses a combination of both to perform homing. Zhang and Kleeman [104] proposed a navigation system that enables a robot to retrace previously learned routes in the same direction. For this, reference images are captured during a teaching phase. For retracing the path, the current image is compared to the reference image using image cross-correlation performed in the Fourier domain. From this, the orientation difference can be computed and route following is performed by compensating the calculated orientation difference. In their work, a planar ground is assumed.

In contrast to the appearance based methods, *feature-based* approaches have been proposed, which only store the configurations of landmarks in the environment at certain places. The robot then calculates homing vectors to move so that the landmark configuration of its current position gets matched with the one in the goal snapshot. Cartwright and Collett [10, 11] developed the *snapshot* model based on experiments with honey bees. This model stores the perceived landmark configuration at a certain location in a so-called snapshot, which contains the bearing angles and the sizes of the landmarks projected on the insect's retina. For homing, the robot computes the direction that matches the current landmark configuration with the stored snapshot. Dai and Lawton [17] introduced the term *viewframe*, which consists of a set of landmarks and their corresponding bearing angles as they are observed from a certain location. Kawamura et al. [45] transferred the viewframe concept into 3-dimensional space by describing distinct places by the projection of landmarks on the surrounding egosphere.

Other works on retracing paths also make use of features, but are not explicitly inspired by insect navigation models. Argyros et al. [1] proposed robot homing by using only angular information of visual features in panoramic images. No range information or geometric representation is computed. The robot tracks image corners to build up a visual memory containing the life cycle of all features. For homing, the robot selects intermediate milestone positions that allow tracking of at least three image features in-between. Then, it employs a local control strategy to subsequently move to the milestone positions until the home position is reached. Goedemé et al. [29] presented a method for *visual path following* of an automatic wheelchair based on sparsely captured omnidirectional images of the environment. Local 2D maps of image features are created by triangulation and corrected using a SLAM approach, while the robot performs a homing motion to the location of the goal. Šegvić et al. [80] introduced a hierarchical environment representation for appearance based navigation, which contains a graph of key images with extracted 2D features at the top level, and local 3D reconstructions at the bottom level. The information in the top level enables robust navigation by visual servoing, while the bottom level is used for predicting feature locations to support tracking. Using this approach, the robot can cover large distances without a consistent reconstruction of the environment. Furgale and Barfoot [27] proposed *Visual Teach and Repeat*, which enables a robot to follow a taught path over several kilometers. In this work, overlapping feature submaps are created during the teach pass which are used for localization in the repeat pass. Global consistency is not enforced, since local consistency is sufficient for the task. This approach requires about 348MB of data per kilometer on average. In the method introduced by Krajník et al. [51] the robot learns straight line sequences, where each segment is associated with a landmark map, the initial orientation of the robot and the segment length. For homing, the robot uses the landmark map only for correcting its current heading, but then moves straight until it has traversed the segment according to its odometry measurements. Krajník et al. [51] reported that this method required 848MB for a run of 8km length, which means an average of 106MB per kilometer. Cherubini and Chaumette [12] proposed a method in which the robot stores a sequence of key images along its path, such that subsequent images contain common static features. For repeating the path, the robot extracts and matches common features in the current and the goal image and moves to align the x-coordinates of the centroids of the feature point clouds.

All works mentioned above fail to give information about how to scale the resulting topological maps and how to efficiently organize the information about the snapshots or viewframes to save memory and computation time. To my knowledge, the first work addressing the problem of scalability and memory efficiency for feature-based homing is the *Landmark Tree-Map (LT-Map)* developed by Augustine et al. [2]. It is based on the snapshot concept but uses the term *viewframe* to describe a configuration of landmarks and their respective angles at a certain location in space. The LT-Map organizes landmark views in a tree so that slowly changing, translation invariant landmarks are located towards the top of the tree while translation variant landmarks are located in the leaves. The LT-Map can be scaled by pruning the tree and, thus, discarding the information about quickly changing landmarks.

3.2. Approach

In this chapter, we develop a global navigation method based on the idea of the LT-Map. This method is independent of any distance information, but solely relies on bearing angle measurements to landmarks, which are extracted from an omnidirectional sensor. The robot memorizes the landmark bearing configurations, so-called *viewframes*, of certain locations and stores them in a non-redundant data structure, which can easily be pruned in case of memory shortage. For homing, the robot retrieves the viewframes and computes homing vectors which subsequently lead the robot to the previous viewframe until the original home position is reached. An overview of the global navigation method is shown in Fig. 3.1. Using this approach the robot can reliably retrace long-range paths without the need to maintain a metrically correct Cartesian map. Thus, the method runs in constant time independent of the length of the path and is suitable for robots with limited computational resources.

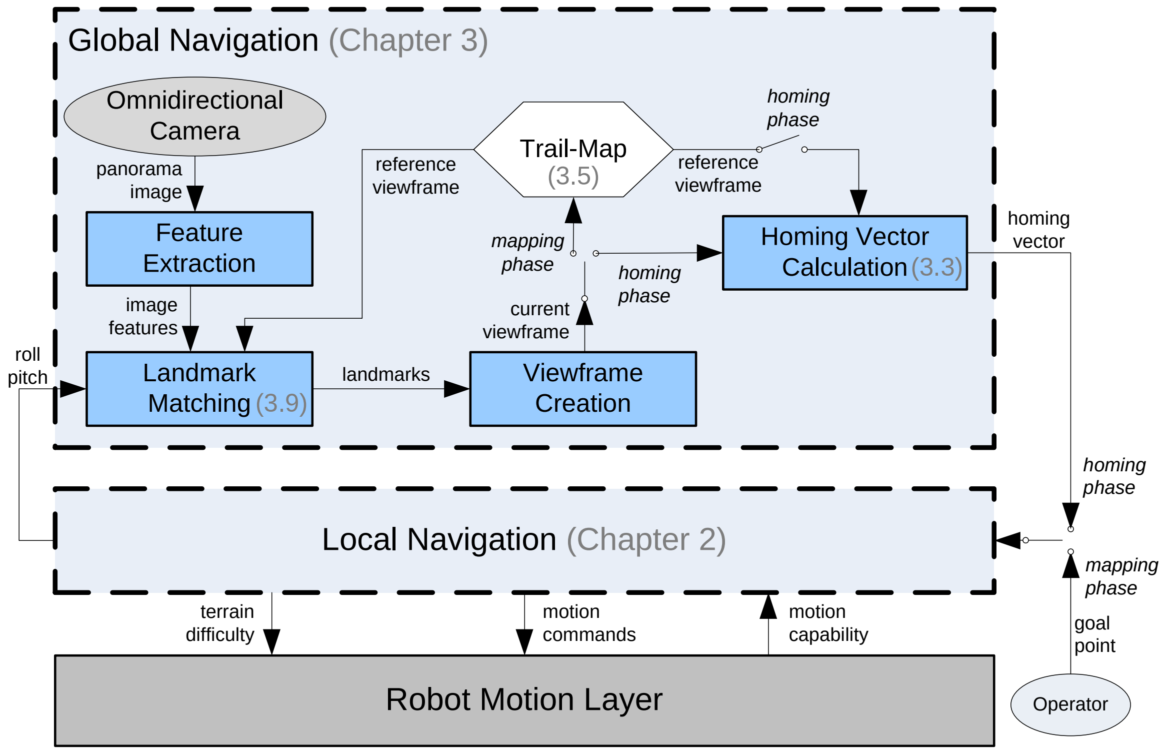


Figure 3.1.: Overview of the global navigation method with section numbers in brackets

3.3. Viewframe-Based Navigation

A viewframe (VF) is defined as the configuration of landmark views which corresponds to a certain location in two- or three-dimensional Euclidean space (ref. Fig. 3.2). Each landmark view (LV) contains the landmark's ID, its descriptor and its bearing angle containing the azimuth $\phi_{i,a}$ (and elevation $\phi_{i,e}$ in the 3D case) under which the landmark L_i is observed. The landmark views are extracted from omnidirectional images. All viewframes are assumed to be rotationally aligned with each other, either using compass information or an estimated orientation. The unit vector pointing in the direction of landmark L_i is $\boldsymbol{l}_i$ and can be computed from $(\phi_{i,a}, \phi_{i,e})$ as

$$\boldsymbol{l}_i = [\cos\phi_{i,e}\cos\phi_{i,a}, \quad \cos\phi_{i,e}\sin\phi_{i,a}, \quad \sin\phi_{i,e}]^T \, . \tag{3.1}$$

The robot records viewframes during a learning phase while it explores unknown regions either autonomously or remotely controlled by an operator. A *dissimilarity measure* is required to decide when a new viewframe is recorded. At some point, the robot is commanded to return to its starting position. For this, it has to compute *homing vectors* that successively

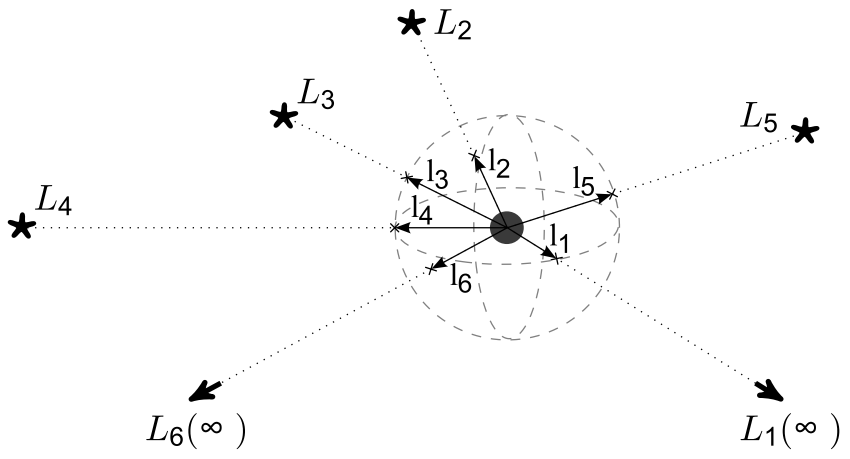


Figure 3.2.: Illustration of a viewframe (adapted from [63]). L_i are the landmarks and $\boldsymbol{l}_i$ are the unit vectors pointing to them.

give the direction to the previously recorded viewframe until the viewframe corresponding to the starting position is reached. The robot decides whether a viewframe is reached based on another dissimilarity measure, which not necessarily needs to be the same measure as for viewframe recording.

3.3.1. Viewframe Dissimilarity Measures

In the mapping phase the robot has to detect when the current view becomes significantly different from the viewframe that it previously recorded. Additionally, during homing it is important for the robot to recognize a known place or to know when the desired goal viewframe is reached. Thus, a measure of viewframe dissimilarity has to be computed.

There are different ways for computing a dissimilarity measure δ_{diss}. One possible choice is the root mean square error between the N corresponding landmark unit vectors $\boldsymbol{l}'_i$ of the current view and $\boldsymbol{l}_i$ of the goal viewframe:

$$\delta_{\text{diss}}^{\text{rmse}} = \frac{1}{N}\sqrt{\sum_{i=1}^{N}\left(\boldsymbol{l}'_i - \boldsymbol{l}_i\right)^2}. \tag{3.2}$$

Furthermore, we can compute the mean absolute errors as

$$\delta_{\text{diss}}^{\text{abs}} = \frac{1}{N}\sum_{i=1}^{N}\left|\boldsymbol{l}'_i - \boldsymbol{l}_i\right|. \tag{3.3}$$

The average angle between the corresponding unit vectors also gives a dissimilarity measure:

$$\delta_{\text{diss}}^{\text{ang}} = \frac{1}{N} \sum_{i=1}^{N} \operatorname{acos}(\boldsymbol{l}_i'^T \boldsymbol{l}_i). \tag{3.4}$$

Another way for computing a dissimilarity measure is using the maximum value of all difference angles between corresponding unit vectors. To prevent large errors due to outliers or strong noise from corrupting the dissimilarity measure, we do not use the maximum value but the *k*th maximum value:

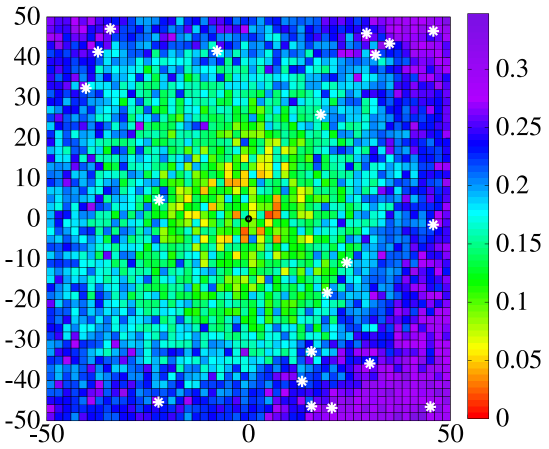
$$\delta_{\text{diss}}^{\text{max}} = k\text{th-max}\left\{\operatorname{acos}(\boldsymbol{l}_i'^T \boldsymbol{l}_i)\right\}, \tag{3.5}$$

where the value for k can be set according to the number of corresponding landmarks, for example as a ratio.

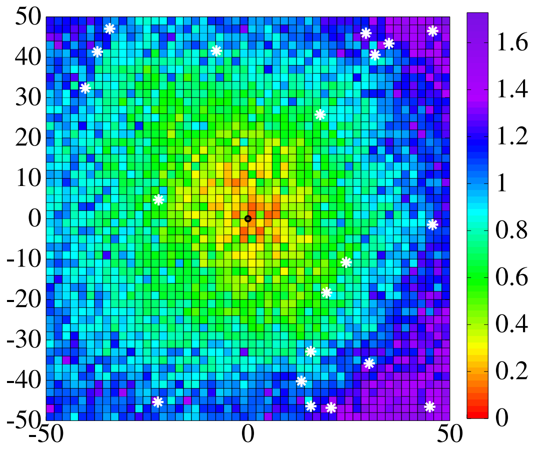
Fig. 3.3 shows the different dissimilarity measures in an environment with 20 landmarks under the presence of noise. As can be seen, the root mean square error measure $\delta_{\text{diss}}^{\text{rmse}}$ and the *k*th maximum measure $\delta_{\text{diss}}^{\text{max}}$ are strongly affected by the measurement errors. The average angle error measure $\delta_{\text{diss}}^{\text{ang}}$ and the mean absolute error measure $\delta_{\text{diss}}^{\text{abs}}$ perform very similar.

The robot records a new viewframe when the dissimilarity measure of the current view compared to the previously recorded viewframe exceeds the threshold $\xi_{\delta_{\text{diss}}}$. Thus, the density of the viewframes depends on the local landmark configuration: In areas with only few close landmarks, the viewframes will be further apart than in areas with many nearby landmarks. That leads to an implicit adaption of the map resolution to the local conditions.

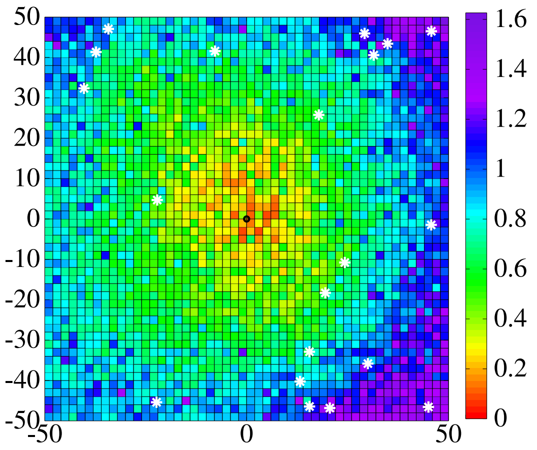
The smaller the threshold $\xi_{\delta_{\text{diss}}}$, the higher the resolution and, hence, the accuracy and the memory requirements of the resulting map. The threshold should be chosen according to the measurement accuracy of the bearing sensor and to the required path accuracy. Since usually no high accuracy is required for the traversal between different workspaces, higher thresholds should be preferred for the benefit of less memory.



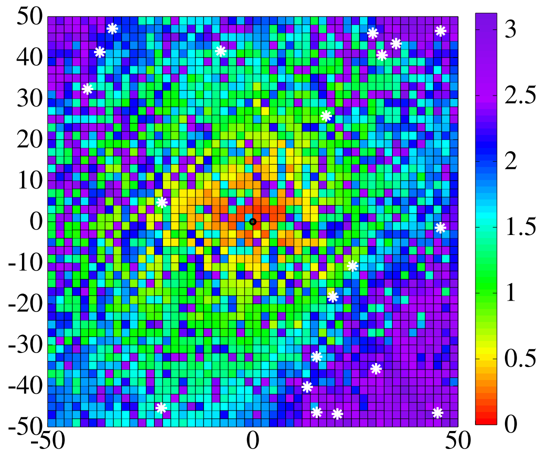
(a) $\delta_{\text{diss}}^{\text{rmse}}$



(b) $\delta_{\text{diss}}^{\text{abs}}$



(c) $\delta_{\text{diss}}^{\text{ang}}$



(d) $\delta_{\text{diss}}^{\max}$ with $k = 2$

Figure 3.3.: Dissimilarity measures for 20 landmarks (white asterisks) under the presence of noise (1.0°), outliers (10%) and occlusions (10%). The black circle at (0, 0) is the reference location.

3.3.2. Homing Vector Calculation Methods

To move from the current position to a goal viewframe, the robot has to compute the moving direction from the landmark angle information that it currently perceives compared to the stored configuration in the viewframe. The resulting direction is usually represented by a vector, called the *homing vector*.

In literature, different methods for calculating homing vectors from the current view to a goal view have been proposed. Apart from image-based methods, where whole images are compared for homing vector calculation [24, 103] this section will focus on the landmark-based methods for homing vector calculation.

Snapshot Model

The original *snapshot model* developed by Cartwright and Collett [10] assumed that the projection of landmarks forms dark and bright sectors on the insect's retina. The method uses the apparent sizes and bearings of these sectors to infer the homing direction. As Fig. 3.4a shows, the sectors of the current view are matched with the closest sectors of the same sign (dark or bright) of the goal snapshot, which can cause mismatches. Then, tangential unit vectors perpendicular to the bisecting line of each sector of the goal viewframe are generated, which point in the direction to align the paired sectors. Radial unit vectors are parallel to the bisecting lines of the sectors of the goal viewframe and act to reduce the differences in the apparent sizes. The views are assumed to be rotationally aligned by an external reference, e.g. a compass. The overall homing vector is computed as the sum of all tangential and radial unit vectors.

Proportional Vector Model

Lambrinos et al. [54] modified the snapshot model by taking the magnitudes of the differences in bearings and apparent sizes into account as shown in Fig. 3.4b. The resulting model is called *proportional vector model* since the lengths of the contributing vectors are proportional to the magnitude of the differences. In the proportional vector model, the landmarks are also matched by pairing the closest sectors in the snapshots.

Average Landmark Vector Model

As a further step, Lambrinos et al. [54] developed the *average landmark vector (ALV) model* [54] (ref. Fig. 3.5a). This model does not require the whole snapshots. Instead, it calculates only an average vector to all the landmarks in each snapshot and compares these ALVs. The homing vector simply becomes the difference between the current ALV and the ALV at the goal snapshot. The vectors to the landmarks can either point to the center of the landmark or to the edges so that the landmark size does not have to be considered. Hence, in the ALV model the apparent size of the landmarks is considered implicitly by the bear-

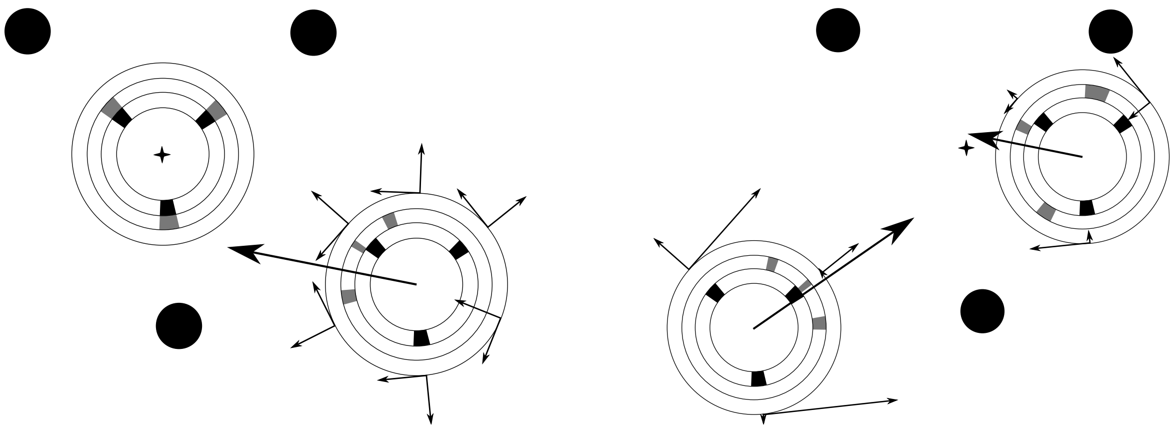

(a) Snapshot model (b) Proportional vector model

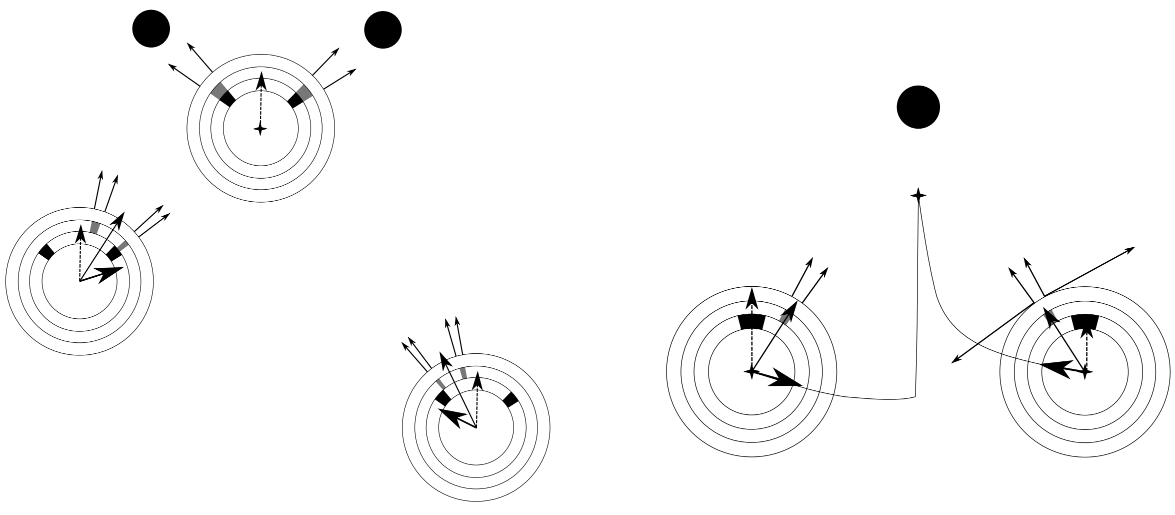
Figure 3.4.: Snapshot model and proportional vector model (images adapted from Lambrinos et al. [54]). The cross denotes the goal location. The inner ring shows the sectors of the goal viewframe, the outer ring shows the sectors of the current view. Closest sectors of the same sign (bright or dark) are matched. Vectors originate at the outer ring in the direction of the bisecting line of each dark or bright sector of the goal viewframe and perpendicular to it. The direction of the tangential vectors is given by the bearing angle difference of the matched sectors. The direction of the radial vectors is given by the size differences of the matched sectors. In the snapshot model, all vectors are unit vectors. In the proportional vector model the lengths of the vectors are proportional to the bearing and size differences, respectively.

ing differences. To increase the influence of the landmark size on the homing vector result, Lambrinos et al. [54] proposed to add perpendicular vectors to each edge, which artificially move the edges apart. This results in straighter homing paths when only few landmarks are present in the environment (ref. Fig. 3.5b). The method assumes that the same landmarks are visible from both snapshot positions. However, Lambrinos et al. [54] also mentioned that the ALV model seems to tolerate occlusions of landmarks. Furthermore, the landmark sectors do not have to be matched, but the ALV model implicitly establishes the matches. The ALV model works well for artificial landmarks, but not for real image data, since it relies on a very robust landmark detection [94].

Difference Vector Model

Assuming that landmark sectors can be matched correctly, the result of the ALV method is equal to the result of the *difference vector model* [54] (ref. Fig. 3.6). The difference vectors are computed as the differences between the unit vectors pointing to the landmarks in the current view and the unit vectors pointing to the closest landmarks in the goal view. Here, the lengths of the difference vectors are also a function of the bearing distance as in the proportional vector model. However, the difference vectors are secant vectors instead of tangential vectors.

The methods for calculating a homing vector presented so far all assume that a landmark has a perceivable size that does only change with the distance from the landmark but not with the bearing the landmark is perceived at. That is only true for cylindrical objects which can clearly be distinguished from the background. This is not always the case in real world scenarios. Landmarks that are detected using feature detection algorithms appear as characteristic points in images. They can have a scale but they usually do not have a size which is clearly perceivable.



(a) ALV model

(b) ALV model with and without increased size information

Figure 3.5.: ALV model (images adapted from [54]). The cross denotes the goal location. The inner ring shows the sectors of the goal viewframe, the outer ring shows the sectors of the current view. The dashed arrow is the ALV of the goal snapshot. The thin solid arrows originating from the center of each view are the current ALVs, computed as the average of all unit vectors pointing in the direction of a sector border (small arrows on the outer ring) and perpendicular to it (b). The difference between the current ALV and the goal ALV is the resulting homing vector (thick arrow).

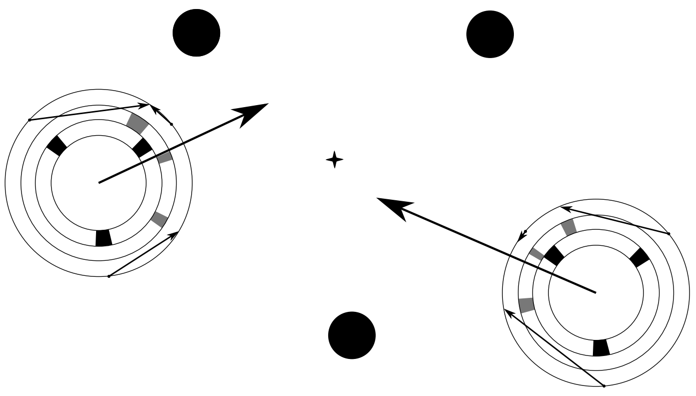


Figure 3.6.: Difference vector model (image adapted from [54]). The cross denotes the goal location. The inner ring shows the sectors of the goal viewframe, the outer ring shows the sectors of the current view. Closest sectors of the same sign (bright or dark) are matched. Difference vectors are secant vectors computed as the differences between the unit vectors pointing to the matching landmark sectors (only the vectors originating from the dark sectors are shown for clarity reasons).

Considering landmarks as point image features without a perceivable size, we get the homing vector $\boldsymbol{h}$ from the difference vector model as

$$\boldsymbol{h} = \frac{1}{N} \sum_{i=1}^{N} (\boldsymbol{l}'_i - \boldsymbol{l}_i) \ , \tag{3.6}$$

where $\boldsymbol{l}'_i$ are the unit landmark vectors in the current view and $\boldsymbol{l}_i$ are the corresponding unit landmark vectors of the goal view. Here, the sum of the difference vectors is normalized by the number of corresponding landmarks N. Fig. 3.7 visualizes the resulting streamlines and the deviations from the direct path to the home position.

Improved Difference Vector Model

When using the difference vector model, the difference vectors are always secants of the unit circle around the current viewframe. For this method to work well, it is assumed that the landmarks are distributed isotropically around each viewframe and that a 360° panorama of the scene is taken. Then, errors in the orthogonal direction to the homing vector cancel each other out. When landmarks are located only in one direction of the viewframe, errors cannot cancel out. Hence, the homing vector is biased, especially when the landmarks appear only in the homing vector direction. This behavior is shown in Fig. 3.8a, where the homing vectors near the connecting line between the landmark cluster and the home position

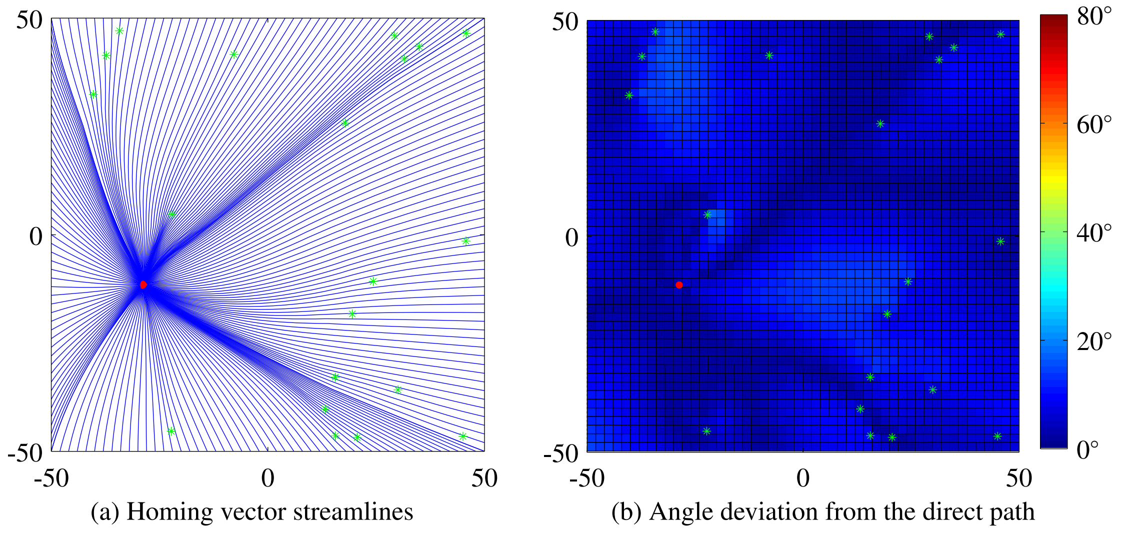


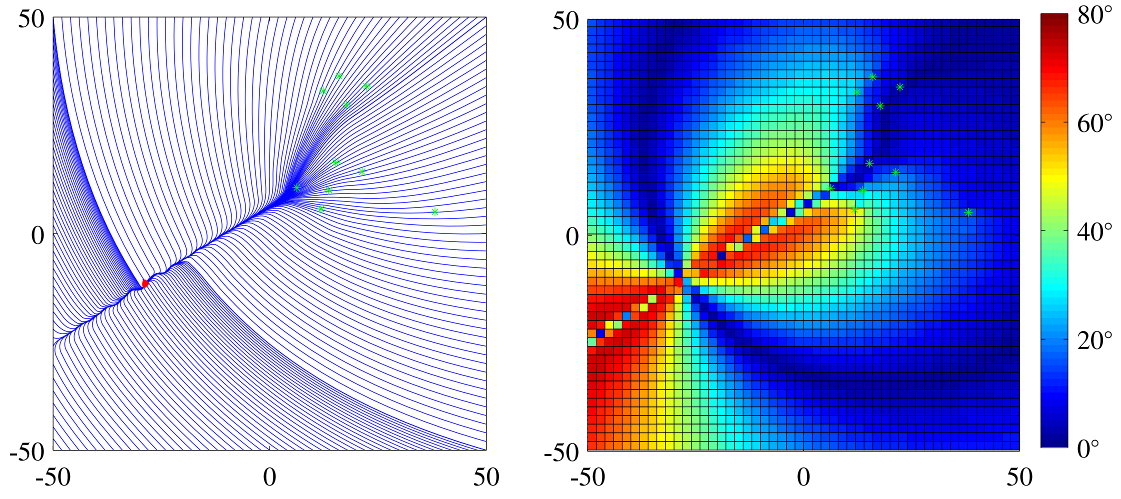
Figure 3.7.: Difference vector model (ref. Eq. (3.6)): Homing vector streamlines and deviations from the direct path. Green asterisks: Landmarks. Red circle: Home location. The x and y coordinates are given in units.

are approximately perpendicular to the desired homing direction. This leads to zigzag-like viewframe approaching behaviors (see Fig. 3.10a).

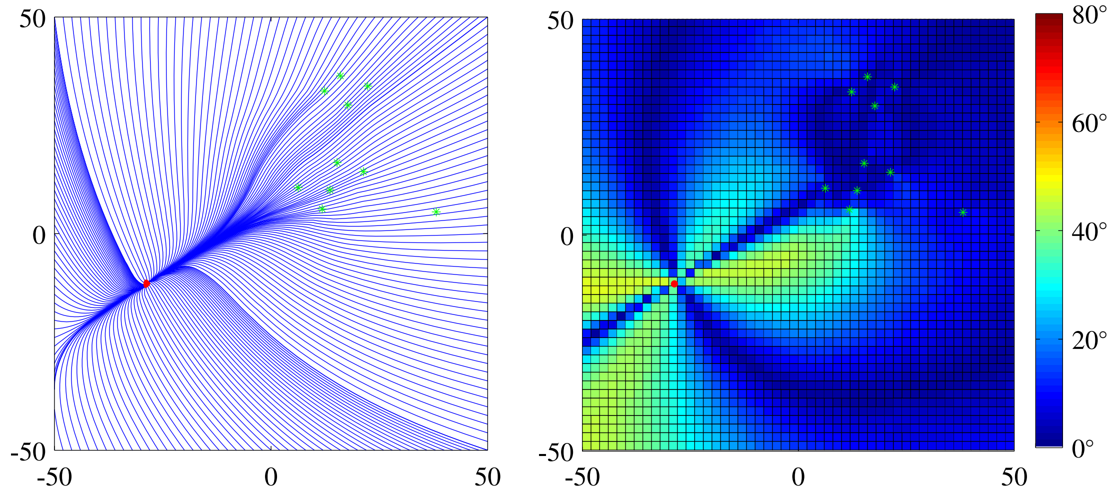
Creating radial homing vector components using the apparent size of the landmark would solve this problem. However, when landmarks are assumed to be points, they do not have an apparent size. In that case, the apparent width of an imaginary landmark between two landmark observations can be used to achieve more direct homing vectors. We used the angles between two landmarks and the robot as apex to generate radial homing vector components. When moving towards two landmarks, the angle between them increases. Hence, we add a component x_i in the positive direction of the bisecting line of the two landmarks when the angle between those landmarks in the goal viewframe is greater than in the current viewframe. Otherwise, we subtract the component:

$$\boldsymbol{h} = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{l}'_i - \boldsymbol{l}_i) + \frac{1}{N-1}\sum_{i=1}^{N-1}\frac{\left(\boldsymbol{l}'_i + \boldsymbol{l}'_{i+1}\right)}{2}x_i \tag{3.7}$$
$$x_i = \left|\mathrm{acos}(\boldsymbol{l}_i^T \boldsymbol{l}_{i+1})\right| - \left|\mathrm{acos}(\boldsymbol{l}'^T_i \boldsymbol{l}'_{i+1})\right| = \beta_{i,i+1} - \beta'_{i,i+1}.$$

The construction of a homing vector with and without the angle differences is illustrated in Fig. 3.9 for an environment with only two landmarks.



(a) Difference vector model (ref. Eq. (3.6))



(b) Improved difference vector model (ref. Eq. (3.7))

Figure 3.8.: Homing vector streamlines (left) and angle deviations from the direct path (right) for the difference vector model and improved difference vector model with a landmark cluster (green asterisks). The x and y coordinates are given in units.

Fig. 3.8b shows the resulting streamline and angle deviation plot. The homing vectors have now improved when only landmarks in the direction of movement are available as shown in Fig. 3.10b. However, the width of the imaginary landmarks not only decreases when the robot travels away from the landmarks, but also when the imaginary landmarks are perceived at a flat angle. Hence, in these areas the resulting homing vectors are erroneous.

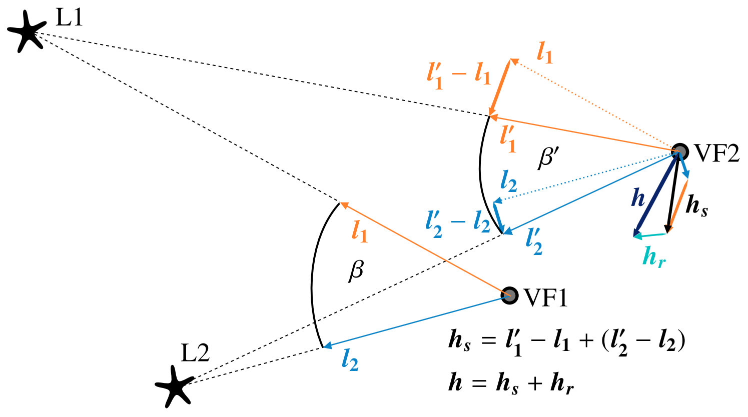
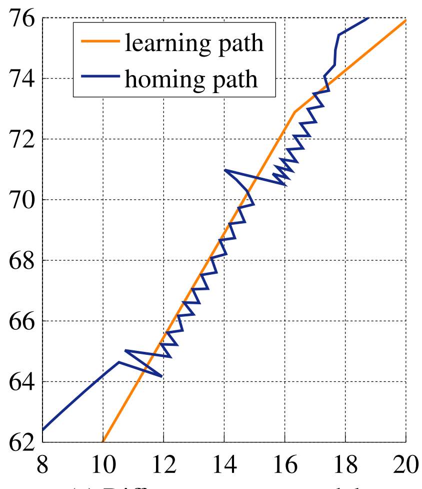
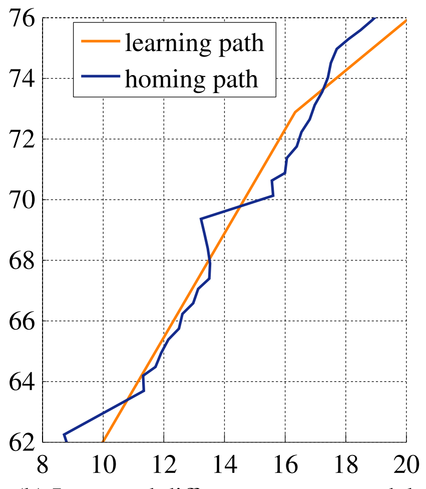


Figure 3.9.: Homing vector construction: $\boldsymbol{h}_s$: homing vector using difference vector model with only secant components. $\boldsymbol{h}_r$: radial component from angle differences. $\boldsymbol{h}$: homing vector using difference vector method and angle differences



(a) Difference vector model



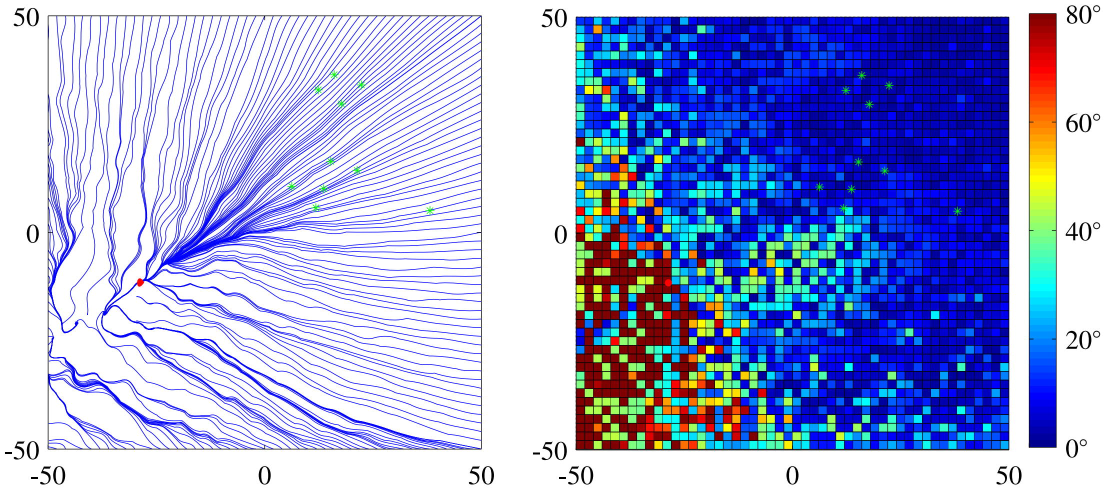
(b) Improved difference vector model

Figure 3.10.: Resulting homing paths using the difference vector model and the improved difference vector model. The x and y coordinates are given in units.

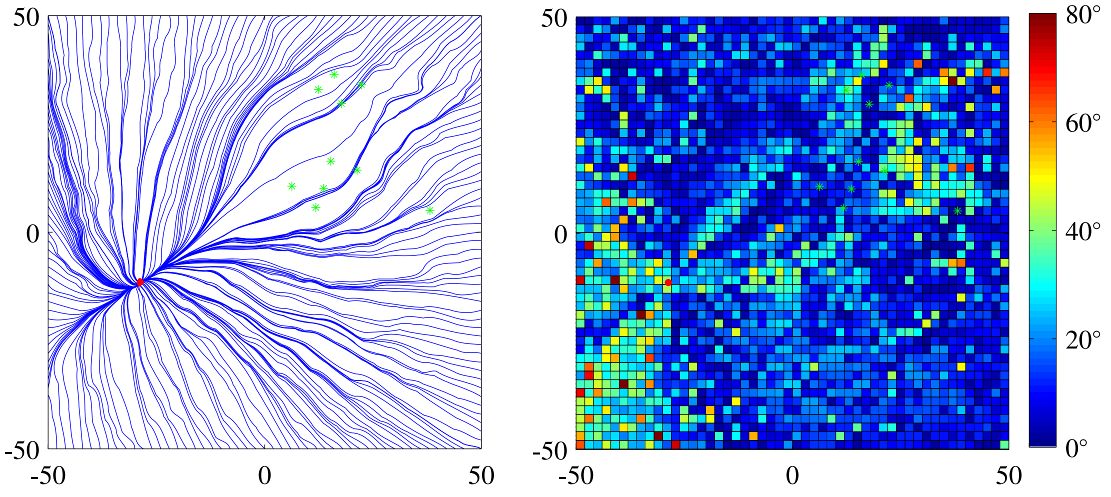
Normalized Difference Vector Models

This section presents another variant of the difference vector model, which is called normalized difference vector model. In contrast to the original difference vector model, this method calculates the homing vector by summing the normalized difference vectors as

$$\boldsymbol{h} = \frac{1}{N} \sum_{i=1}^{N} \frac{(\boldsymbol{l}'_i - \boldsymbol{l}_i)}{|\boldsymbol{l}'_i - \boldsymbol{l}_i|} . \tag{3.8}$$



(a) Improved difference vector model (ref. Eq. (3.7))



(b) Normalized improved difference vector model (ref. Eq. (3.9))

Figure 3.11.: Homing vector streamlines (left) and angle deviations from the direct path (right) for the improved difference vector model and the normalized improved difference vector model with a landmark cluster (green asterisks) and 10% outliers. The x and y coordinates are given in units.

We can apply the same kind of normalization to the improved difference vector model:

$$\boldsymbol{h} = \frac{1}{N}\sum_{i=1}^{N}\frac{\left(\boldsymbol{l}'_i - \boldsymbol{l}_i\right)}{\left|\boldsymbol{l}'_i - \boldsymbol{l}_i\right|} + \frac{1}{N-1}\sum_{i=1}^{N-1}\frac{\left(\boldsymbol{l}'_i + \boldsymbol{l}'_{i+1}\right)}{\left|\boldsymbol{l}'_i + \boldsymbol{l}'_{i+1}\right|}\operatorname{sign}(x_i) \tag{3.9}$$

$$x_i = \left|\operatorname{acos}(\boldsymbol{l}_i^T \boldsymbol{l}_{i+1})\right| - \left|\operatorname{acos}(\boldsymbol{l}'^T_i \boldsymbol{l}'_{i+1})\right| = \beta_{i,i+1} - \beta'_{i,i+1}.$$

The advantage of the normalization becomes apparent when false landmark matches occur. Since false landmark matches are likely to yield unit vectors pointing in a very different direction than the unit vector corresponding to the true match, the resulting difference vectors are often large. Thus, these false matches have a very strong influence on the homing vector direction. We can decrease this influence when we use all difference vectors in the normalized form. Fig. 3.11a shows the streamlines and deviations of the homing vectors computed by the improved difference vector model when 10% landmark outliers are present. In this case, homing would most probably fail when the robot starts at the lower left corner of the plot. In contrast, with the normalized version, the homing vectors are more stable in the presence of outliers (ref. Fig. 3.11b). The normalized improved difference vector model is related to the original snapshot model [10], which also uses unit vector components according to the apparent size of a landmark and according to the bearing changes.

Tangential Correction Vector Method

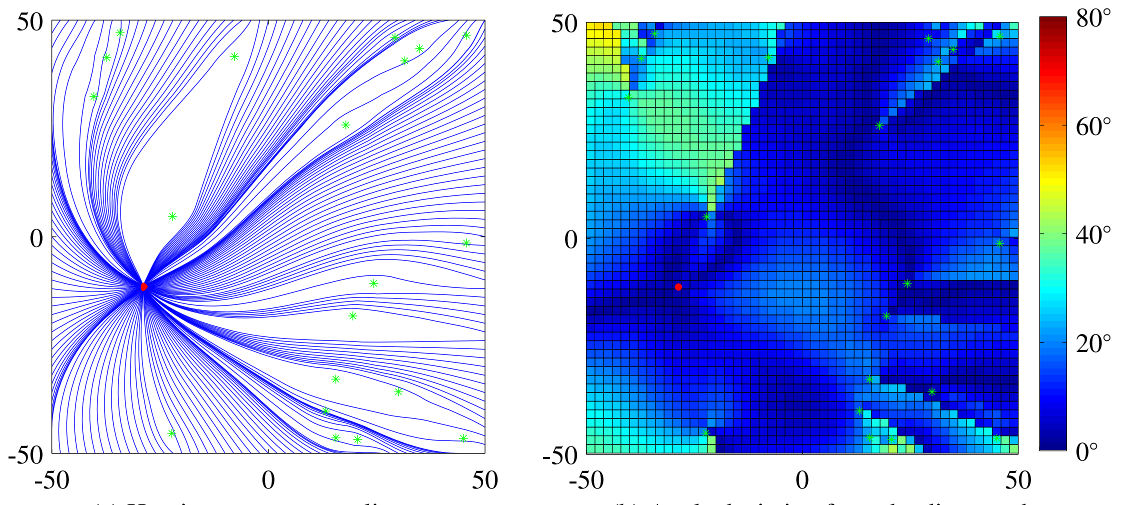
Weber et al. [98] proposed a method for homing vector calculation based on tangential correction vectors that are perpendicular to the current landmark bearing ϕ'_i and proportional to the difference between the current and the goal bearing ϕ_i as

$$\boldsymbol{h} = \sum_{i=1}^{N} \left|\phi_i - \phi'_i\right| \boldsymbol{l}'_{i\perp} \quad \text{with} \quad \boldsymbol{l}'_{i\perp} = \begin{cases} \boldsymbol{R}(90°)\boldsymbol{l}'_i & \text{if} \quad \phi_i < \phi'_i \\ \boldsymbol{R}(-90°)\boldsymbol{l}'_i & \text{if} \quad \phi_i \geq \phi'_i \end{cases} \tag{3.10}$$
$$\text{and} \quad \boldsymbol{R}(\alpha) = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix}.$$

The resulting streamlines and path deviation plots shown in Fig. 3.12 reveal that this homing vector calculation method leads to a landmark avoiding behavior. This is beneficial when the landmarks are also obstacles, which is not the case when using image features as landmarks, since image features can also be detected on flat, but textured ground.

Image Based Visual Servoing (IBVS) Methods

Another way for calculating homing vectors originates from the visual servoing background. In visual servoing, the velocity of an end-effector or a robot is controlled by the visual deviation of the current scene from a reference scene. Transformed to the navigation task this means that an error function has to be defined which contains the deviation of the currently perceived landmark configuration from a goal landmark configuration. The robot's velocity is then controlled in a way that the error function decreases.



(a) Homing vector streamlines (b) Angle deviation from the direct path

Figure 3.12.: Tangential correction vector method (ref. Eq. (3.10)): Homing vector streamlines and angle deviations from the direct path to the home position (red circle). Green asterisks: Landmarks. The x and y coordinates are given in units.

IBVS Based on Bisecting Homing Vector Components Liu et al. [59] introduced a method for homing vector calculation based on bearing-only landmark information in 3D. It uses the angles β_i formed between the unit vectors $\boldsymbol{l}'_i$ and $\boldsymbol{l}'_{i+1}$ to the landmarks L_i and L_{i+1} to generate homing vector components along the bisecting lines of these angles. The homing vector is then calculated as

$$\boldsymbol{h} = \sum_{i=1}^{N} \boldsymbol{v}_{P_i} \boldsymbol{l}'_i \tag{3.11}$$

$$\boldsymbol{v}_{P_i} = -2 \begin{pmatrix} \cos(\frac{\beta'_1}{2}) & 0 & 0 & \dots & 0 & \cos(\frac{\beta'_N}{2}) \\ \cos(\frac{\beta'_1}{2}) & \cos(\frac{\beta'_2}{2}) & 0 & \dots & 0 & 0 \\ 0 & \cos(\frac{\beta'_2}{2}) & \cos(\frac{\beta'_3}{2}) & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \cos(\frac{\beta'_{N-1}}{2}) & \cos(\frac{\beta'_N}{2}) \end{pmatrix} \begin{pmatrix} \beta'_1 - \beta_1 \\ \beta'_2 - \beta_2 \\ \beta'_3 - \beta_3 \\ \dots \\ \beta'_N - \beta_N \end{pmatrix}$$

$$\beta_i = \operatorname{acos}(\boldsymbol{l}_i^T \boldsymbol{l}_{i+1}) \quad \text{and} \quad \beta'_i = \operatorname{acos}(\boldsymbol{l}'^T_i \boldsymbol{l}'_{i+1}) \quad \text{for} \quad i < N$$

$$\beta'_N = \operatorname{acos}(\boldsymbol{l}'^T_N \boldsymbol{l}'_1).$$

Fig. 3.13 shows the resulting streamlines and angle deviations.

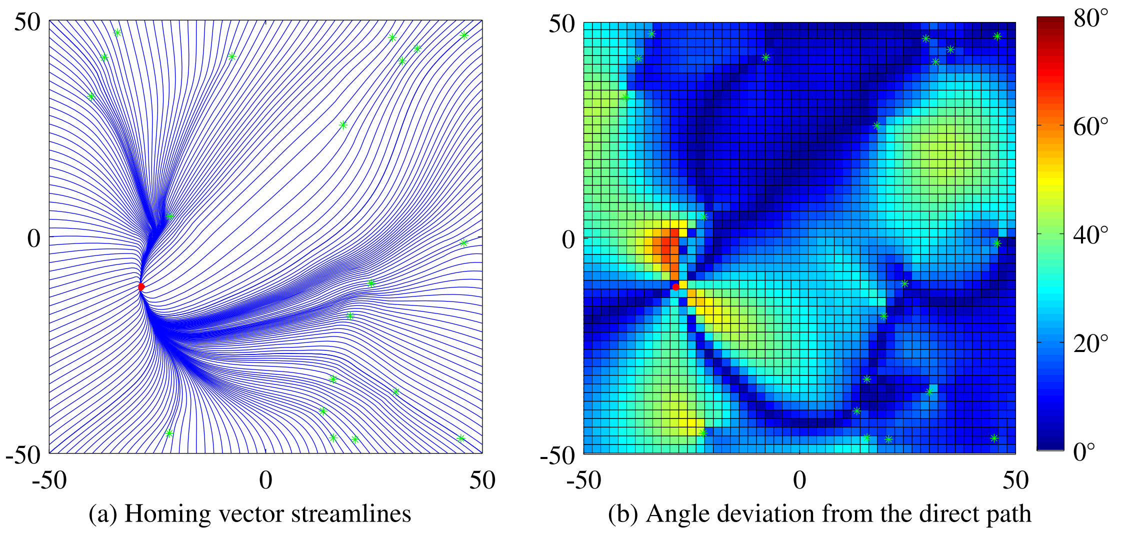


Figure 3.13.: IBVS method using bisecting components (ref. Eq. (3.11)): Homing vector streamlines and deviations from the direct path to the home location (red circle). Green asterisks: Landmarks. The x and y coordinates are given in units.

IBVS Based on Landmark Angle Differences In another work Liu et al. [60] developed a visual homing controller solely based on the scale values of the detected SIFT features. In analogy to the derivations in Liu et al. [60], below we will derive a visual homing controller based on landmark angles and prove its stability.

The bearing of a landmark is defined as $\phi_i = \operatorname{atan2}(y_i - y, x_i - x)$, where x, y are the robot coordinates in the world frame and x_i, y_i are the landmark coordinates in the world frame. The bearing is also given in the world frame, independent of the robot's current heading. Furthermore, the distance to a landmark L_i is $d_i = \sqrt{(x_i - x)^2 + (y_i - y)^2}$. We can then define the error function for visual servoing as an error vector $\boldsymbol{e} = (\boldsymbol{\phi}' - \boldsymbol{\phi})$, where $\boldsymbol{\phi}'$ is the bearing vector in the current view and $\boldsymbol{\phi}$ is the bearing vector of the goal view. We can calculate the derivative of a single element of the error function as

$$\frac{de_i}{dt} = \frac{d}{dt}\left(\phi_i'(x,y) - \phi_i\right) = \frac{d\phi_i'}{dx}\frac{dx}{dt} + \frac{d\phi_i'}{dy}\frac{dy}{dt} \tag{3.12}$$

$$= \frac{y_i - y}{d_i^2}v_x - \frac{x_i - x}{d_i^2}v_y = \frac{1}{d_i}\underbrace{\left[v_x \sin\phi_i' - v_y \cos\phi_i'\right]}_{v_{i\perp}}, \tag{3.13}$$

where $v_x = \frac{dx}{dt}$ and $v_y = \frac{dy}{dt}$.

In the 1D case the term in square brackets can be interpreted as $v_{i\perp}$, which is the projection of the current velocity on a line perpendicular to the ray from the current robot position to the landmark. Thus, the derivative of the ith element of the error vector becomes

$$\frac{de_i}{dt} = \frac{1}{d_i} v_{i\perp}. \tag{3.14}$$

To ensure an exponential decrease of the error, the velocity control law would be

$$v_{i\perp} = \lambda_i e_i, \qquad \text{with} \quad \lambda_i < 0. \tag{3.15}$$

In the 2D case, we can combine the single velocity components of the 1D controllers, so that the full control law becomes

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \sum_{i=1}^{n} \lambda_i (\phi_i' - \phi_i) \begin{bmatrix} \sin(\phi_i') \\ -\cos(\phi_i') \end{bmatrix}. \tag{3.16}$$

We can prove the convergence of this control law using Lyapunov theory as in Liu et al. [60]. A non-negative energy function can be defined as

$$E = \frac{1}{2} \sum_{i=1}^{n} (\phi_i' - \phi_i)^2. \tag{3.17}$$

The stability is proven when

$$\frac{dE(t)}{dt} < 0. \tag{3.18}$$

The derivative of the energy function is

$$\begin{aligned}
\frac{dE}{dt} &= \frac{dE}{d\phi_i'} \frac{d\phi_i'}{dt} && (3.19)\\
&= \sum_{i=1}^{n} (\phi_i' - \phi_i) \frac{d\phi_i'}{dt} && (3.20)\\
&= \sum_{i=1}^{n} (\phi_i' - \phi_i) \frac{1}{d_i} \left[v_x \sin \phi_i' - v_y \cos \phi_i' \right] && (3.21)\\
&= v_x \sum_{i=1}^{n} \frac{\phi_i' - \phi_i}{d_i} \sin \phi_i' - v_y \sum_{i=1}^{n} \frac{\phi_i' - \phi_i}{d_i} \cos \phi_i'. && (3.22)
\end{aligned}$$

By setting $\lambda = -\frac{1}{d_i}$ and using Eq. (3.16) we get

$$\frac{dE}{dt} = -(v_x^2 + v_y^2) \leq 0 \tag{3.23}$$

This term is always smaller than or equal to zero and equality applies for example when $\phi_i' = \phi_i$. Since d_i cannot be measured, the variable λ is set to a negative constant value.

For the calculation of a homing vector, we can transform the control law in Eq. (3.16):

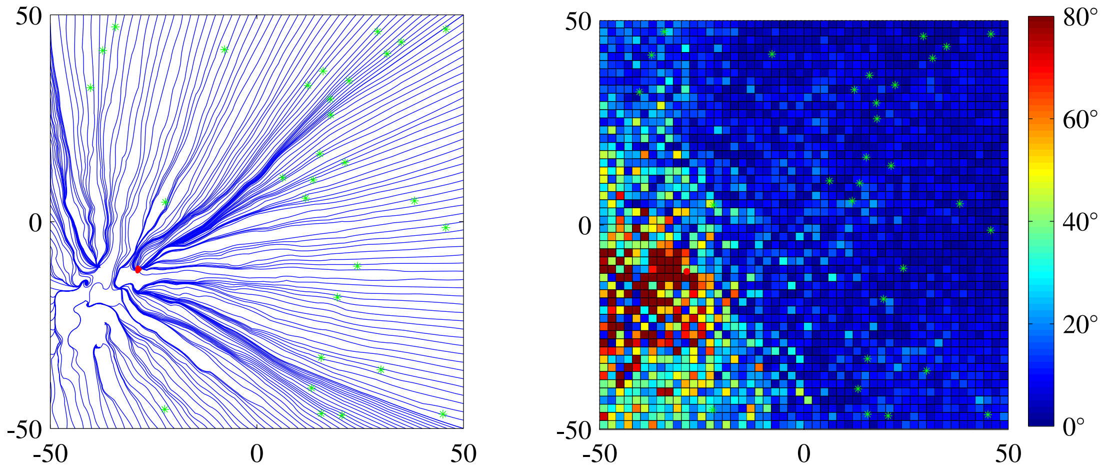
$$\begin{bmatrix} h_x \\ h_y \end{bmatrix} = \lambda \sum_{i=1}^{n} (\phi_i' - \phi_i) \begin{bmatrix} \sin(\phi_i') \\ -\cos(\phi_i') \end{bmatrix}, \tag{3.24}$$

where the angle difference $(\phi_i' - \phi_i)$ always has to be in the range $(-\pi, \pi]$. When we use the homing vector only in normalized form, the exact value of λ does not have any influence.

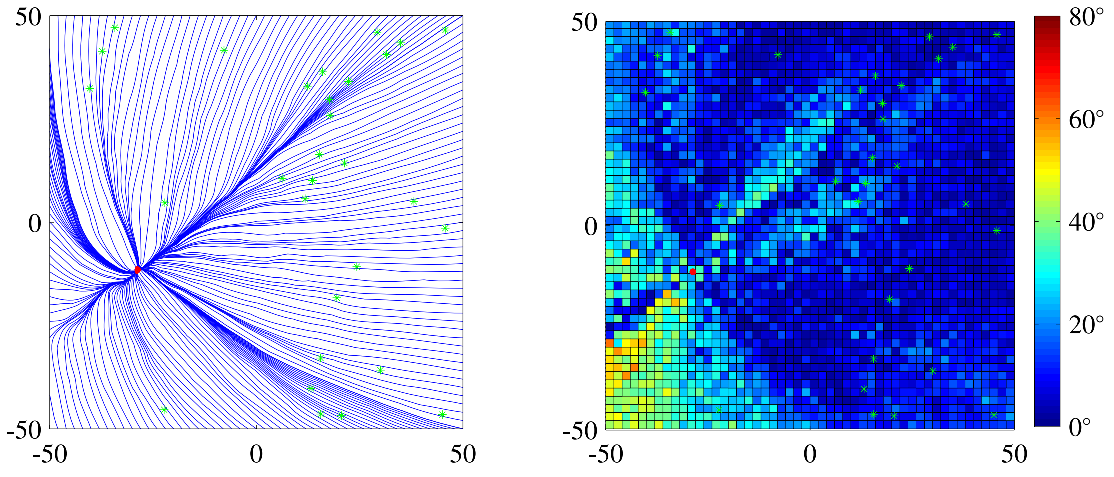
It is interesting to note that the homing vector calculation scheme derived by this visual servoing approach is exactly the same as the tangential correction vector method proposed by Weber et al. [98] given in Eq. (3.10).

Comparison of Homing Vector Calculation Methods

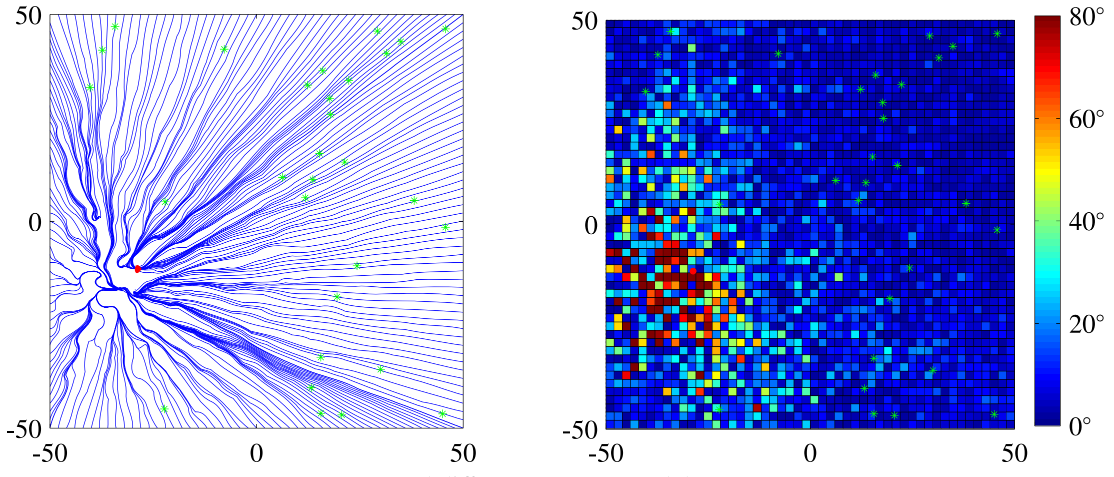
To compare the performance of different homing vector calculation methods, we defined a reference scenario consisting of an environment with a general landmark distribution that is neither isotropic nor totally clustered. We added angular measurement noise with a standard deviation of 1°, 10% landmark occlusions and 10% false landmark matches as outliers. Then, we generated homing vector streamline plots and computed the angular deviations from the direct path to the home position. We define a homing vector computation method as robust, if all streamlines from any direction reach the home location. Furthermore, the angular deviation from the direct path to the home location should be as small as possible. Table 3.1 gives an overview of the discussed homing vector calculation methods, their robustness and their angular deviations from the direct path. Fig. 3.14 and 3.15 visualize the streamlines and angle deviations. As can be seen, only the normalized difference vector model and the improved normalized difference vector model are robust according to our definition, which means that all streamlines reach the home location. Furthermore, the normalized improved difference vector method has the lowest angle deviations from the direct path to the home position. Hence, in a general environment with outliers and noise, the normalized improved difference vector method should be chosen.



(a) Difference vector model after Lambrinos et al. [54] (Eq. (3.6))

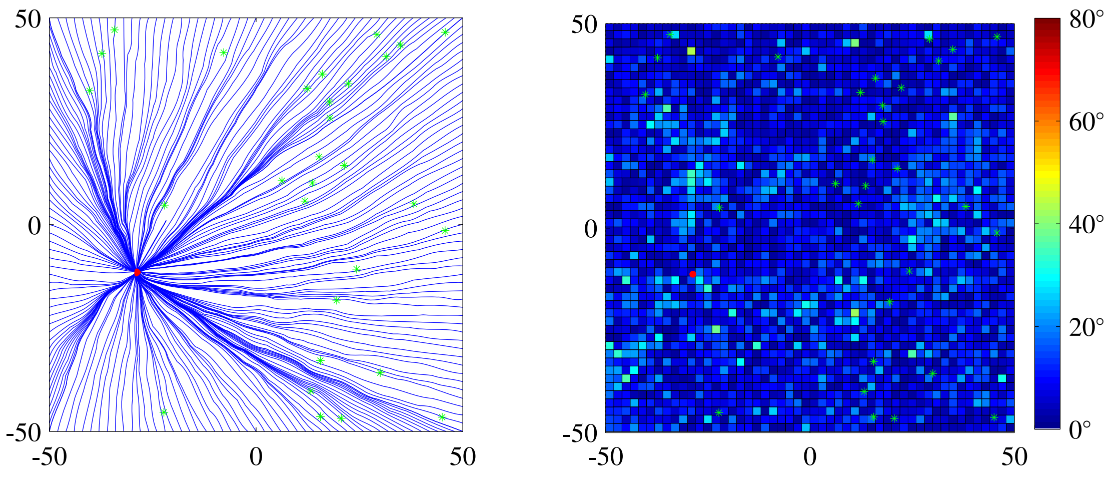


(b) Normalized difference vector model (Eq. (3.8))

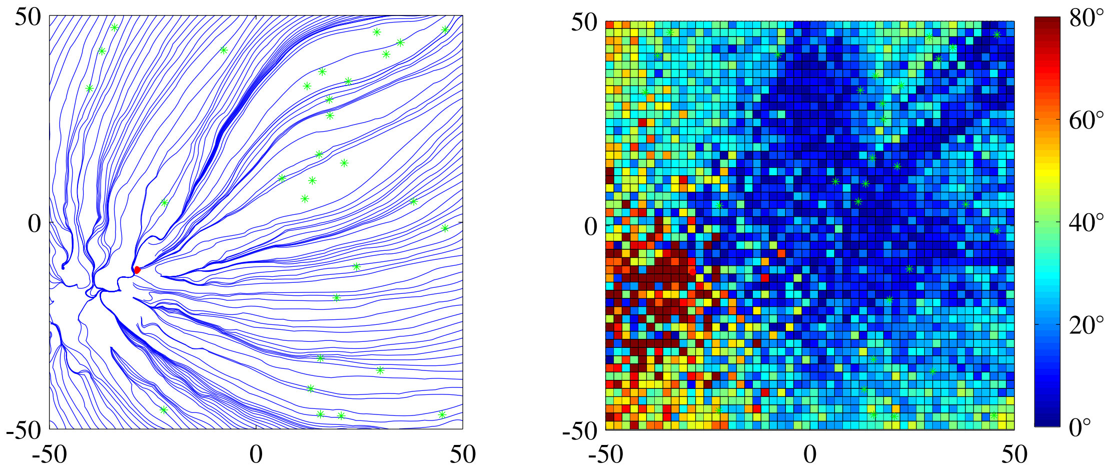


(c) Improved difference vector model (Eq. (3.7))

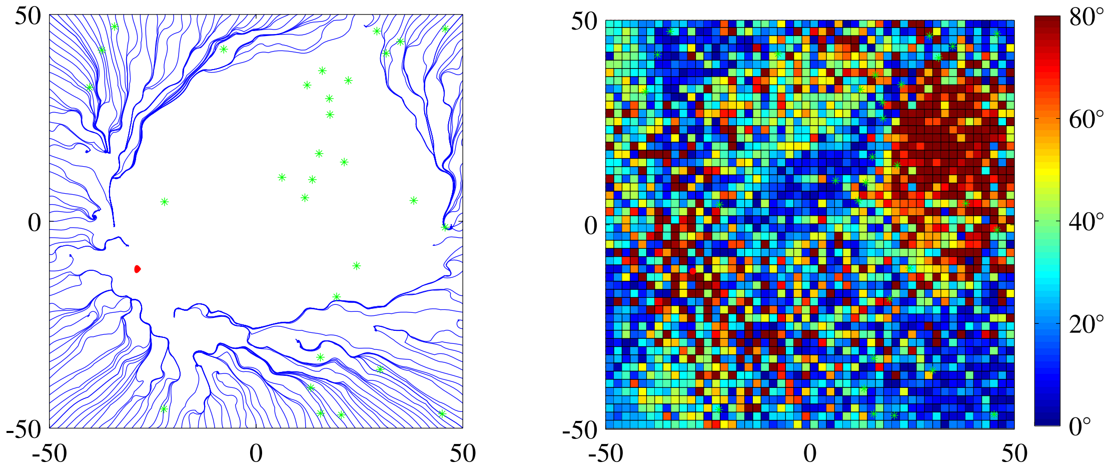
Figure 3.14.: Homing vector streamlines (left) and deviations from the direct path (right) for the reference scenario (general landmark configuration (green asterisks), noise (1.0°), outliers (10%), occlusions (10%)). Red circle: home location. The x and y coordinates are given in units.



(a) Normalized improved difference vector model (Eq. (3.9))



(b) Tangential correction vector method after Weber et al. [98] (Eq. (3.10))



(c) Image based visual servoing after Liu et al. [59] (Eq. (3.11))

Figure 3.15.: Homing vector streamlines (left) and deviations from the direct path (right) for the reference scenario (general landmark configuration (green asterisks), noise (1.0°), outliers (10%), occlusions (10%)). Red circle: home location. The x and y coordinates are given in units.

Table 3.1.: Comparison of homing vector calculation methods for reference scenario

	robust	mean angle deviation	standard deviation of angle deviation	maximum angle deviation
Difference vector model (Fig. 3.14a) [54]	-	14.1°	21.2°	176.7°
Normalized difference vector model (Fig. 3.14b)	✓	12.2°	10.5°	60.7°
Improved difference vector model (Fig. 3.14c)	-	11.1°	17.8°	178.9°
Normalized improved difference vector model (Fig. 3.15a)	✓	*7.3°*	*5.7°*	*38.6°*
Tangential correction vector method (Fig. 3.15b) [98]	-	25.9°	23.1°	177.6°
IBVS based on bisecting components (Fig. 3.15c) [59]	-	36.4°	31.5°	180.0°

Except for the tangential correction vector method [98], all of the described methods for calculating homing vectors work in 2D, as well as in 3D. Flying robots can make use of three-dimensional homing vectors, but wheeled or legged robots are restricted to the ground, so that they can only change the yaw component of their orientation actively. That means, the z-component of the homing vector or the homing elevation angle can be ignored, and only the azimuth component of the homing angles or the x- and y-components of the homing vector are considered. However, in environments with large height changes, the elevation angle can help to ensure that the robot is on the correct path, which for example leads uphill.

3.4. The Landmark-Tree Map

A translational motion of the robot results in large bearing changes of close landmarks, whereas more distant landmarks hardly change their bearings. A straightforward solution for building a map from the recorded viewframes would store each viewframe as a node of a topological map. However, that means that landmark views (LVs) of distant landmarks, which do not change over several viewframes, are stored redundantly. Furthermore, in case of memory shortage, there is no easy way to downscale the map.

Aiming at a scalable topological representation of the environment, Augustine et al. [2] introduced the LT-Map (Landmark-Tree Map) for navigation based on bearing-only landmark observations. The main idea behind the LT-Map is to build up a tree structure instead of storing a list of viewframes containing all LVs. In that tree, LVs which do not change significantly between consecutive viewframe positions are shared between these viewframes. As a result, the LVs are sorted by their degree of translation invariance. LVs which do not change over many viewframes are on a higher level in the tree and correspond to translation invariant objects. LVs which change significantly from viewframe to viewframe are in the leaves and lower nodes of the tree. These LVs correspond to close objects, which are translation variant. Viewframes can be retrieved from the tree by following the path from a leaf to the root node and collecting the LVs of all visited nodes.

The LT-Map data structure is claimed to be memory efficient because LVs that stay constant over several viewframes are only stored once. When the angle towards a landmark changes more than a given threshold δ_{ang}, a new LV is stored with the new bearing angle, the same landmark ID and the new landmark descriptor, which allows stable matching. Furthermore, the LT-Map data structure is scalable. When the robot runs out of memory, the leaves of the tree can be pruned. This means that local information is discarded but the translation invariant, global information is preserved. Hence, the robot will not follow the original path exactly, but it will still be able to find its way. Only the probability of the robot to get lost increases with pruning.

Fig. 3.16 shows an example of a landmark tree created by the original LT-Map algorithm from the simple 2D scenario shown in Fig. 3.17. The angle threshold for creating a new LV was set to $\delta_{\text{ang}} = 1°$. When viewing the tree in Fig. 3.16 in more detail, we can see that some LVs are split to different nodes although their bearing angle difference is smaller than the assumed threshold of 1°. For example, the nodes 6 and 7 both contain LVs to L_2 with

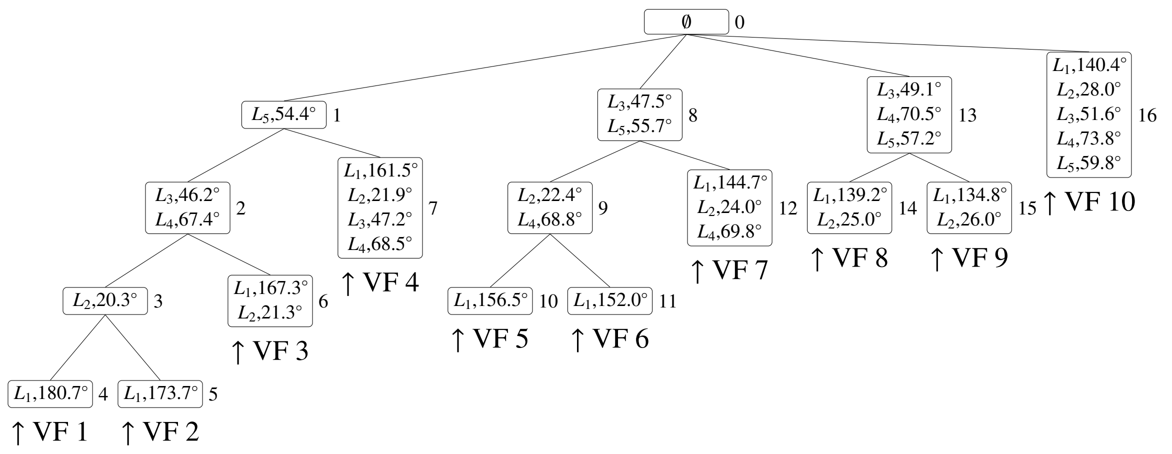


Figure 3.16.: Example of an LT-Map: The nodes contain the landmarks L_i with their angle when inserting them into the tree. The nodes are numbered for referencing. VF: viewframe

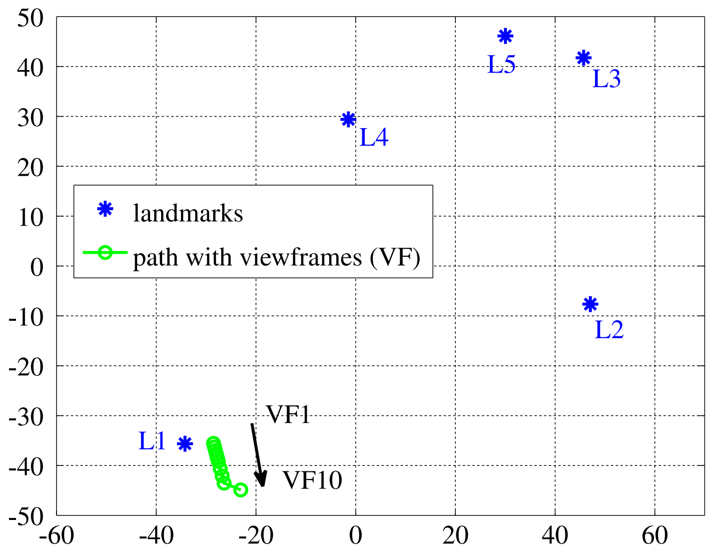


Figure 3.17.: Scenario with ten viewframes $VF1$ to $VF10$ and five landmarks L_1 to L_5

angles of 21.3° and 21.9°, respectively. Furthermore, the LVs to L_3 are split between nodes 7 and 8, and the LVs to L_4 are split between nodes 7 and 9, although their angle differences are smaller than the threshold. Hence, the intuition that landmarks whose bearings change slowly (translation invariant) always appear in the upper levels of the tree and landmarks with quickly changing bearings (translation variant) are located in the lower levels, is not always correct. This behavior is caused by the structure of the tree. A LV is only shared by more than one viewframe if the viewframes also share all higher-level LVs. Since the angles of the LVs to L_3 and L_4 change from viewframe 3 to 4, the lower LV to L_2 is not considered.

That means, it is not even checked whether the bearing of L_2 has changed at all. Hence, the height of the LVs in the tree does not give reliable information on the level of translation invariance. For this reason, the pruning operation is likely to preserve information of volatile landmarks while discarding information of stable, translation invariant landmarks.

Hence, a tree does not seem to be the optimal data structure for the viewframe map concept, because it does not reflect the structure of an ideal viewframe map. A tree always represents a hierarchy of information and does not permit overlaps between neighboring branches. For the viewframe map, such overlaps are required. For this reason, a data structure different from a tree is better suited for representing the different levels of translation invariance.

3.5. The Trail-Map

To come up with an alternative data structure for the viewframe map problem, let us have a look at Fig. 3.18, which shows a path through a simplified environment with three landmarks. When we assume an angle threshold of $\delta_{\text{ang}} = 45°$ for mapping, each landmark spans eight sectors visualized by the colored lines[1]. Within each sector, the corresponding landmark is perceived with a bearing difference of at most 45°. A new viewframe is acquired when the angle of at least one landmark changes more than δ_{ang}. Hence, every time the robot leaves a landmark sector, it acquires a new viewframe. In this environment, a viewframe region is defined by the intersection of three specific landmark sectors. The first three viewframe regions of the robot path are highlighted in Fig. 3.18. When crossing a sector border, the robot also observes a new LV. The LVs of the three different landmarks are visualized by the colored path segments.

When traversing the path, the robot observes several LVs, but only one LV changes with the acquisition of each new viewframe. Hence, the robot should rather store the observed LVs instead of the single viewframes which redundantly contain many LVs. This was also the key idea behind the LT-Map. Fig. 3.19 illustrates the observed LVs and the corresponding viewframe numbers.

To make the viewframe map scalable in terms of memory, we want to introduce a hierarchy of how translation invariant each LV is. For this, the LVs are ordered by the number of

[1]For the sake of simplicity and without loss of generality we set the angle intervals to full 45° intervals in this example instead of letting the angle intervals start individually at the bearing angle at which the landmark is perceived first.

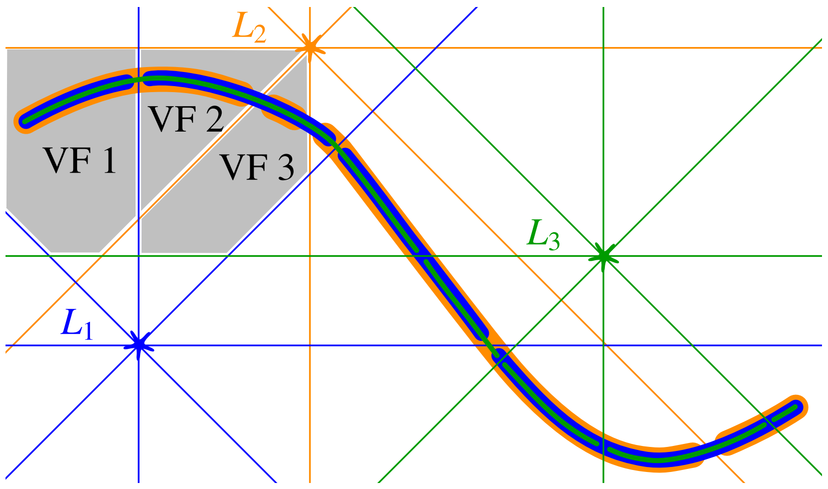


Figure 3.18.: Intuitive illustration of viewframe mapping. VF 1-3: highlighted first three viewframes. L_i: Landmarks with sectors

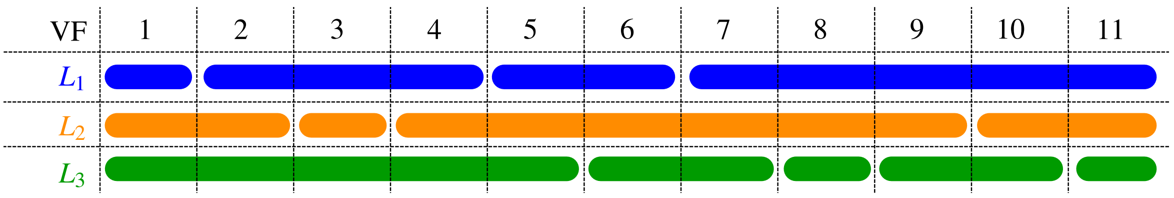


Figure 3.19.: From the LVs to the Trail-Map representation (step 1)

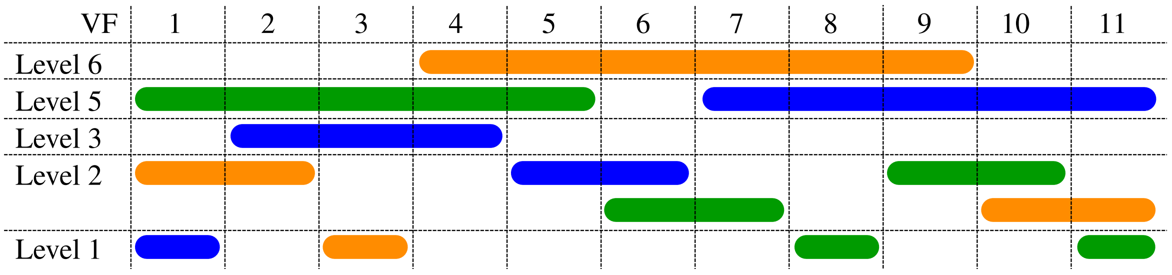


Figure 3.20.: From the LVs to the Trail-Map representation (step 2)

viewframes they span. LVs that span many viewframes have a higher level of translation invariance than LVs that span only a few viewframes. LVs which span the same number of viewframes should be on the same level in the hierarchy. Fig. 3.20 shows the hierarchically ordered LVs. The level number gives the number of viewframes that each of the contained LVs spans. Level 4 does not exist since no LV spans exactly 4 viewframes. The resulting structure is the key idea of the Trail-Map, which stands for *Tra*nslation *I*nvariance *L*evel *Map*. Now, when the robot runs out of memory, the LVs in the lower levels can be deleted without discarding stable and important long-term information.

Similar to the density of the viewframes, the degree of translation invariance and, hence, the pruning operation, also adapt to the local landmark distribution. For regions with only far landmarks, LVs in a specific level might span a much longer metric distance than LVs of the same level in regions with close landmarks. As a result, after pruning there will still be more viewframes in regions with a high landmark density than in regions with only distant landmarks.

To represent the structure of the translation invariance levels without memory overhead, we chose to use a combination of linked lists. On the top level, there is a list of translation invariance levels, which is ordered by the level number. Each level contains a linked list of LVs in the order they were observed by the robot. A landmark view in this list consists of the landmark's descriptor, its ID, its bearing as unit vector, and the number of the viewframe when it was added to the list. The viewframe number is required for the extraction of single viewframes from the Trail-Map. Fig. 3.21 illustrates the implementation of the Trail-Map data structure.

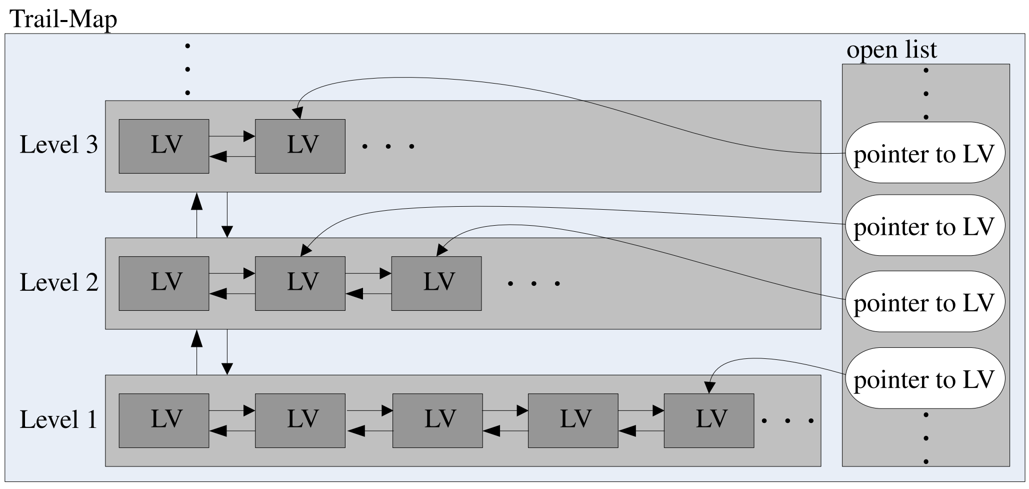


Figure 3.21.: Implementation of the Trail-Map. LV: Landmark view

Algorithm 1 gives pseudo-code for creating this list of landmark view lists. In the beginning, the Trail-Map contains only one level: level 1. Furthermore, there is an *open list* which always contains references to the landmarks that the robot observed in the previous step. The open list is also empty initially. When the robot observes the first viewframe, all LVs to the landmarks in the viewframe are added to level 1 and references to all these LVs are stored in the open list (lines 19-21). When the robot observes the next viewframe, level 2 is created (lines 1-2) and all LVs of the open list are compared to the features in the current viewframe (lines 3-5). If a landmark in the open list is also present in the current viewframe,

Algorithm 1: Appending a viewframe to the Trail-Map

```
Input  : viewframe $VF$; $trailMap$; $openList$
Output: $trailMap$; $openList$
1  if !$trailMap$.GetHighestLevel.IsEmpty then
2      $trailMap$.addLevel;
3  foreach $l$ in $openList$ do
4      foreach $f$ in $VF$.LandmarkViews do
5          if $f$.descriptor matches $l$.descriptor then
6              if $|f$.angle-$l$.angle$| < \delta_{\text{ang}}$ then
7                  $trailMap$[$l$.getLevel].remove($l$);
8                  $trailMap$[$l$.getLevel+1].append($l$);
9                  $openList$.updatePointerTo($l$);
10             else
11                 $openList$.removePointerTo($l$);
12                 $trailMap$[1].append($f$);
13                 $openList$.appendPointerTo($f$);
14             $VF$.remove($f$);
15             found:=true;
16             break;
17     if !found then
18         $openList$.removePointerTo($l$);
19 foreach $f$ in $VF$.LandmarkViews do
20     $trailMap$[1].append($f$);
21     $openList$.appendPointerTo($f$);
```

and if its angle is within the angle threshold, then the LV is removed from its current level and appended to the end of the next higher level (lines 7-8). The reference to that LV in the open list is updated (line 9). If the angle of the landmark has changed more than the specified threshold, then the new LV is added to level 1, the reference to the old LV in the open list is deleted and the reference to the new LV is added at the end of the open list (lines 11-13). If a landmark in the open list cannot be found in the current viewframe, then its reference is deleted from the open list (line 18). All landmarks of the current viewframe that were observed for the first time are added to level 1 and their references are appended to the open list (lines 19-21). By repeating this procedure for each newly acquired viewframe, we achieve the following properties for the Trail-Map:

1. The level each LV is stored in corresponds to the number of viewframes for which the LV did not change its angle by more than the specified threshold δ_{ang}.
2. The references in the open list are always ordered by the level number of the corresponding landmark view.
3. The LVs in the single levels are ordered by the viewframe number they were first observed at.

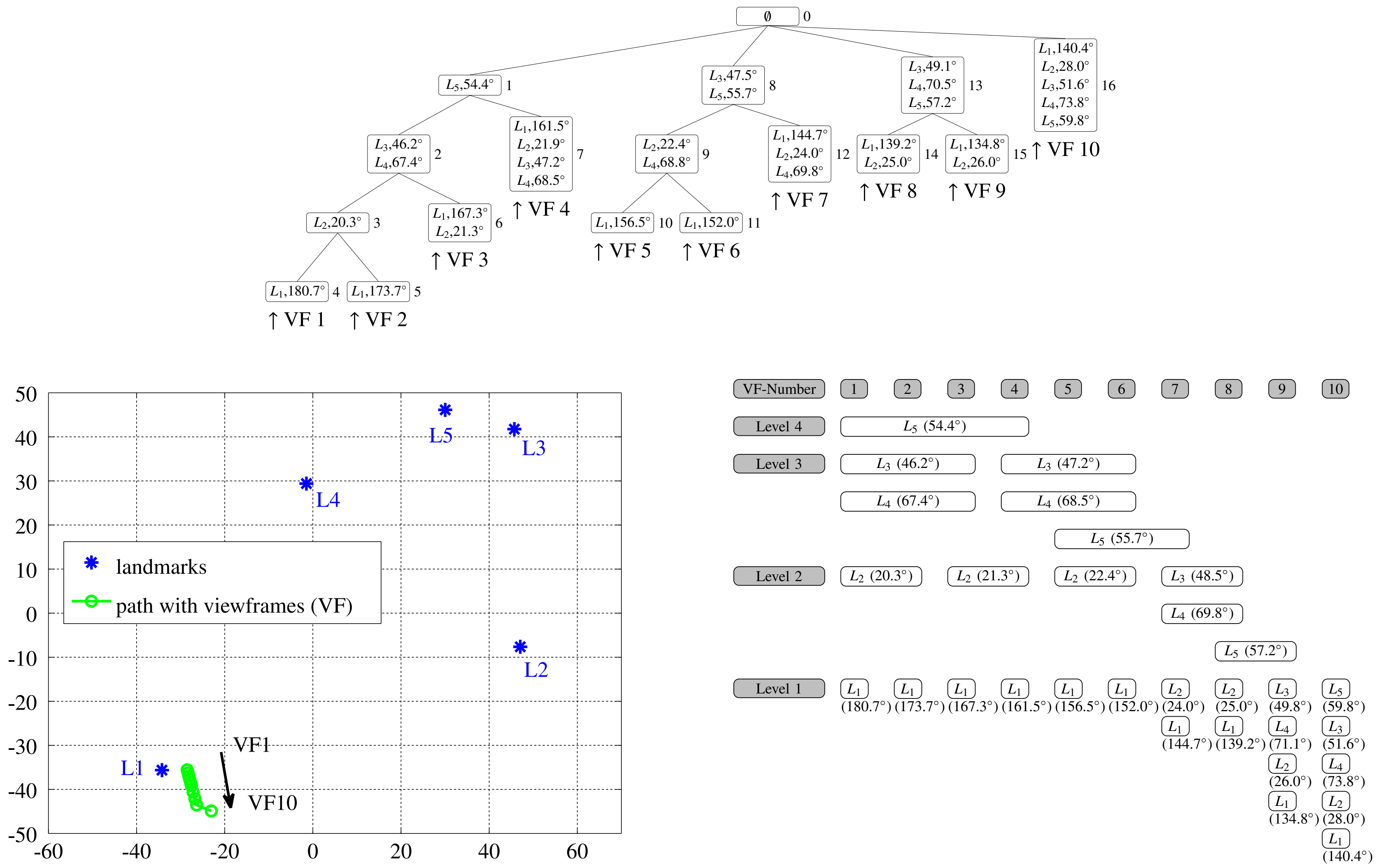
Compared to the LT-Map, the Trail-Map has some advantages:

- *Intuitive:* The resulting structure of the Trail-Map corresponds to the expected hierarchy of the LVs.
- *Non-redundant:* A landmark view is stored only once as long as the landmark's bearing does not change significantly.
- *Efficient:* The Trail-Map operations, especially the map pruning, are faster than for the LT-Map (ref. Section 3.6.2).

As a disadvantage, the retrieval of the viewframes from the Trail-Map has become more difficult. While for the LT-Map, only the paths from the leaves to the root node have to be traversed, full lists must be searched when looking for a specific viewframe number in the Trail-Map. However, since the lists are sorted, this search can be implemented efficiently. Furthermore, for general navigation it is not required to extract specific viewframes, but in most cases the viewframes are retrieved in the same or in reverse order of their visit. This stepwise retrieval can be solved with minimal costs due to the sorted structure of the lists that emerges from the construction algorithm. That means, when having retrieved one viewframe, to get the next or the previous viewframe we only have to consider the direct neighbors of the LV in each level list and check whether the sum of the level number and the insertion frame number of that LV fits to the number of the viewframe to be retrieved. This procedure is repeated until the newly acquired viewframe contains at least one different LV than the previous one. Furthermore, viewframes with too few LVs are discarded to allow robust navigation.

When landmarks are occluded or cannot be detected in some images, translation invariant landmarks would be inserted in lower levels due to Algorithm 1. To prevent this, landmarks which are not visible in some images and reappear at the same bearing angle should be inserted such as they were observed all the time. To do this, LVs which disappear in the next viewframe, are held in the open list and marked as *waiting*. They are only deleted from the open list, when they could not be observed for more than a specified number of viewframes or when they are observed at a significantly different angle. In case they are detected again at an angle within the angle threshold, they are moved up to the level where the landmark would have been if it had been visible all the time.

The same landmark configuration that resulted in the example LT-Map in Fig. 3.16 gives the Trail-Map structure depicted in Fig. 3.22. The Trail-Map contains 31 LVs as compared to 33 in the LT-Map. Furthermore, a new LV is only inserted when the angle changes significantly, irrespective of how the other LVs in the same viewframe behave. Thus, inconsistencies as pointed out for the LT-Map do not occur. When pruning the Trail-Map in Fig. 3.22 by one level, the viewframes 1 and 2, and 5 and 6, respectively, will be regarded as one viewframe by the viewframe retrieval algorithm, because they contain the same LVs.



∅ 0
L_5,54.4° 1
L_3,46.2° L_4,67.4° 2
L_2,20.3° 3
L_1,180.7° 4 ↑ VF 1
L_1,173.7° 5 ↑ VF 2
L_1,167.3° L_2,21.3° 6 ↑ VF 3
L_1,161.5° L_2,21.9° L_3,47.2° L_4,68.5° 7 ↑ VF 4
L_3,47.5° L_5,55.7° 8
L_2,22.4° L_4,68.8° 9
L_1,156.5° 10 ↑ VF 5
L_1,152.0° 11 ↑ VF 6
L_1,144.7° L_2,24.0° L_4,69.8° 12 ↑ VF 7
L_3,49.1° L_4,70.5° L_5,57.2° 13
L_1,139.2° L_2,25.0° 14 ↑ VF 8
L_1,134.8° L_2,26.0° 15 ↑ VF 9
L_1,140.4° L_2,28.0° L_3,51.6° L_4,73.8° L_5,59.8° 16 ↑ VF 10
landmarks
path with viewframes (VF)
L1
L2
L3
L4
L5
VF1
VF10
VF-Number 1 2 3 4 5 6 7 8 9 10
Level 4: L_5 (54.4°)
Level 3: L_3 (46.2°) L_4 (67.4°); L_3 (47.2°) L_4 (68.5°); L_5 (55.7°)
Level 2: L_2 (20.3°); L_2 (21.3°); L_2 (22.4°); L_3 (48.5°) L_4 (69.8°); L_5 (57.2°)
Level 1: L_1 (180.7°); L_1 (173.7°); L_1 (167.3°); L_1 (161.5°); L_1 (156.5°); L_1 (152.0°); L_2 (24.0°) L_1 (144.7°); L_2 (25.0°) L_1 (139.2°); L_3 (49.8°) L_4 (71.1°) L_2 (26.0°) L_1 (134.8°); L_5 (59.8°) L_3 (51.6°) L_4 (73.8°) L_2 (28.0°) L_1 (140.4°)

Figure 3.22.: Comparison of LT-Map and Trail-Map. Top: LT-Map, Bottom left: Scenario, Bottom right: Trail-Map

3.6. Performance Comparison of the Trail-Map and the LT-Map

To compare the performances of the Trail-Map and the LT-Map, we created a MATLAB simulation environment consisting of uniquely identifiable landmarks spread randomly within an area of 200×200 units. Fig. 3.23 shows a path starting at $(0, 0)$ with a length of about 130 units. The simulated robot with an omnidirectional landmark sensor was led along this path and recorded viewframes. We chose the kth maximum angle dissimilarity measure $\delta_{\text{diss}}^{\text{max}}$ (ref. Section 3.3.1) with $k = 1$ for learning and for retracing the path, since no noise and outliers were simulated. The dissimilarity threshold for creating a new viewframe was set to $\xi_{\delta_{\text{diss}}^{\text{max}}}^{\text{map}} = 5°$. That means, the robot recorded a new viewframe every time one landmark changed its bearing by more than 5° compared to the previously recorded viewframe. We also set the dissimilarity threshold for homing to $\xi_{\delta_{\text{diss}}^{\text{max}}}^{\text{hom}} = 5°$. The angle thresholds for creating a new LV in the LT-Map and the Trail-Map were $\delta_{\text{ang}} = 5°$.

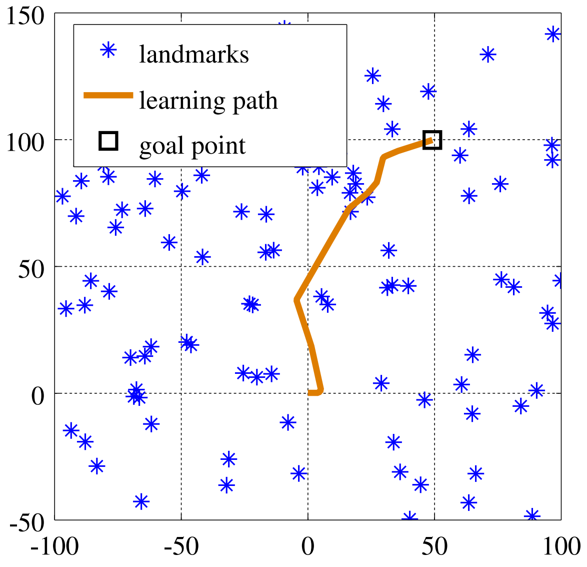


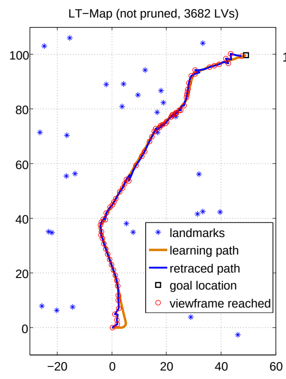
Figure 3.23.: Simulation environment with learning path. The robot should retrace the path to reach the goal location.

In this simulation, we placed the robot back to the starting position after learning the path and let the robot compute homing vectors to retrace the path in the same direction as during learning. Since the robot has a perfect omnidirectional sensor, the direction of retracing the learned path has no influence on the performance. For calculating the homing vector, we used the difference vector model in Eq. (3.6) as described in Section 3.3.2, because we did not simulate any noise and had an isotropic landmark distribution. However, we adapted the length of the navigation vector to the local landmark configurations. When the angle of a landmark between two navigation steps changed less than δ_{ang}, the navigation vector length was increased, and when a landmark bearing changed by more than $2\delta_{\text{ang}}$ between two navigation steps, the navigation vector length was decreased. Thus, in regions with nearby landmarks, the single navigation steps were smaller than in regions with landmarks that were far away. We did not simulate any noise, outliers, or occlusions, because the basic behavior of the LT-Map and the Trail-Map should be compared. The simulated robot should navigate along the recorded path using the LT-Map and the Trail-Map at different pruning levels. To ensure that the robot could always reach the goal position with a good accuracy, the last viewframe acquired at the goal point was never pruned.

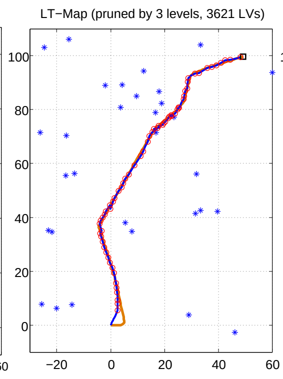
3.6.1. Memory and Navigation Performance

To evaluate the memory and navigation performance, we randomly spread 100 uniquely identifiable landmarks within the area of 200×200 units. We ran the simulation 100 times with different isotropic landmark configurations for each pruning level and randomly changed the landmark configuration after each run. We computed the path error as the area between the learning trajectory and the retraced path.

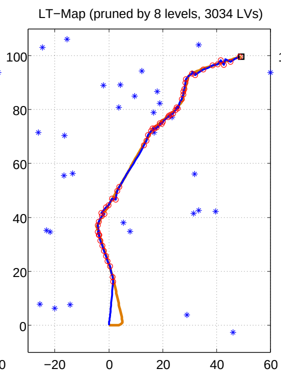
Fig. 3.24 shows the retraced paths for different pruning levels for one example environment. The resulting Trail-Map contained 1364 LVs spread over 38 levels in this case, the lowest being 1 and the highest being 50. The tree of the LT-Map contained 3682 LVs and had a maximal depth of 26. Fig. 3.24a and Fig. 3.25a show the retraced paths using the full LT-Map and the full Trail-Map, respectively. The accuracy of the followed trajectories was approximately similar in this example. However, when the LT-Map was pruned to the number of landmarks of the full Trail-Map, the retraced path already degraded significantly (see Fig. 3.24e). As Fig. 3.24f shows, with 600 LVs the resulting path of the LT-Map had only very few viewframe locations left, hence, the accuracy decreased noticeably. In contrast, when the Trail-Map was pruned to the same proportion of its original size (see Fig. 3.25d), viewframes were still spread over the whole path. Even with as few as 153 LVs (which is 11% of the original number), the path was still followed well.



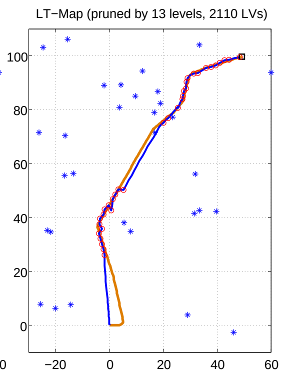
(a) 100% LVs, $\epsilon = 54.7$



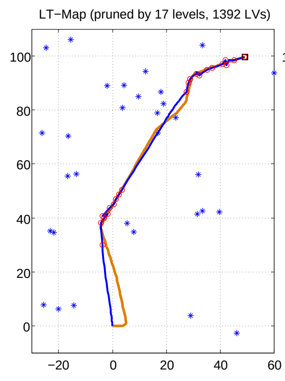
(b) 98% LVs, $\epsilon = 52.9$



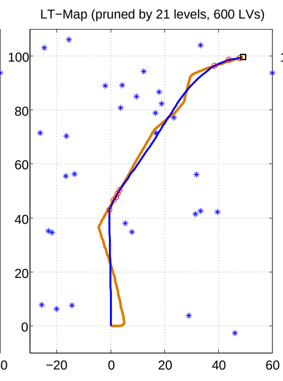
(c) 82% LVs, $\epsilon = 72.7$



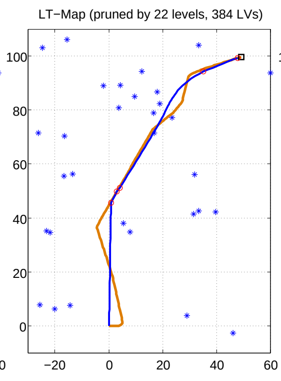
(d) 57% LVs, $\epsilon = 156.6$



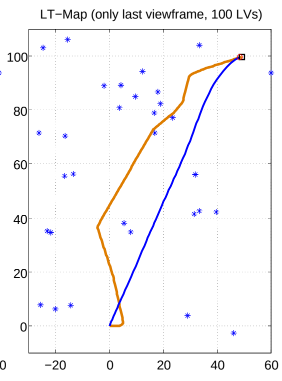
(e) 38% LVs, $\epsilon = 174.8$



(f) 16% LVs, $\epsilon = 175.2$

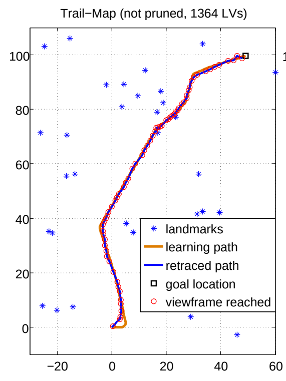


(g) 10% LVs, $\epsilon = 195.9$

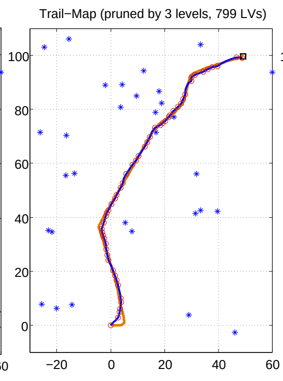


(h) 3% LVs, $\epsilon = 1204.2$

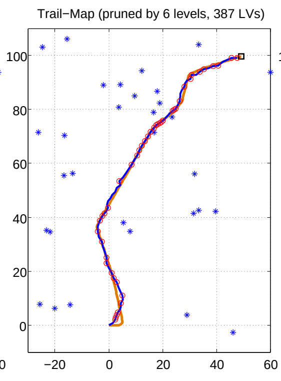
Figure 3.24.: LT-Map pruning behavior (ϵ: path error in units^2)



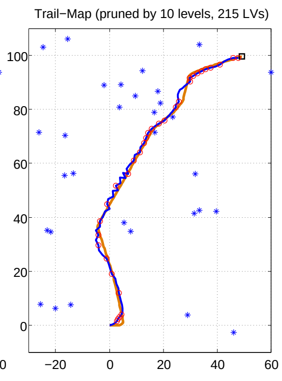
(a) 100% LVs, $\epsilon = 55.5$



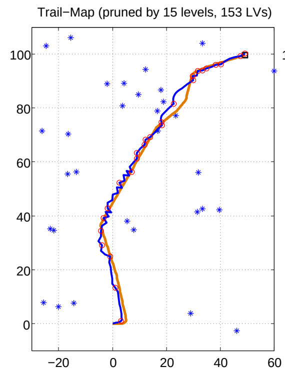
(b) 59% LVs, $\epsilon = 56.0$



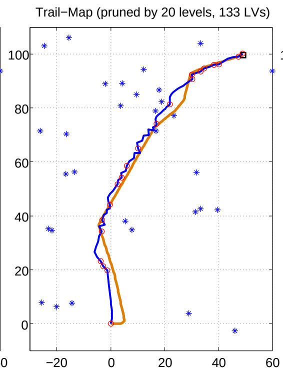
(c) 28% LVs, $\epsilon = 61.9$



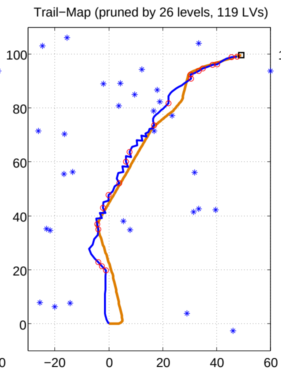
(d) 16% LVs, $\epsilon = 99.5$



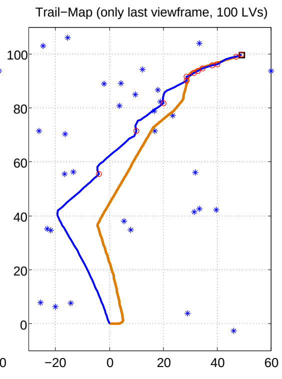
(e) 11% LVs, $\epsilon = 142.7$



(f) 10% LVs, $\epsilon = 236.9$



(g) 9% LVs, $\epsilon = 262.5$



(h) 7% LVs, $\epsilon = 699.9$

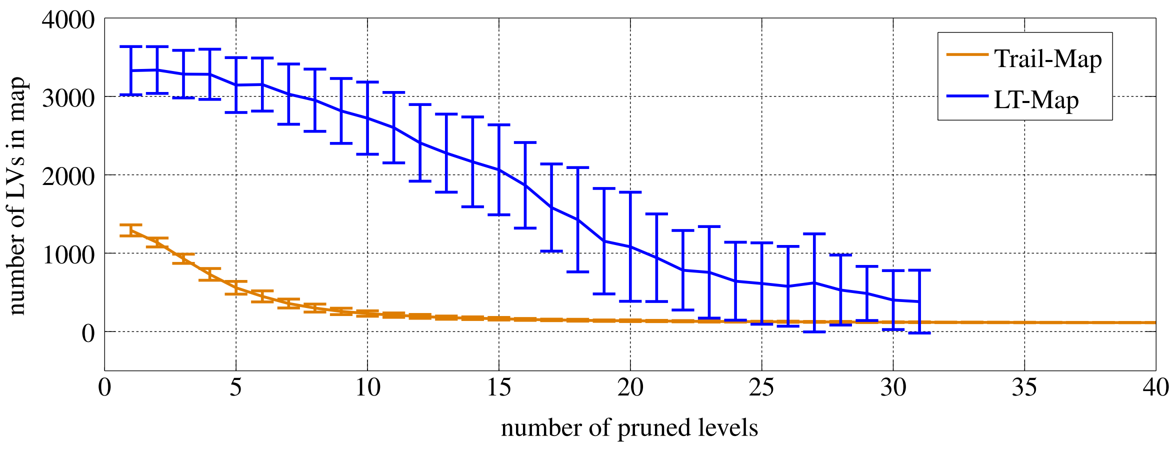
Figure 3.25.: Trail-Map pruning behavior (ϵ: path error in units^2)

When comparing the followed trajectories of the LT-Map and the Trail-Map using only the goal viewframe, Fig. 3.24h and Fig. 3.25h show that the trajectory using the LT-Map was a nearly direct path to the goal location, while the Trail-Map path still went through intermediate viewframes. The reason for this is that in the Trail-Map the last viewframe still contains the information about the levels of each LV. LVs of the higher levels produce intermediate viewframes which are reached first. Later, LVs in the lower levels create refined viewframes that finally lead to the goal position. This kind of information is lost when pruning the LT-Map tree.

Fig. 3.26 visualizes the statistics of the simulation results. One of the most noticeable aspects is that the full LT-Map contained 3315 LVs on average while the Trail-Map required 1278 LVs, which is only 38.5% of the size of the LT-Map. This shows that the Trail-Map data structure is much more memory efficient because LVs are not stored redundantly. When pruning the Trail-Map, the number of LVs reduces quickly. In contrast, when pruning the LT-Map, in the beginning only a few deep branches are affected. This shows that the LT-Map is not as well-balanced as the Trail-Map.

As shown in Fig. 3.26, the path error resulting from the full LT-Map was slightly smaller than the path error of the full Trail-Map. The reason for that is the already mentioned fact that the LT-Map often inserts LVs again with their current angle, because higher-level LVs have changed, although the lower LVs have not exceeded the threshold δ_{ang} yet. Hence, in the LT-Map the landmark angle resolution is on average higher than δ_{ang}, which results in a more accurate navigation path.

What should be noted for interpreting the data in Fig. 3.26, is the fact that for the 100 simulated landmark configurations, not all resulting LT-Maps had the same depth. That means, after pruning 20 levels or more, only a fraction of the created LT-Maps could be considered in the calculation of the mean, because many LT-Maps had no LVs left. The same applied to the resulting Trail-Maps, because not all Trail-Maps had more than 30 levels. Furthermore, the standard deviations of the number of LVs in the LT-Map grow very big when pruning many levels, because the LT-Map contains the most LVs towards the root of the tree. The standard deviations of the number of LVs in the Trail-Map show opposite behavior because in the Trail-Map most LVs are located in the lower levels. We are aware of the fact that the distributions of the path error and the number of LVs are not always Gaussian, but we use the standard deviations in the error bar plots to give an impression of how much the different values vary.



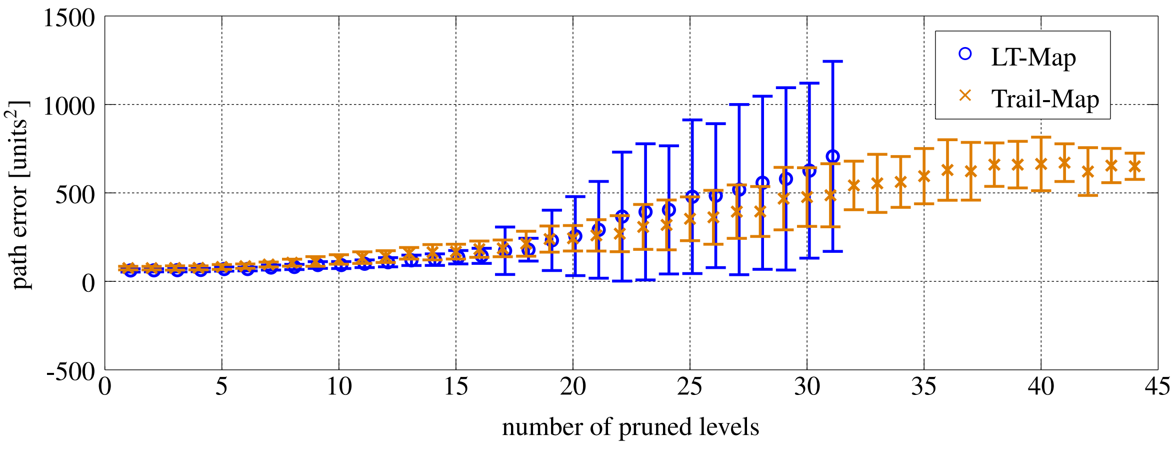


Figure 3.26.: Statistical comparison of retracing a path using the LT-Map and the Trail-Map depending on the number of pruned levels (means and standard deviations)

The most remarkable plot is Fig. 3.27, which illustrates the average behavior of the path error depending on the percentage of remaining LVs. The navigation performance of the LT-Map degraded quickly. The path error of the LT-Map exceeded the minimum path error of the Trail-Map after pruning about 10% of the LVs. In contrast, the Trail-Map could be pruned to about 50% of the original number of LVs without significant loss of path accuracy. This advance in performance of the Trail-Map is remarkable, keeping in mind that the 100% mark corresponds to 3315 LVs for the LT-Map, but only to 1278 LVs for the Trail-Map. Comparing the number of required LVs to achieve a path error of about $\epsilon = 77$, the LT-Map required 2938 LVs, while the Trail-Map only needed 546 LVs on average. That means a memory saving of more than 80%. Considering path errors of about $\epsilon = 160$, the Trail-Map saved about 90% of LVs compared to the LT-Map.

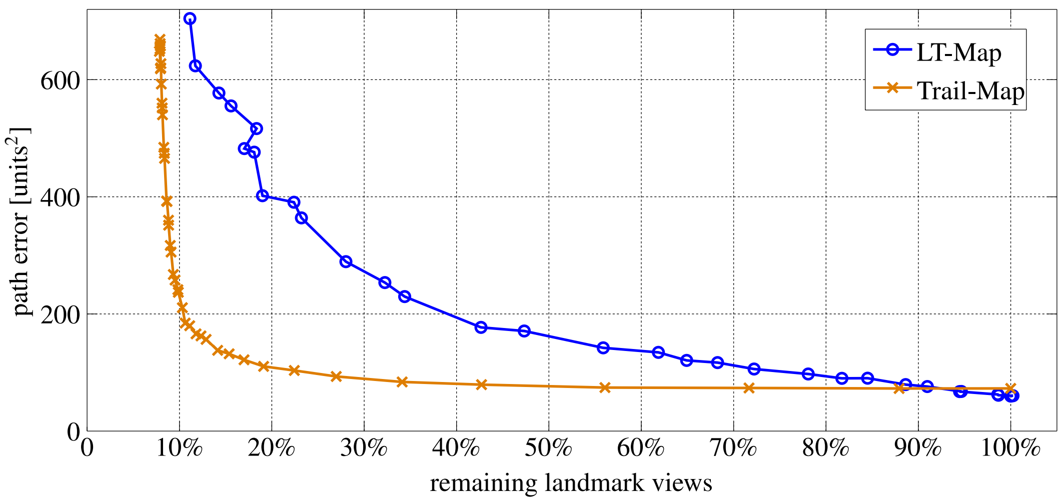
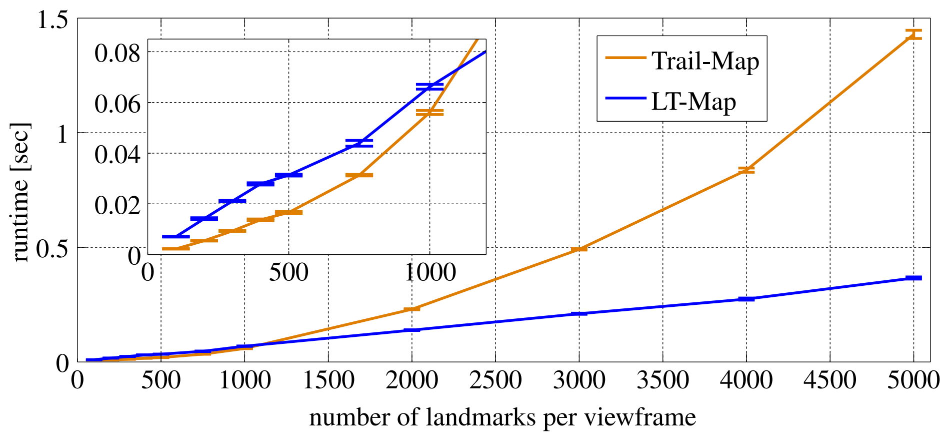


Figure 3.27.: Comparison of path errors for retracing a path with the LT-Map and the Trail-Map for different pruning ratios

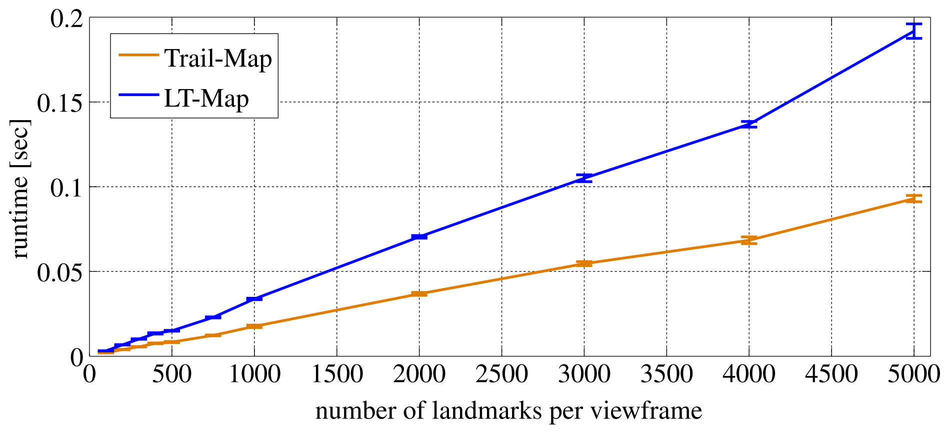
3.6.2. Runtime Performance

To show that the Trail-Map does not only save memory but is also computationally efficient, we compared the runtimes of our Trail-Map implementation and the original implementation of the LT-Map. Both maps were implemented in C++ and compiled with the same compiler and flags. Runtimes are measured on a 2.67 GHz CPU. We used the simulation environment and path described in the previous section and created environments with 100 up to 5000 randomly distributed landmarks spread within the range of 200 × 200 units. The recorded viewframes along the learning path were imported into the C++ programs to create the different maps. We computed the runtimes of creating a full map from all recorded viewframes, of pruning the map and of retrieving the complete list of viewframes from the maps in the same order of their first traversal.

According to Algorithm 1, a viewframe is added to the Trail-Map in O(N^2) with N being the number of LVs per viewframe. The quadratic behavior can be seen in Fig. 3.28a. The worst case complexity of the LT-Map creation is also O(N^2), but since the landmark comparison with the previous viewframe can abort in the LT-Map as soon as a LV changes significantly, the best case complexity of creating the LT-Map is O(N). Since N stays constant during the mapping phase, the runtime of the map creation process does not increase with the map size. As Fig. 3.28a shows, the Trail-Map can be created faster than the LT-Map when the viewframes do not contain more than about 1000 LVs, which is sufficient for most applications.



(a) Map creation time



(b) Map retrieval time

Figure 3.28.: Runtime comparison of the LT-Map and the Trail-Map

The retrieval of the viewframes from the Trail-Map is faster than from the LT-Map as shown in Fig. 3.28b. A possible reason could be the fact that, although the number of LVs per viewframe is the same, the LT-Map contains more LVs than the Trail-Map due to redundancies.

Additionally, the Trail-Map can be pruned magnitudes faster than the LT-Map, as shown in Fig. 3.29. Especially when pruning only one level, the LT-Map needs to traverse the full tree to find the leaves which have to be pruned, while only one level list has to be deleted from the Trail-Map. For 5000 LVs per viewframe, pruning takes up to 350 milliseconds for the LT-Map, but less than 1 millisecond for the Trail-Map. The runtime for pruning the Trail-Map increases with the number of pruned levels, because more LVs have to be deleted. In contrast, the runtime for pruning the LT-Map decreases with the number of pruned levels, because a smaller part of the tree has to be traversed to find the branches to be trimmed.

For 1000 LVs per viewframe, the LT-Map can be pruned by one level in 30 milliseconds, the Trail-Map in 70 nanoseconds.

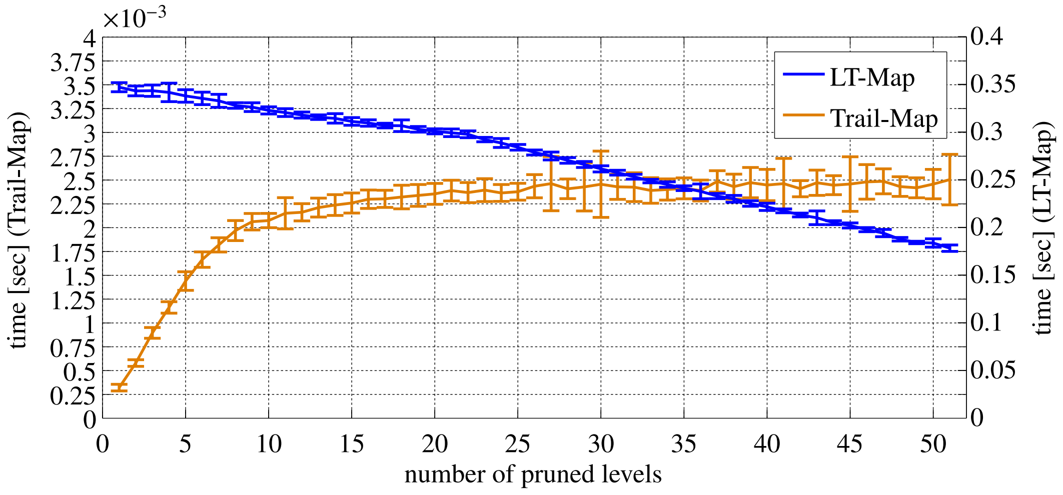


Figure 3.29.: Runtimes for pruning the LT-Map and the Trail-Map (for 5000 LVs per viewframe). Please note, that the curves for the LT-Map and the Trail-Map have different scales.

To conclude the comparison of the LT-Map and the novel data structure Trail-Map, it was shown that the Trail-Map outperforms the LT-Map in terms of navigation performance, memory requirements and runtime, because the level structure of the Trail-Map is more appropriate for representing a map consisting of viewframes than the tree-structure of the LT-Map.

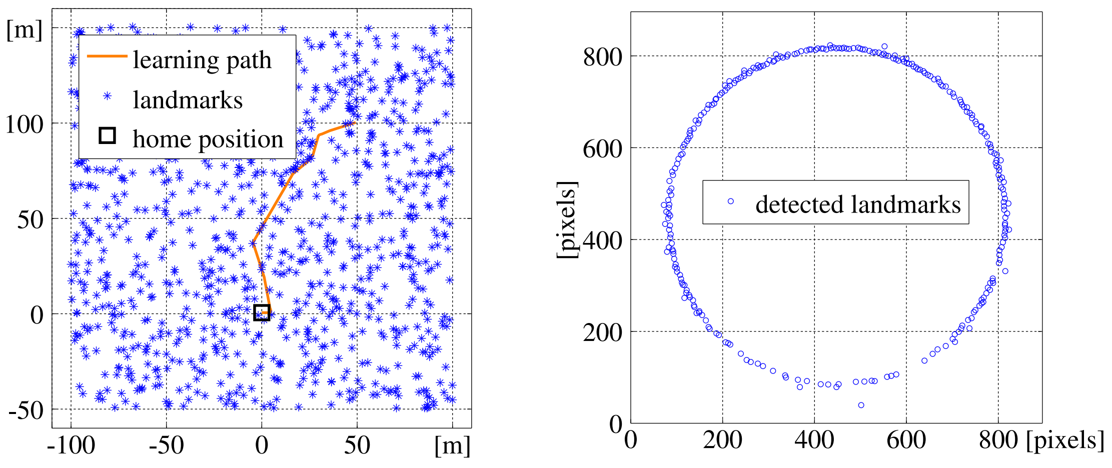
3.7. Trail-Map-Based Homing under the Presence of Sensor Noise

To evaluate the homing performance using the Trail-Map under realistic conditions, we conducted further simulations considering sensor noise.

For these simulations we used the EKF-SLAM Toolbox for Matlab [84]. Although the main purpose of this toolbox is the simulation of SLAM[2], it incorporates different robot and sensor models that we adjusted to viewframe-based homing. We made use of the omnidirectional camera model, simulating a catadioptric camera with a resolution of 896×896 pix-

[2]A comparison of the Trail-Map based homing method and SLAM is shown in Section 3.8

els, mounted at a height of 0.5m on the simulated robot. We randomly distributed 1000 landmarks within an area of $200 \times 200\text{m}^2$, with heights ranging from 0m to 1m, simulating a bounded desert-like open area. The simulated landmarks are projected into the camera frame so that a camera image is simulated. To this image, pixel noise can be applied. Furthermore, we added functionality to simulate natural occlusions of landmarks. If more than one landmark was within a pixel window of $d_{\text{win}} \times d_{\text{win}}$ in the image, only the landmark closest to the robot had been visible. For all simulations we chose a value of $d_{\text{win}} = 5$, resulting in 285 of the 1000 landmarks being visible on average. Each landmark had an ID and we assumed known data association. A learning path was defined by a sequence of steering commands and following these directions for a step length of 0.75m each. The resulting path had a length of about 130m. The setup is shown in Fig. 3.30. Since the robot could only move on the plane, we only considered translational odometry errors in x and y directions and rotational errors around the z-axis (yaw), simulating a 2D motion.



(a) Landmarks and learning path (b) Camera image at home position

Figure 3.30.: Simulation setup

In the simulations, we again used the kth maximum angle dissimilarity measure $\delta^{\text{max}}_{\text{diss}}$ (ref. Section 3.3.1) and set k variable to 1% of the landmark matches between subsequent viewframes. The dissimilarity threshold for recording a new viewframe was set to $\xi^{\text{map}}_{\delta^{\text{max}}_{\text{diss}}} = 5°$. That means, a new viewframe was recorded every time at least 1% of all matching landmarks changed their bearings by more than $\xi^{\text{map}}_{\delta^{\text{max}}_{\text{diss}}} = 5°$ compared to the previously recorded viewframe. The angle threshold for creating the Trail-Map was again set to $\delta_{\text{ang}} = 5°$. The homing vectors were computed using the improved difference vector model as given in Eq. (3.7). The homing dissimilarity threshold was also set to $\xi^{\text{hom}}_{\delta^{\text{max}}_{\text{diss}}} = 5°$. To account for landmark occlusions, the buffer size for landmarks that were visible in one viewframe but not in the following one was set to 5 viewframes.

To avoid unnecessary turning motions, the robot assumed to have passed a viewframe when the calculated homing vector changed by more than 90° [43]. Then, the robot discarded the computed homing vector and switched to the next viewframe.

In case the landmarks of the goal viewframe are not visible from the current position of the robot, no homing vector can be computed. Thus, we implemented a searching behavior similar to the searching patterns performed by desert ants [70]. For this, the robot started driving in a spiral until it could observe enough landmarks of the goal viewframe to compute a homing vector.

To ensure that important points can be reached with a high accuracy regardless of the pruning level of the Trail-Map, the viewframes of such points should be stored separately, e.g. on hard disk. In our simulations, we stored the exact viewframe of the home position and used this viewframe for the final homing steps. In addition, we decreased the dissimilarity measure to $\xi^{\text{hom}}_{\delta^{\text{max}}_{\text{diss}}} = 1^\circ$ for approaching the final viewframe.

To ensure that the learning paths were identical for all runs, the robot executed the odometry commands without any errors during the learning phase and only the robot's prediction of the performed motion was disturbed by odometry noise. During the homing phase, the robot's actual motion was disturbed by the specified odometry errors, but its motion prediction was identical to the commanded odometry.

During the learning phase, the robot created a Trail-Map by using the dissimilarity measure to decide when to record a new viewframe. Then, homing was performed by driving in the calculated homing direction for a step size of 0.5m until the similarity condition for reaching the next viewframe was met or the viewframe was passed. If no homing vector could be computed, the robot started a spiral search motion.

For evaluating the homing performance, we introduced different criteria:

- **Path error**: The path error is the area between the learning path and the homing path. The smaller the path error is, the more accurate the learning path is followed. Thus, the robot minimizes the risk of driving into unknown and possibly untraversable areas.
- **Endpoint error**: The endpoint error is the distance between the robot's final position after homing and the original home position where the robot started from.

- **Homing path length**: The shorter the homing path is, the less energy the robot has to consume.
- **LVs in Trail-Map**: The number of landmark views in the Trail-Map indicates how much memory is required for the map.

In the perfect noise-free case with $\delta_{\text{ang}} = 5°$ and $\xi^{\text{map}}_{\delta^{\text{max}}_{\text{diss}}} = 5°$, the robot created a Trail-Map with 106 viewframes and 7867 landmark views and performed homing in 129m with a path error of 23.6m^2 and an endpoint error of 0.69m. When the Trail-Map was pruned by 5 levels, the path error increased by 23% to 29.1m^2, the endpoint error was 0.34m and the homing path length decreased to 128m. However, after pruning the Trail-Map had 2033 landmark views, which is only 26% of the original number. Fig. 3.31 shows that both homing trajectories are very similar.

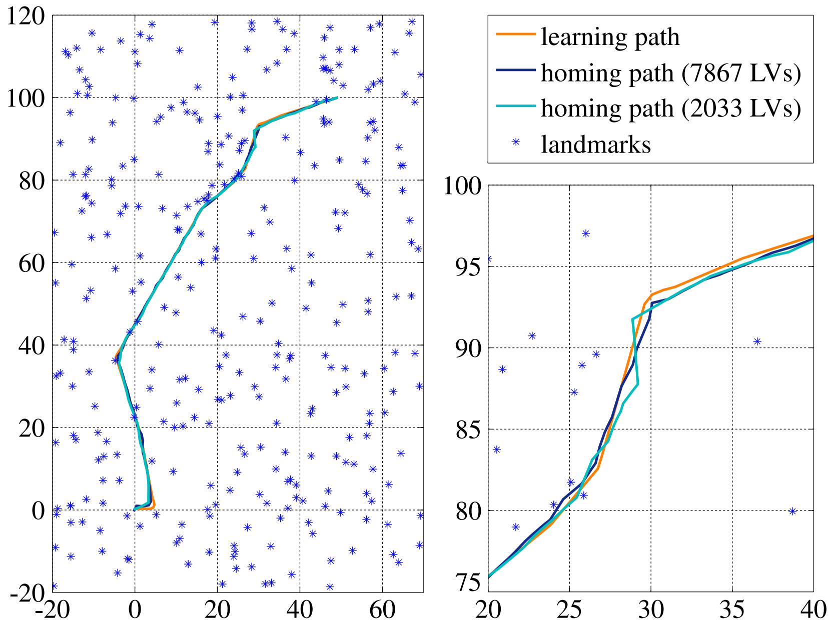


Figure 3.31.: Homing paths without noise, not pruned and pruned by 5 levels

The influence of different noise sources is shown in the following sections. First, we will analyze the influence of single noise sources using the full Trail-Maps. In the end, we will investigate the homing performance with a combination of all error sources and pruning of the Trail-Map. For each noise scenario, we compute the mean values and standard deviations based on 20 simulation runs with different random seeds.

3.7.1. Pose Estimation Errors

The robot usually has only access to an estimate of its pose originating from its odometry sensors or other position and orientation sensors, possibly fused using a Kalman Filter framework. Pose estimation errors are caused by noisy sensors, such as odometers, inertial measurement units or a compass. Slip can also lead to erroneous pose estimates.

Translational Error

We simulated the learning and the homing phases with translational odometry errors equal in x and y direction with standard deviations between $\sigma_{\text{trans}} = 0.01$m and $\sigma_{\text{trans}} = 0.5$m while all other error sources were set to 0. As shown in Fig. 3.32, the translational error has only little influence on the accuracy and the length of the resulting homing path. Only for very large odometry errors of $\sigma_{\text{trans}} = 0.5$m, which is 100% of the homing step size, the path error and the path length increase significantly. The endpoint error stays approximately constant about 0.5m for all simulated odometry errors. Since no translational odometry information is used to create the Trail-Map, the influence of the translational odometry error is only caused by the inaccuracy of following the computed homing vector during the homing phase.

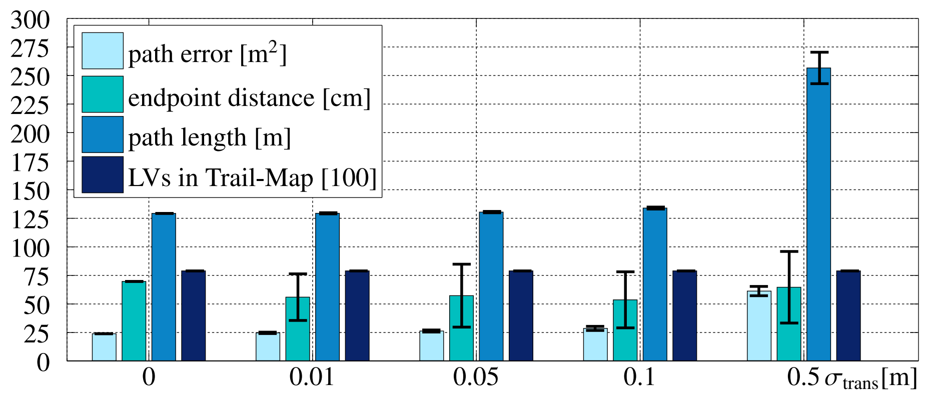


Figure 3.32.: Influence of a translational odometry error in x and y direction on the Trail-Map-based homing performance

Accumulating Rotational Errors

We simulated rotational odometry errors of the yaw angle with standard deviations between $\sigma_{\text{rot,acc}} = 0.05°$ and $\sigma_{\text{rot,acc}} = 1°$. Since the viewframes in the Trail-Map are assumed to be perfectly rotationally aligned, errors in the robot's orientation estimate have a significant influence on the Trail-Map itself and on the resulting homing path. As Fig. 3.33 shows, the path error starts to increase for rotational errors of $\sigma_{\text{rot,acc}} \geq 0.5°$, as do the path length and the endpoint error. The path length increases since the calculated homing vectors are erroneous so that the robot has to correct its heading frequently. As the figure also shows, the standard deviations of all measures increase strongly for $\sigma_{\text{rot,acc}} \geq 0.5°$, because due to the accumulating behavior of the orientation error, the final orientation estimate varies in a huge range. The Trail-Map size stays approximately constant for $\sigma_{\text{rot,acc}} \leq 0.5°$ but also increases to 8200 LVs for $\sigma_{\text{rot,acc}} = 1°$ because the threshold $\delta_{\text{ang}} = 5°$ for constructing the Trail-Map is exceeded more quickly when the rotational alignment of the viewframes is not correct. For longer paths the result will be worse since odometric rotational errors accumulate.

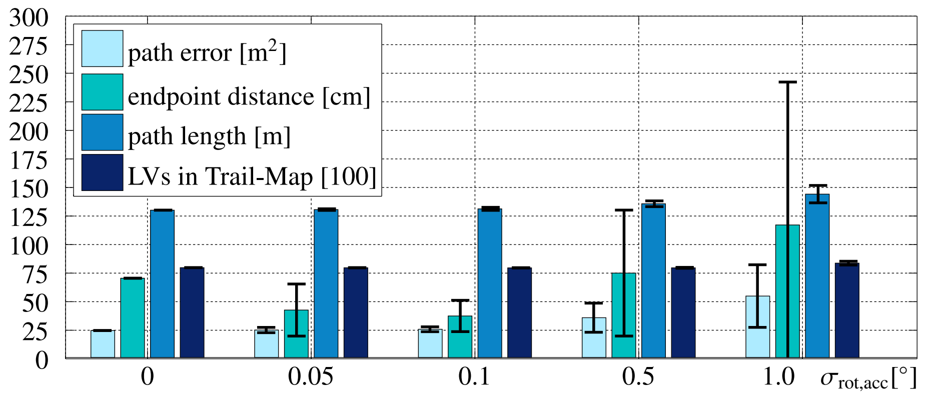


Figure 3.33.: Influence of accumulating rotational errors of the yaw angle on the Trail-Map-based homing performance

Absolute Rotational Errors

In case the robot has a compass to measure its orientation, the robot can estimate its orientation without any drift, but the estimated orientation will still be noisy. We simulated absolute yaw errors with standard deviations between $\sigma_{\text{rot,abs}} = 0.5°$ and $\sigma_{\text{rot,abs}} = 5°$. Fig. 3.34 shows the influence of the yaw errors on the homing accuracy and the map size. The path error and the path length increase for $\sigma_{\text{rot,abs}} > 2°$, while the map size already grows for $\sigma_{\text{rot,abs}} > 1°$. The endpoint distance stays constant for all simulated values of $\sigma_{\text{rot,abs}}$. The map size grows extremely, first because many more viewframes are recorded during the learning phase since the dissimilarity between two viewframes is increased by the rotational error. Second, the angle threshold $\delta_{\text{ang}} = 5°$ is exceeded for nearly every landmark view in a

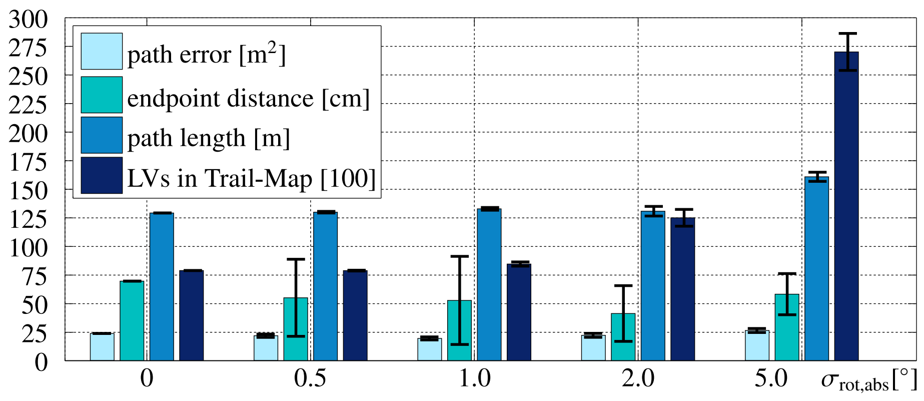


Figure 3.34.: Influence of an absolute rotational error of the yaw angle on the Trail-Map-based homing performance

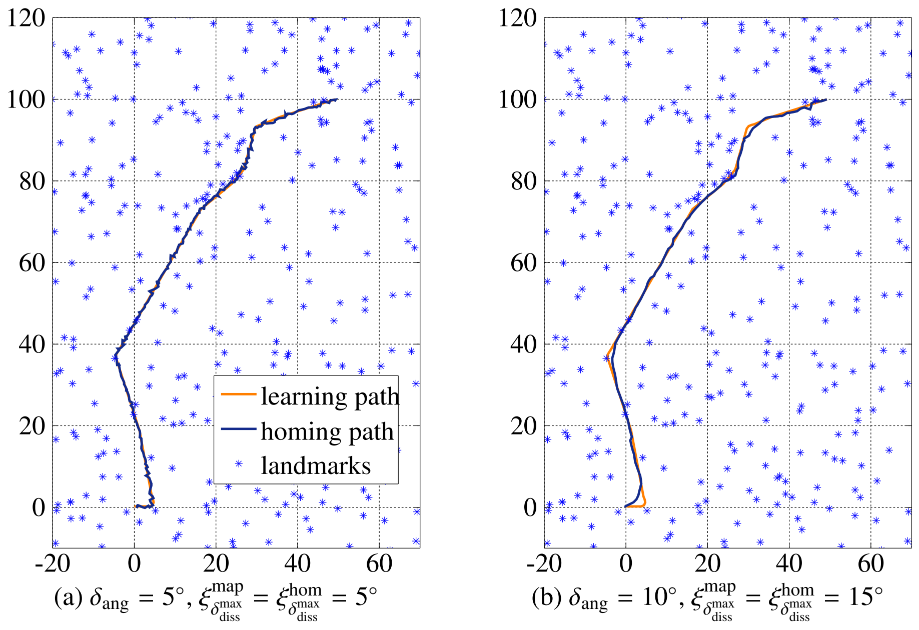


Figure 3.35.: Homing for a large absolute rotational error of $\sigma_{\text{rot,abs}} = 5°$

new viewframe. Thus, the resulting Trail-Map has only very few levels and most landmark views are in the lowest level, which calls for adjusting the parameters δ_{ang}, $\xi^{\text{map}}_{\delta^{\text{max}}_{\text{diss}}}$ and $\xi^{\text{hom}}_{\delta^{\text{max}}_{\text{diss}}}$. An error of $\sigma_{\text{rot,abs}} = 1°$ is tolerated for the given angle threshold of $\delta_{\text{ang}} = 5°$ and the dissimilarity threshold of $\xi^{\text{map}}_{\delta^{\text{max}}_{\text{diss}}} = \xi^{\text{hom}}_{\delta^{\text{max}}_{\text{diss}}} = 5°$. For larger absolute rotation errors of $\sigma_{\text{rot,abs}} = 5°$, choosing $\delta_{\text{ang}} = 10°$ and $\xi^{\text{map}}_{\delta^{\text{max}}_{\text{diss}}} = \xi^{\text{hom}}_{\delta^{\text{max}}_{\text{diss}}} = 15°$ gives satisfying results (ref. Fig. 3.35). The resulting Trail-Map has 7775 landmark views, the homing path length is 129m and the path error is 40.9m^2.

3.7.2. Observation Errors

Observation errors are image noise, occlusions of landmarks and landmark outliers. Image noise is caused by the pixel matrix of the sensor. Apart from the natural occlusions described previously, sometimes landmarks cannot be detected by the image feature detector, although they are visible. We call this kind of error random occlusions. Landmark outliers can be caused by false positives in the landmark matching process and are simulated as measurements of known landmarks at random orientations.

Image Noise

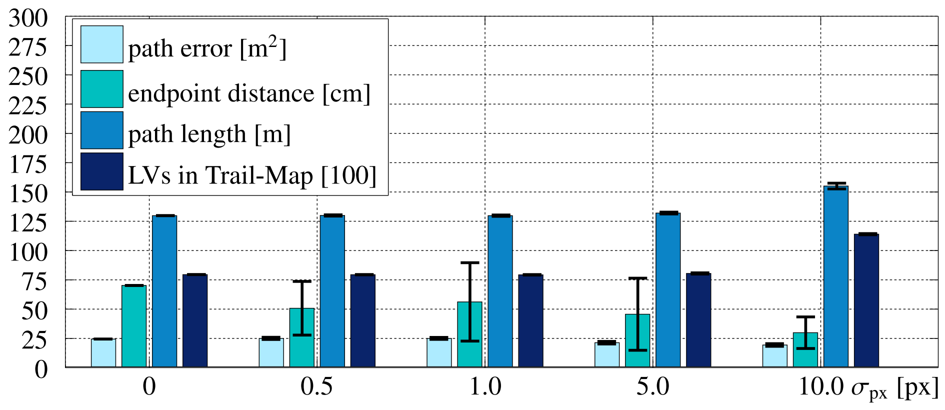
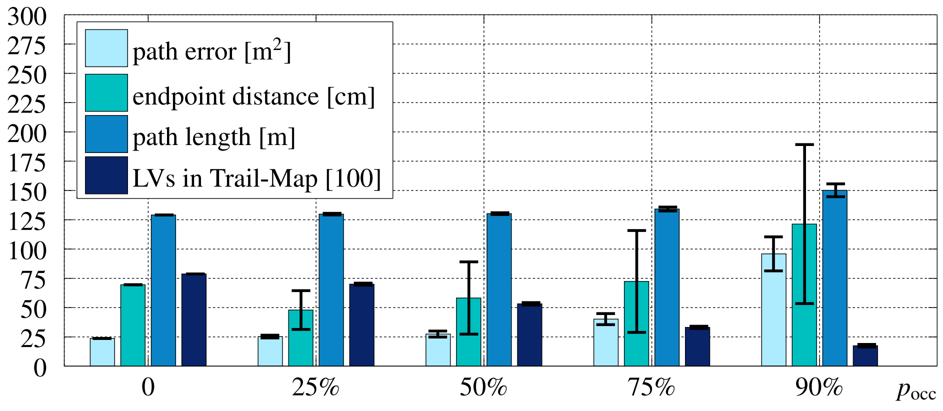


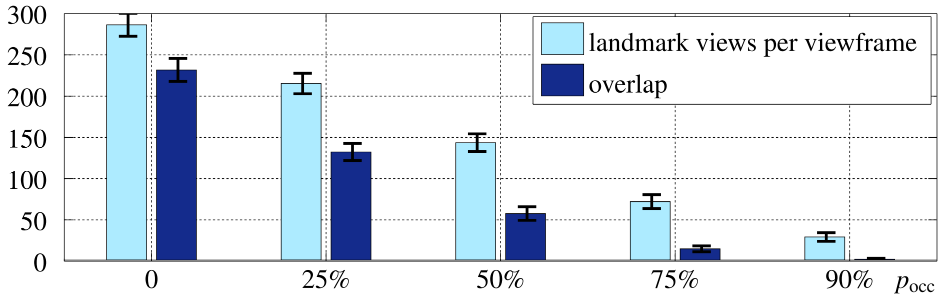
Figure 3.36.: Influence of pixel noise on the Trail-Map-based homing performance

The image noise changes the position of a landmark in the simulated camera image and, thus, has an influence on the accuracy of measuring the landmark angles. Fig. 3.36 shows the influence of different pixel standard deviations ranging from $\sigma_{\text{px}} = 0.5\text{px}$ to $\sigma_{\text{px}} = 10\text{px}$ on the homing performance. In general, the effect of pixel noise is low. The path length and the size of the Trail-Map increase for large standard deviations of $\sigma_{\text{px}} > 5\text{px}$. The path error decreases for higher σ_{px}, because many more viewframes are created during the learning phase due to the high noise in measuring the landmark angles. Thus, the path can be followed more accurately. The endpoint error also decreases, because due to the high noise in measuring the landmark angles, a higher ratio of actual landmark angle differences has to be below the dissimilarity threshold of $\xi_{\delta_{\text{diss}}^{\max}} = 5°$, before the home viewframe is assumed to be reached.

Random Occlusions



(a) Random landmark occlusions

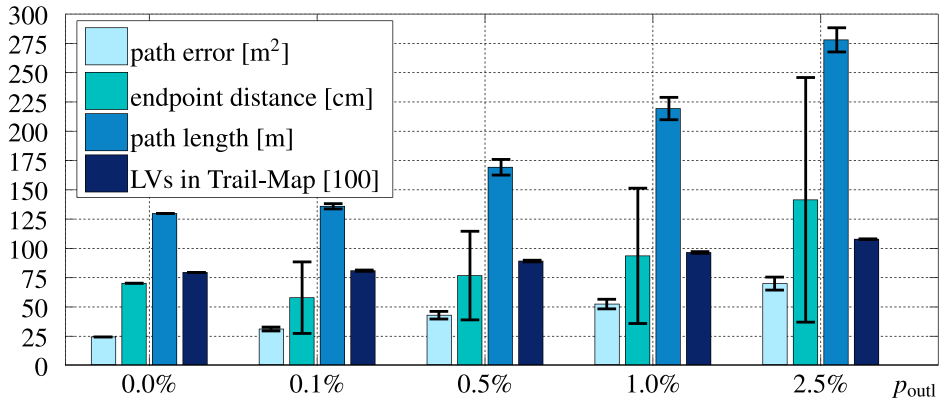


(b) Detected landmarks and overlap for different occlusion probabilities

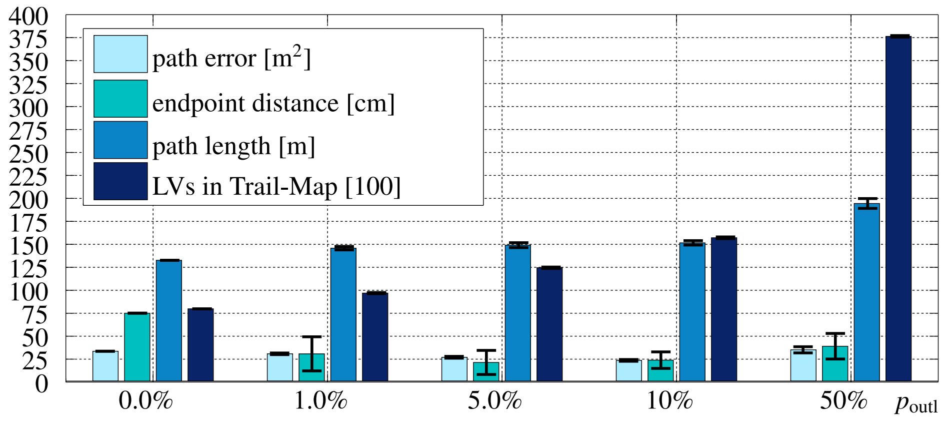
Figure 3.37.: Influence of random landmark occlusions on the Trail-Map-based homing performance

We simulated random occlusions by randomly deleting a percentage p_{occ} of all visible (not naturally occluded) landmarks in each simulated image. Fig. 3.37b shows how the number of detected landmarks per image and the number of common landmarks between successive viewframes decrease. As Fig. 3.37a shows, up to an occlusion probability of 50% the homing performance does not degrade visibly. For $p_{\text{occ}} \geq 0.75$ the homing path length, the endpoint error and the path error increase. However, even for $p_{\text{occ}} = 0.9$, where only 28 landmarks are detected per image and the overlap is only 1 landmark, homing is still successful. Only for $p_{\text{occ}} > 0.9$ we observed frequent failures in reaching the home position. The size of the Trail-Map decreases since less landmarks are detected. Here, the importance of the buffer during the Trail-Map creation becomes obvious: Without buffering unobservable landmark views, the Trail-Map size would increase by nearly 50% to 10340 landmark views on average for an occlusion probability of $p_{\text{occ}} = 0.25$.

Outliers



(a) Improved difference vector model



(b) Normalized improved difference vector model

Figure 3.38.: Influence of outliers on the Trail-Map-based homing performance for different homing vector calculation methods

Similar to the occlusions, we simulated the outliers by randomly assigning an orientation to a certain percentage p_{outl} of all visible landmarks. As Fig. 3.38a shows, the homing behavior using the improved difference vector model for calculating the homing vectors is very sensitive to outliers. An outlier probability of only 0.5% already leads to significantly larger path lengths and path errors, because the robot meanders around the learned path. Also the number of elements in the Trail-Map increases because the outlier landmarks are in most cases added as new entries in level 1, due to their large angle difference. Since the method is robust against occlusions but very sensitive to outliers, a strict outlier rejection method should be implemented, which rather rejects correct matches than allowing for false positives. Since this is hard to achieve in practice, another way to decrease the effect of false landmark matches would be the use of a homing vector calculation method which is more robust against outliers. As shown in Section 3.3.2, the normalized improved difference vector model is a good choice. Using this method, more than 10% of outliers are tolerated and only the size of the Trail-Map increases (ref. Fig. 3.38b), since the outliers are usually added as new landmark views in level 1. Even with 50% outliers, homing is still successful. This shows, how much the homing vector calculation method affects the homing performance under the presence of false landmark matches. The normalized improved difference vector model produces higher path errors in the noise-free case than the version without normalization, because information about the magnitude of the change of the bearing angles is neglected in the normalized version. The path errors decrease for up to 50% outliers, which can be explained by the fact that the dissimilarity measures increase under the presence of outliers. Thus, the size of the region where the robot assumes to have reached a viewframe decreases. This leads to more accurate path following.

3.7.3. Combination of all Noise Sources

To show the homing behavior with a combination of all error sources we use the noise values as given in Table 3.2. The resulting Trail-Map has 5600 landmark views, the path error is 29.5m^2, the endpoint error is 0.4m and the homing path length is 137m. Fig. 3.39 shows the development of the path error and the path length when the map is pruned. As already shown in Section 3.6 for the noise-free case, the Trail-Map can be pruned by more than 50% without a significant loss of path accuracy.

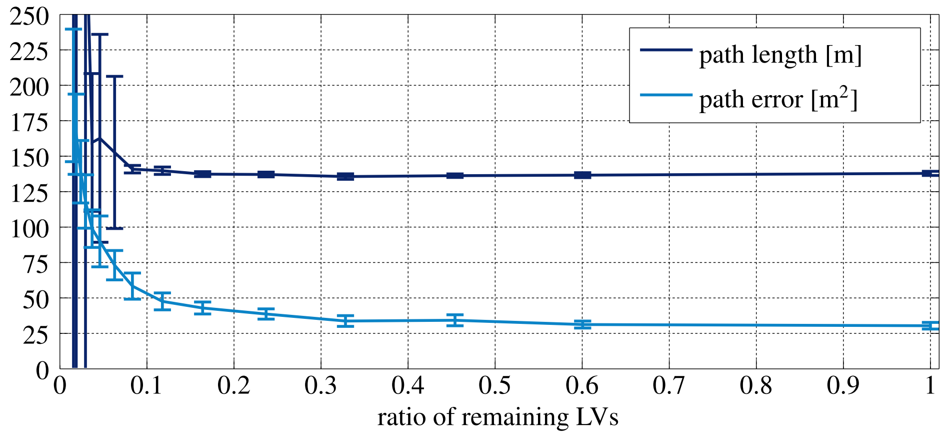


Figure 3.39.: Path error and path length for different pruning ratios under the presence of noise and occlusions

Table 3.2.: Chosen noise parameters for simulating Trail-Map-based homing

parameter	σ_{trans}	$\sigma_{\text{rot,acc}}$	$\sigma_{\text{rot,abs}}$	σ_{px}	p_{occ}	p_{outl}
value	0.1m	0.0°	1.0°	1px	0.5	0.0

The endpoint distance is omitted in this plot since its average values fluctuate strongly between 0.3m and 1.2m for different pruning levels. For applications where the home position must be reached with a very high precision, e.g. for docking to a charging point, artificial landmarks should be used that are clearly visible to the robot from a distance of about 1.5m. Furthermore, the step size for reaching the home position should be decreased significantly.

3.8. Trail-Map-Based Homing versus SLAM

The application of SLAM methods is often the first choice when it comes to the navigation of a mobile robot in real-world environments, where measurement uncertainties arise. SLAM concurrently optimizes the robot's estimates of its pose or path and the map, using odometry measurements and environment observations. Thus, the robot creates a consistent metric map of its environment and it knows its position with respect to the map.

In general, SLAM methods benefit most from reobserving landmarks when the robot returns to a previously visited location. While the robot is moving through the environment, the uncertainty of its pose estimate grows, because it is based only on relative odometry measurements. This pose uncertainty also affects the uncertainty of landmark positions which the robot observes and adds to its map. The robot performs a loop closure when it reobserves a landmark that had already been added to the map. Then, the robot can correct its pose estimate and also the estimated landmark positions in the map. In cases where the robot does not perform any loop closures, the uncertainty of its pose keeps growing, as does the uncertainty of the landmark positions in its map.

Considering applications where the mobile robot is only required to retrace a previously traversed path, a globally consistent metric map is not necessarily required. The robot only needs knowledge about the traveled path to be able to autonomously find its way back to its starting position. The Trail-Map implicitly contains all the information required to retrace the path. When using SLAM for the homing task, the robot has to store its path additionally to the estimated map[3]. During homing, the robot localizes within the estimated map and

[3]Some SLAM methods, for example FastSLAM and GraphSLAM, already contain the full robot trajectory.

computes control commands to follow the stored path coordinates according to its pose estimate.

In the robot homing application, we assume that the robot does not visit any places along its learning trajectory again. During homing, the robot actively moves along the previously recorded path, so that many loop closures occur. However, if a map of the environment and the estimated coordinates of the robot path are not required, the robot can omit the loop closure SLAM updates, and can use the already recorded map for localization. Considering this special case, SLAM methods might show different properties than in the general case of possible loop closures.

We should also address the memory requirements of SLAM methods and the Trail-Map. The Trail-Map stores landmark views, which are a combination of the landmark's ID, its descriptor and its bearing angle. Thus, the Trail-Map contains many different landmark views belonging to a single landmark. In contrast, the map in feature-based SLAM stores the landmark's descriptor and the landmark coordinates only once, as long as the descriptor can be matched. Here, SLAM methods seem to outperform the Trail-Map. However, SLAM methods do not only require memory for storing the map, but also for additional information that is necessary to perform map corrections, for example covariance matrices.

In this section we will discuss how different SLAM methods would perform the homing task and compare them to Trail-Map-based homing.

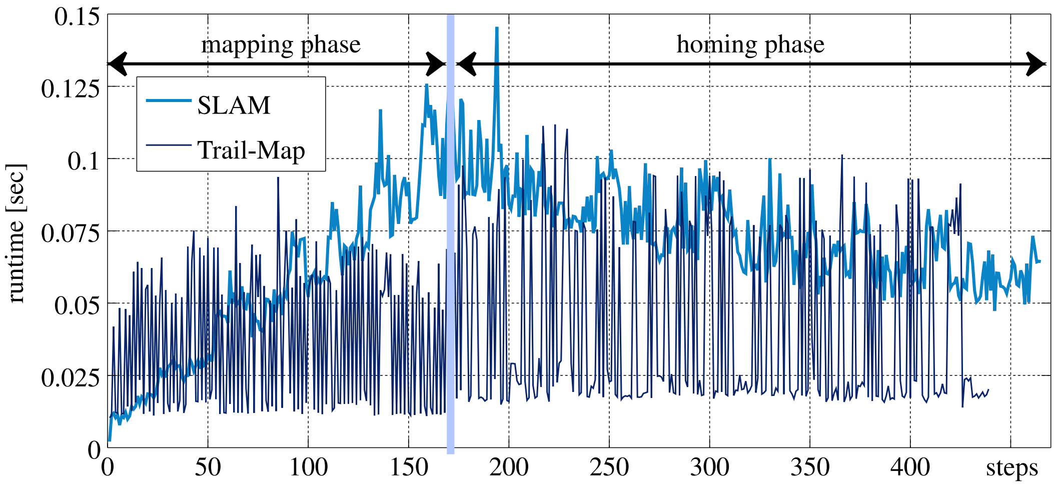
3.8.1. EKF SLAM

In EKF SLAM, an extended Kalman filter (EKF) is used to estimate a state vector consisting of the robot's current pose and the landmark positions. The covariances of the robot pose and the landmark positions are kept in a covariance matrix. When new landmarks are observed, they are added to the robot's state vector. Thus, the state vector and the corresponding covariance matrix grow with the number of observed landmarks, which means that the memory requirements are quadratic in the number of landmarks due to the covariance matrix. Since in a Kalman filter framework, the covariance matrix has to be inverted in every update step, the computational complexity of EKF SLAM is also quadratic in the number of state variables, and, hence, in the number of landmarks in the map.

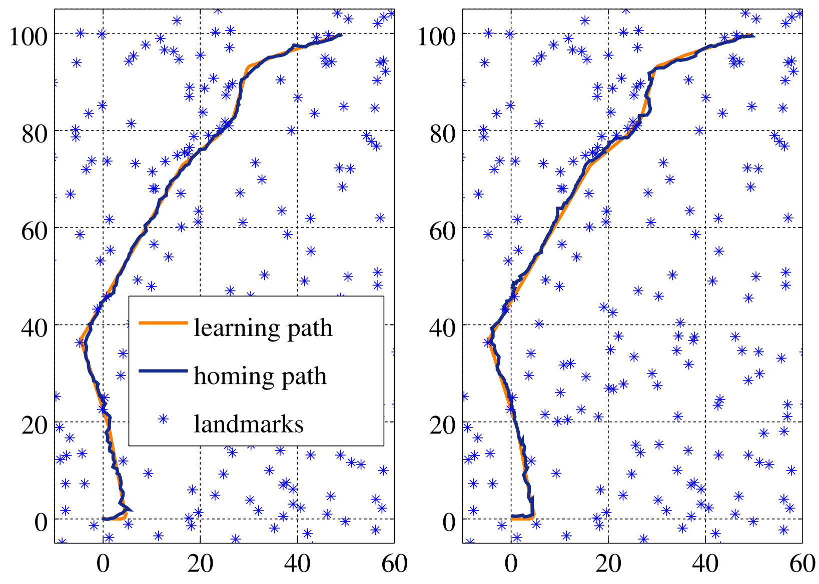
We compared the EKF-SLAM method implemented in the EKF-SLAM Toolbox [84] to the Trail-Map-based homing method. For this, we implemented the SLAM-based homing using the setup described in Section 3.7 and shown in Fig. 3.30 as follows: During the learning track, the robot performs the SLAM prediction and correction steps every 0.75m and stores its estimated position as a waypoint. During the homing phase, the robot calculates a homing vector that leads it to the previously recorded waypoint and drives in that direction for the size of the homing vector step of 0.5m. It assumes it reached the waypoint when its distance to the waypoint is smaller than 0.5m, or when the calculated homing vector changes by more than 90°. When the robot approaches the last waypoint, i.e. the home position, the threshold for reaching the waypoint is set to 0.1m. This ensures that the robot tries to reach the start point with a high precision – comparable to the Trail-Map-based homing to the first viewframe. After each homing step, the robot performs the SLAM prediction and correction.

To reduce the computational requirements of the EKF SLAM method, the authors of the EKF-SLAM Toolbox limit the number of updated landmarks per step to the 5 landmarks with the highest uncertainty [83] and initialize only one new landmark in every step. To save computation time, we additionally switched off the initialization of new landmarks during the homing phase. We simulated the SLAM-based homing over a path of about 130m length using the setup described in Section 3.7 and the noise parameters given in Table 3.2. The results of averaging 20 runs are shown in Table 3.3. The homing path error is 65.4m^2, the endpoint error is 0.58m, the path length is 143m and the final map contains 419 states. In addition to the state variables, the robot needs to maintain a symmetric covariance matrix with 419×419 entries. The runtimes for the SLAM-related initialization and correction steps sum up to 31sec. The development of the computation time is shown in Fig. 3.40a. The computation time increases during mapping as the number of landmarks in the map increases. During homing, inconsistent landmarks are deleted from the map [85], which results in decreasing computation times.

The Trail-Map-based homing computations need only 19sec with the full map, and 17sec with a map pruned by 5 levels to 920 landmark views, while still producing a smaller path error of 42.0m^2, shorter paths of 136.5m and similar endpoint errors of 0.57m (ref. Fig. 3.40b-Fig. 3.40c). The benefit of the Trail-Map would increase with the robot's motion range since the robot would observe more landmarks and the computational load of the EKF-SLAM grows with the number of landmarks in the map. Furthermore, we expect that the runtime differences between the Trail-Map and the SLAM implementation will be more significant in other programming environments than MATLAB, since C++ is better suited for the list data structure of the Trail-Map, while EKF-SLAM mainly requires matrix operations.



(a) Runtimes per step



(b) Pruned Trail-Map (c) EKF SLAM

Figure 3.40.: Comparison of Trail-Map-based and EKF-SLAM-based homing

Table 3.3.: Simulation results of EKF-SLAM from the EKF-SLAM Toolbox [84] and Trail-Map-based homing using the setup shown in Fig. 3.30 averaged over 20 runs

	EKF SLAM	Trail-Map
path error	65.4m^2	42.0m^2
path length	143.0m	136.5m
endpoint error	0.58m	0.57m
map size	419 states	920 LVs
summed computation times	31sec	17sec

3.8.2. Submap SLAM

An efficient way to apply SLAM to the homing problem would be a submap SLAM approach [22, 57, 56]. The map of the environment is divided into overlapping submaps, which are locally optimized using SLAM. The local submaps are bounded in size, which leads to constant time local SLAM updates relating to the path length. The coordinate frames of the local submaps are either referenced to a global coordinate frame or relative to the adjacent submap coordinate frames, and their locations can also be optimized using SLAM. However, since the submaps do not have to be globally consistent for the homing task, no correction of the submap relations must be performed. Thus, constant time mapping and homing is possible. The idea of using submaps is to some extent similar to the viewframe-based mapping approach, which also stores all visible landmarks at the viewframe positions. However, the Trail-Map avoids redundancies by only storing landmark views that have changed significantly. Submap SLAM does not offer such a method, but since it contains the coordinates of each landmark in the map only once, instead of storing several landmark views of a landmark, the memory requirements for the maps might be comparable. However, to perform the local SLAM updates, submap SLAM additionally requires the covariance matrices for the submaps, which are quadratic in the number of landmarks per submap. In case the robot omits local SLAM updates during homing but only performs localization, the covariance matrix could also be deleted after leaving the corresponding submap in the mapping phase.

3.8.3. FastSLAM

FastSLAM [68] is based on a Rao-Blackwellized particle filter to solve the SLAM problem. It uses a particle filter to estimate the full trajectory of the robot and estimates the landmark locations by EKFs. By doing so, it can exploit the fact that the landmarks observed from two different robot poses are conditionally independent, given the robot poses. This allows to factor the map estimation problem into m single estimation problems for each of the m landmarks in the map. Efficient implementations of FastSLAM have a complexity of $O(n \log(m))$, where n is the number of particles and m is the number of landmarks in the map [91]. The main drawback of FastSLAM is that a high number of particles is required to ensure fast convergence, and that the amount increases exponentially in the number of dimensions. This not only means high memory requirements, but also leads to an increased computation time.

3.8.4. GraphSLAM

In GraphSLAM, all the information the robot collects as it moves through the environment is stored in a graph [91]. This graph consists of the robot poses and the landmarks as nodes. Each pose node is linked to the landmark nodes the robot has observed at this location by edges representing a measurement constraint. Subsequent pose nodes are also connected by edges which represent the robot odometry estimates as another constraint. The edges in the graph can be thought of as springs in a spring-mass-model, which automatically relaxes and yields the optimal distribution of the masses, which are the nodes representing the robot poses and the landmarks in the map. When this graph is transformed into an information matrix, this matrix is sparse, because it only has elements between subsequent robot poses, and between the landmarks and the poses of the robot at the time of observation. This can also be explained by the conditional independencies between landmarks and poses given all robot poses and landmarks. The sparse character of the information matrix can be exploited for fast matrix operations. The optimal map and robot trajectory can be computed by nonlinear least squares optimization, e.g. using g^2o [52]. Smoothing and mapping (SAM) [18] or its incremental version iSAM [44] are also solutions to the GraphSLAM problem. In iSAM the update is restricted to the part of the state vector that actually changed, thus resulting in constant-time updates as long as no loop closure is detected. Retrieving the current estimates of the trajectory and the map is linear in the number of variables the trajectory and map consist of, but can also be decreased to constant time when only the recently changed variables are extracted, which is sufficient for the homing task. However, since each landmark measurement has to be stored for GraphSLAM, the memory consumption is higher than in the Trail-Map, which avoids storing landmark views redundantly.

3.8.5. RatSLAM

RatSLAM [67] differs from all the above mentioned SLAM methods. Instead of creating a metrically correct map of the environment it rather computes a topologically consistent representation. RatSLAM is based on the hippocampal model of rodents. The robot pose is represented by regions of high activity in a competitive attractor network consisting of so-called *pose cells*. Odometry measurements shift the region of activity in a corresponding direction. The pose cells are associated with *local view cells*, which are triggered by the visual appearance at a location. The associations between the pose cells and the local view cells are learned. When a local view cell fires, it injects activity in the associated pose cells and, thus, corrects the pose estimate of the robot. The combination of a pose cell activity pattern and a local view cell activity pattern is stored as an experience in an *experience map*. Each experience is assigned a position in experience space and subsequent experiences are connected by transitions which contain the position changes between the experiences measured by odometry. Whenever an experience matches an already stored experience, a loop closure is detected and all positions assigned to the experiences are corrected accordingly. The experience map can be used to plan and execute paths [66].

Applied to the homing task, RatSLAM behaves similar to the viewframe-based homing method presented in this chapter. A topological chain of experiences is stored during the mapping phase, where each experience contains the visual appearance at the corresponding location in the environment. This process runs in constant time if the experience map is not searched for loop closures. For homing, the robot has to execute the transitions between the experiences in the experience map. Depending on how the appearance of the locations is encoded in the local view cells, the robot might have to rely on good odometry estimates. If it cannot generate motion information by comparing the appearance of the environment with the desired activity pattern of the local view cell, it will be lost when it deviates too much from its learned path. Since RatSLAM does not extract landmarks but is based on the appearance of the environment, we cannot estimate its memory consumption compared to the Trail-Map.

3.8.6. Data Association

All SLAM methods, as well as Trail-Map-based navigation, have to find correspondences between the landmarks in the map and the current landmark observations. For this, each currently observed landmark has to be compared to all possibly corresponding landmarks in the map. This process can be linear in the number of landmarks in the map in a naive implementation. By organizing the landmarks in a tree structure according to their estimated coordinates, for example in a quadtree, octree or kd-tree, the complexity of data association is logarithmic in the size of the map. Constant complexity is achieved when the amount of landmarks in question is limited to a subset of all observed landmarks which does not grow with the length of the traversed path. This can be realized by storing the landmark observations in a grid with fixed cell sizes, at the expense of the required memory. Furthermore, when the map is divided into submaps which are limited in size, also only a constant number of landmarks has to be compared to the current observations, provided that the robot knows which submap to search in. This shows that not only the complexity of the SLAM method itself has to be considered, but also the implementation of data association has a major effect on the runtime. In the Trail-Map-based homing method, the number of landmarks in question is limited by the number of landmark views in the current reference viewframe, which is independent of the length of the path. Thus, the Trail-Map implicitly offers constant complexity for the data association task.

3.8.7. Conclusion

Table 3.4.: Comparison of SLAM methods and the Trail-Map for the robot homing task. m: number of observed landmarks in the environment. n: number of particles in FastSLAM. p: number of submaps, viewframes or poses with landmark observations

	computational complexity	memory requirements
EKF SLAM	$O(m^2)$	$O(m^2)$
Submap SLAM	$O(1)$	$O(p)$ comparable with Trail-Map
FastSLAM	$O(n \log(m))$	$O(n \cdot m)$
GraphSLAM	$O(1)$	$O(p)$ but higher than Trail-Map
RatSLAM	$O(1)$	$O(p)$ but unknown factor
Trail-Map	$O(1)$	$O(p)$

Table 3.4 summarizes the computational complexities and memory requirements of the discussed SLAM methods, when applying them to the task of robot homing. The special property of robot homing is the absence of loops in the robot path during mapping. During homing, the loop closures can be neglected and the estimated map can used for localization. Thus, when no loop closures occur, some SLAM solutions offer changed properties. Others, like EKF SLAM and FastSLAM cannot benefit from the absence of loops. The EKF SLAM solution is intractable due to its quadratic computational complexity. The FastSLAM solution is logarithmic in the size of the map, but has a high constant factor caused by the large number of particles that are required. A SLAM solution that can offer constant runtimes is a submap SLAM approach without updating the relations between the submaps. Its memory requirements are comparable with those of the full (unpruned) Trail-Map, if the covariance matrices of the submaps are discarded. RatSLAM also offers constant runtimes if no loop closures are detected, but must implement a method for encoding the appearance of a location which can also be used if the robot deviates from its path. GraphSLAM can be solved in

constant time for the homing task as well, but suffers from a memory overhead since every single measurement has to be stored.

In addition, a SLAM method without loop closure detection can no longer be considered a true SLAM method. Put differently, to the best of my knowledge there is no true SLAM method which offers runtime behavior constant in the size of the map and the path length. Constant runtimes are only achieved by a few methods when loop closures are neglected, and some of these methods suffer from memory overheads. Furthermore, none of the discussed SLAM solutions offers a scaling method. The Trail-Map representation is memory efficient, because it avoids storing redundant measurements, and it has a computational complexity that is constant with respect to the path length. Furthermore, it offers an easy scaling method. Thus, a Trail-Map based solution is a promising alternative to SLAM methods for robot homing without loop closures.

3.9. Application of the Trail-Map to Real Data

Bringing the homing method to the real world poses several challenges that were not present in the simulations. First of all, we have to extract landmarks as natural features in omnidirectional camera images, which are subject to noise and illumination changes. Furthermore, the landmarks have to be identified by matching the detected image features to their corresponding landmarks in the map. Additionally, the robot has to estimate its orientation for correctly aligning the viewframes in the Trail-Map. Apart from estimating the yaw angle, the real robot does not move on a perfect plane anymore but can encounter tilt which also has to be considered when measuring the directional unit vectors to landmarks.

This section will address these challenges and describe the approaches taken to prepare the experimental validation of the Trail-Map-based homing method.

3.9.1. Coordinate Frames

In the simulations, the robot always moved on a perfect plane, which allowed to parameterize its orientation only by the yaw angle. For experiments with a real robot, we also have to consider roll and pitch angles in the computation of unit vectors to the landmarks. Therefore, 3D coordinate transformations are necessary.

First, we unwarp the omnidirectional camera images to panorama images. Then, we transform a feature location in the panorama image coordinate system u, v into a unit vector in the camera coordinate system using

$$\boldsymbol{l}_i^c = \begin{bmatrix} \cos\phi_{i,e}\cos\phi_{i,a} \\ \cos\phi_{i,e}\sin\phi_{i,a} \\ \sin\phi_{i,e} \end{bmatrix}^T \tag{3.25}$$

$$\phi_{i,a} = \frac{2\pi}{n_{cols}}(-u_i) \tag{3.26}$$

$$\phi_{i,e} = \frac{2\pi}{n_{cols}}(v_0 - v_i), \tag{3.27}$$

where $\phi_{i,a}$ and $\phi_{i,e}$ are azimuth and elevation, respectively. The row coordinate corresponding to the horizon is v_0. The value n_{cols} contains the number of image columns.

To transform the vector $\boldsymbol{l}_i^c$ to the robot coordinate frame, we have to know the rotation $\boldsymbol{R}_c^r$ between the robot and the omnidirectional camera. This transformation defines the image forward direction. The landmark unit vector in the robot coordinate frame is

$$\boldsymbol{l}_i^r = \boldsymbol{R}_c^r\,\boldsymbol{l}_i^c. \tag{3.28}$$

Finally, to have rotationally aligned viewframes, we transform the landmark vectors to the world coordinate frame. We define the origin of the world frame to be at the robot's home position. The orientation of the world frame is aligned with the gravity vector and is defined to have zero yaw angle at the home pose. Then, the landmark vector in world coordinates is

$$\boldsymbol{l}_i^w = \boldsymbol{R}_r^w\,\boldsymbol{l}_i^r, \tag{3.29}$$

where we compute the matrix $\boldsymbol{R}_r^w$ from the robot pose, which is estimated in the multisensor data fusion process as described in Section 2.1. While the roll and pitch angles in this estimate are stabilized by the gravity vector, the yaw angle results only from relative measurements when no compass is available. Thus, we will perform additional corrections of the yaw angle using the landmark information. This process is described below.

3.9.2. Landmark Detection and Matching

The main issue of the real world implementation is the landmark detection and matching process, which was neglected in the simulations, where we assumed known data association. In the simulation sections, we used the term *landmark* for a characteristic point or object in the environment that can be observed and identified from several locations. When cameras are used for observing landmarks, we assume that the projection of the landmark into the camera image results in a *feature*. Thus, in this thesis, the term *feature* refers to a characteristic point in an image, that corresponds to a *landmark* in the environment. Measures for the degree of characteristic are often based on the image gradient for detecting regions of high or low local contrast. This, in practice, often leads to image features being detected on object borders due to the high contrast between the object in the foreground and the background. Although such a feature does not correspond to a certain object in the environment, it can still be used for navigation purposes. However, it will only be visible within a relatively small angle and changes or disappears if the viewpoint shifts.

Commonly used state-of-the-art feature extractors are for example SIFT [61], SURF [4] and BRISK [58]. They are all invariant against rotation and scale changes. A comparison of SIFT, SURF and BRISK performed by Jäger [42] showed that there was not much difference between those three, but BRISK is the computationally most efficient feature detection and matching method. BRISK is based on AGAST [62] and FAST [75] to detect keypoints in scale space. Then, a binary descriptor is computed from brightness comparisons within a circular sampling pattern. The descriptors are matched using Hamming distance, which is only an XOR operation followed by a bit count, and, thus, can be computed very fast.

Fig. 3.41 shows the matches obtained using the original BRISK C++ code provided by Leutenegger et al. [58] for two sample outdoor images. The panorama images from the omnidirectional camera were downscaled to 30% i.e. 432×84 pixels to reduce the effects of pixel noise. The BRISK features were detected with a FAST/AGAST detection threshold of 10, an octave number of 4, a pattern scale of 0.75 and a matching distance of 500. About 580 features were detected in each image and 135 features could be matched. Apparently, there are several false matches that have to be rejected.

Depending on the homing vector calculation method, the Trail-Map navigation behavior is very sensitive to false landmark matches as the simulations have shown (ref. Section 3.7.2). Thus, it is important to reject false landmark matches. Therefore, we make use of the method suggested by Jäger [42]. That method is based on a RANSAC (Random Sample Consensus)

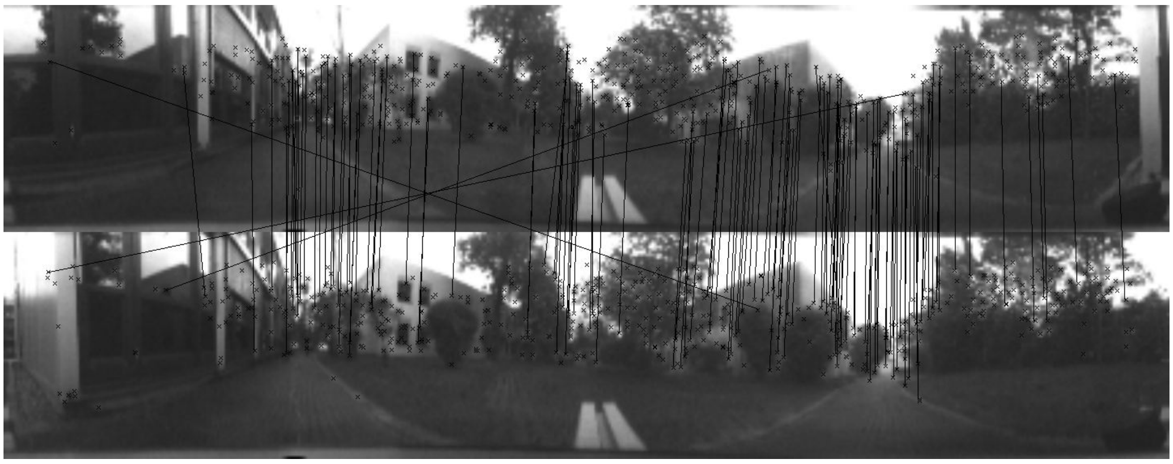


Figure 3.41.: Detected and matched features using BRISK

algorithm [23], which finds a maximum set of inliers by fitting a model to randomly drawn samples und checking how many data points are conform to the model. Pseudocode for the outlier rejection is given in Alg. 3 in Section A.3. First, the RANSAC algorithm is applied to the feature matches using a translational invariant model for estimating the rotation between the two image acquisition points (ref. Alg. 7). Details on the computation of the rotation are given in Section A.1. Then, RANSAC is applied to the remaining feature matches using a translational variant model to estimate the translation between the images (ref. Alg. 6). Section A.2 gives details on computing the translation. The resulting valid matches and outliers for the two sample images are shown in Fig. 3.42. In this example, 14 false matches have been identified and rejected, using error thresholds of 0.75° for the translational and rotational model. For the RANSAC algorithm, a maximum number of 50 iterations was found to be sufficient. Furthermore, the algorithms terminate earlier when 90% of the input landmark matches fit to the model.

The above mentioned rotational model does not make use of the roll and pitch angles from the pose estimation process. However, since roll and pitch measurements are absolute measurements, their values are known with a constantly low uncertainty, and, thus, should be considered in the landmark outlier rejection process. For this reason, we propose a different model for computing the rotation between two sets of unit vectors to landmarks with known roll and pitch angles. The model assumes that the landmark unit vectors $\boldsymbol{l}_i^w$ and $\boldsymbol{l}_i^{w'}$ are already aligned using the estimated roll and pitch angles. Then, the yaw angle α that best

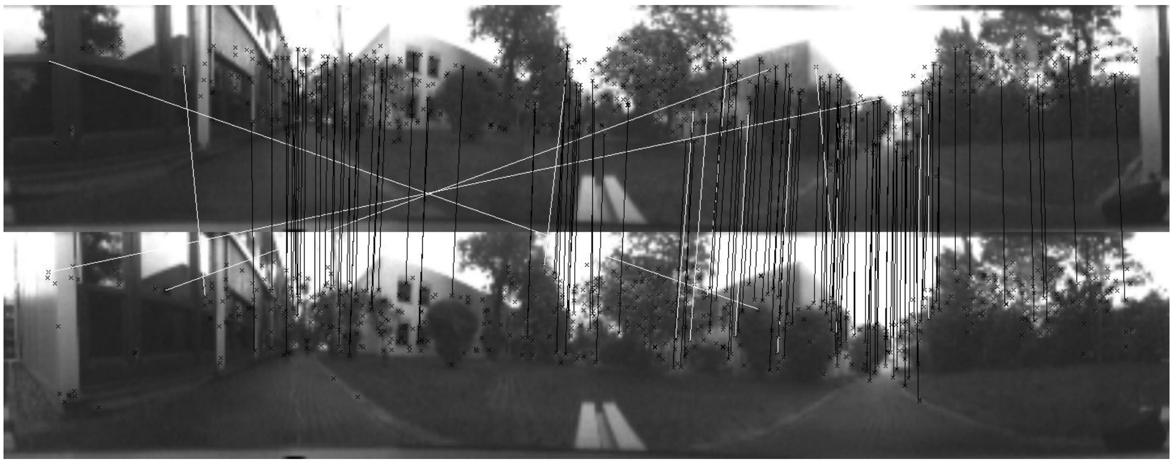


Figure 3.42.: Valid landmark matches (black) and rejected outliers (white)

explains the rotation between the two sets of landmark vectors can be computed as the mean of the signed yaw angles between the corresponding landmark unit vectors:

$$\alpha = \frac{1}{N}\sum_{i=1}^{N}\left(\operatorname{atan2}(\boldsymbol{l}^{w'}_{i,y}, \boldsymbol{l}^{w'}_{i,x}) - \operatorname{atan2}(\boldsymbol{l}^{w}_{i,y}, \boldsymbol{l}^{w}_{i,x})\right), \tag{3.30}$$

where $\boldsymbol{l}^{w'}_{i}$ is the landmark unit vector in the current frame and $\boldsymbol{l}^{w}_{i}$ is the corresponding landmark unit vector in the known reference frame.

3.9.3. Rotational Alignment of the Viewframes

As the simulations in Section 3.7 have shown, the homing algorithm is very sensitive to rotational misalignments of the viewframes. If no reliable compass is available, the estimated yaw angle by fusing inertial and visual odometry data (ref. Section 2.1) will drift over time. That does not pose a problem during the mapping phase, because the viewframes are recorded sequentially and the robot performs only relatively small motions between two subsequent viewframes. However, during homing the error in the yaw estimate can be large, especially for homing towards early viewframes close to the home position. For this reason, we use the rotation that is estimated by the RANSAC algorithm in the outlier rejection step for aligning the viewframe before dissimilarity measures or homing vectors are computed. Thus, we can entirely omit a compass or any other yaw estimate.

3.9.4. Homing Vector Smoothing

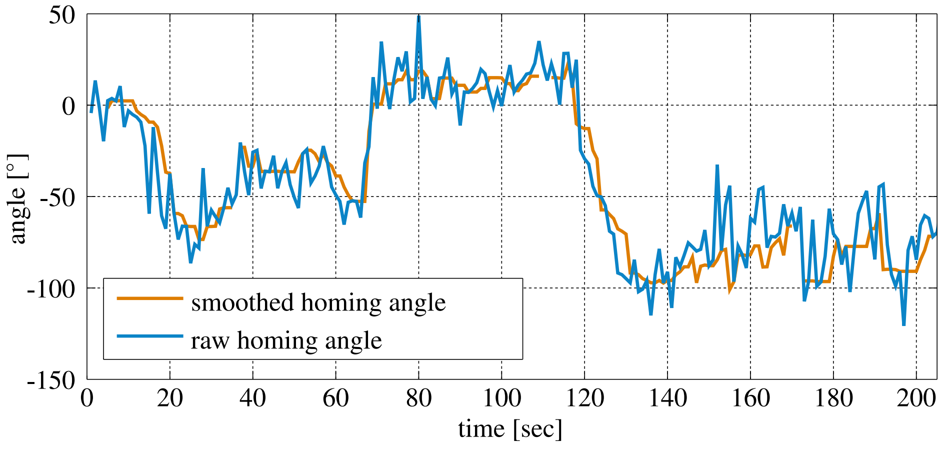


Figure 3.43.: Raw homing angles computed by the improved difference vector model. Smoothed homing angles computed as the median of the last 7 raw homing angles.

Since the single homing vectors computed from the current panorama image and the goal viewframe are noisy, we use the method described by Jäger et al. [43] for smoothing the homing vectors. In this method, the median out of the last 7 homing vectors to the current goal viewframe is computed. To avoid waiting times when the robot reaches a viewframe and starts homing to the next one, the robot computes homing vectors to the current and the following goal viewframe simultaneously for each panorama image. Thus, a median homing vector for the next goal viewframe is already available when the robot starts homing to it. Fig. 3.43 shows the raw homing angles computed by the improved difference vector model, and the resulting homing angles after smoothing for a sample homing run.

3.9.5. Robot Motion Control

To make the robot follow the computed homing vector, we applied a very simple motion strategy. A homing vector is calculated continuously as the robot moves through the environment. At every time step, the robot computes its forward velocity v_x and its turn velocity

ω_z depending on the difference between its current yaw angle α and the current homing vector direction ϕ_h as

$$v_x = v_{\max} \max\left(0, 1 - \frac{|\phi_h - \alpha|}{\phi_t}\right), \tag{3.31}$$

$$\omega_z = \omega_{\max} \max\left(-1, \min\left(1, \frac{\phi_h - \alpha}{\phi_t}\right)\right), \tag{3.32}$$

where $v_{\max}$ and $\omega_{\max}$ are the maximal forward and turn speeds. If the difference between the current yaw angle and the current homing vector direction is higher than a threshold ϕ_t, the robot performs pure turning. Otherwise, the robot combines forward and turn motions, while the forward velocity increases and the turning velocity decreases the closer the robot's yaw angle gets to the desired homing direction.

3.10. Experimental Evaluation

To experimentally evaluate Trail-Map based homing, we used a Pioneer 3-DX robot equipped with an omnidirectional camera, a stereo camera and an inertial measurement unit. The omnidirectional camera was a catadioptric system with a PointGrey USB camera providing images of 896×896 pixels, which were unwarped to panorama images of 1440×280 pixels and downscaled to 30%, i.e. 432×84 pixels, for landmark detection. The stereo camera and the IMU are used for pose estimation by fusing visual odometry and inertial data. However, only the estimated roll and pitch angles are used for computing aligned landmark unit vectors. The computational hardware is an Intel Core i7-3740QM CPU with 2.70GHz and a Spartan 6 LX75 FPGA Eval Board to perform dense stereo matching using SGM [39] at a rate of more than 10Hz. The next section will show the results of experiments conducted in an indoor laboratory environment. After that, in Section 3.10.2 we will analyze the long-range performance based on experiments in outdoor terrain.

3.10.1. Indoor Laboratory Experiments

The first set of experiments was conducted in an indoor laboratory environment on flat ground without obstacles. Ground truth for the experiments was provided by an optical tracking system based on several infrared cameras that track a reflecting target body mounted on the robot.

Mapping

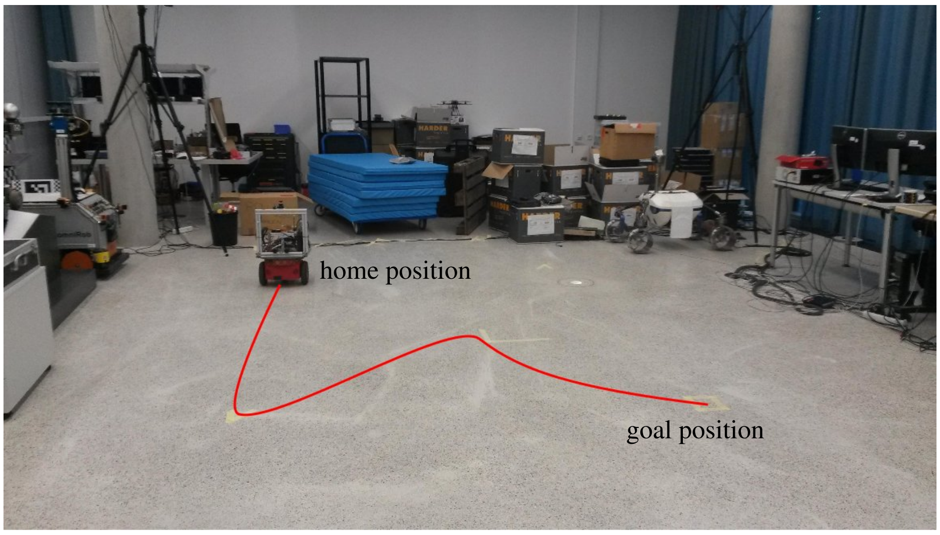


Figure 3.44.: Experimental setup for the indoor laboratory experiment with home position, path and goal position

We remotely controlled the robot along a trajectory of about 5.1 meters length to a goal point in the laboratory environment. The setup and the path are shown in Fig. 3.44. During the traverse the robot recorded a Trail-Map, using an angle threshold of $\delta_{\text{ang}} = 10°$ and a dissimilarity threshold of $\xi^{\text{map}}_{\delta^{\text{ang}}_{\text{diss}}} = 0.05\text{rad} \approx 2.9°$ for mapping. The viewframe dissimilarities were computed using the average angle error method in Eq. (3.4). The resulting Trail-Map consisted of 17 viewframes which contained 882 landmark views on average. Thus, in total 15000 landmark views were recorded for this path. However, the Trail-Map had only 10992 landmark views, because slowly changing landmark views are not stored redundantly. Thus, the Trail-Map saved more than 26% of memory compared to straightforward viewframe storage. The trajectory with the recorded viewframes, along with the number of landmark views in each viewframe and the landmark matches between the viewframes is shown in Fig. 3.45. It can be seen that the viewframes VF11 to VF15 in the center of the free area are further apart from each other than the viewframes which are closer to the border of the free driving area. This shows that the distances between the viewframes are smaller in regions with close landmarks and grow if only distant landmarks can be observed. The landmark views were spread over 12 levels (ref. Table 3.5), where nearly 75% of the landmark views were located in level 1. The reason for the large amount of landmark views in level 1 can already be inferred from the number of matches between the viewframes, which is on average only 261 compared to an average of 882 landmark views per viewframe. That means that many landmarks in the viewframes cannot be matched with the landmarks in the neighboring viewframes, which leaves them in level 1. Table 3.6 summarizes the statistics of the mapping process.

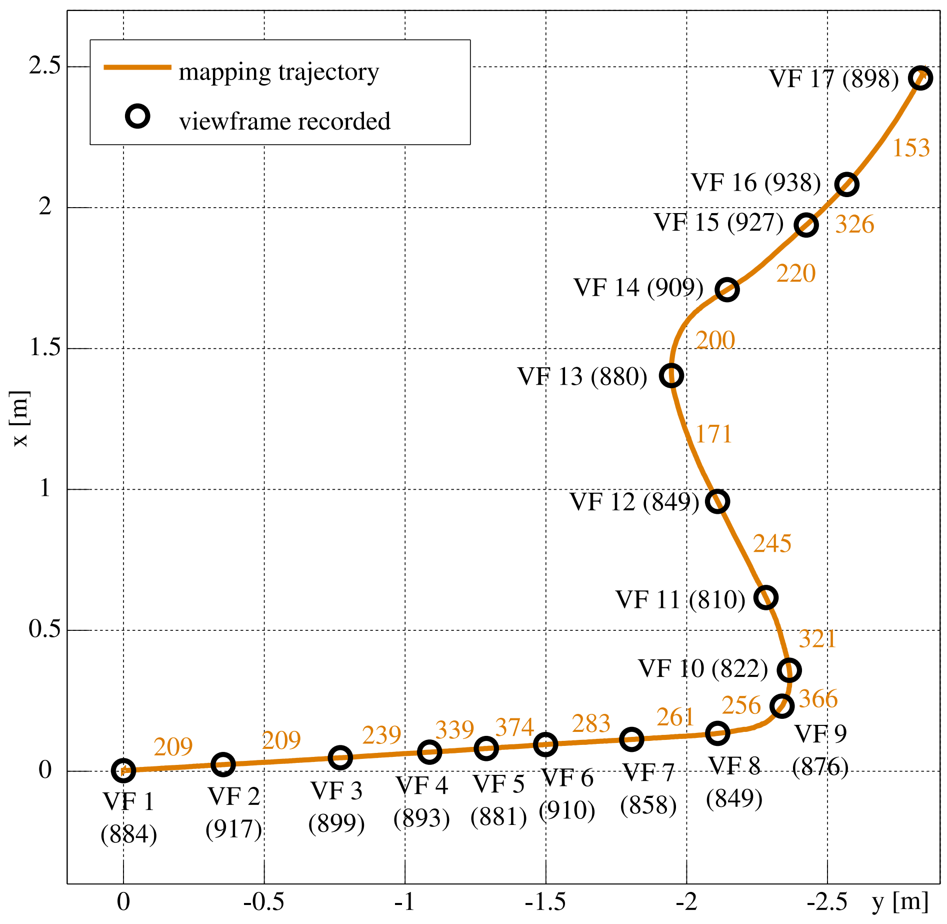


Figure 3.45.: Path with viewframes (VF) for the indoor laboratory experiment. Numbers in brackets: number of landmark views in the viewframe. Orange numbers: number of landmark matches between the viewframes

Table 3.5.: Distribution of landmark views (LV) over the levels in the indoor laboratory experiment

level	1	2	3	4	5	6	7	8	9	10	11	12
LVs	8171	1430	634	328	185	108	79	31	16	6	3	1
ratio	74.3%	13.0%	5.8%	3.0%	1.7%	1.0%	0.7%	0.3%	0.1%	0.05%	0.03%	0.01%

Table 3.6.: Mapping statistics of the indoor laboratory experiment

path length	5.1m
observed LVs	15000
recorded viewframes	17
levels in Trail-Map	12
LVs in in full Trail-Map	10992
avg. LVs per viewframe	882 ± 35.5
avg. matches between viewframes	261 ± 68.1

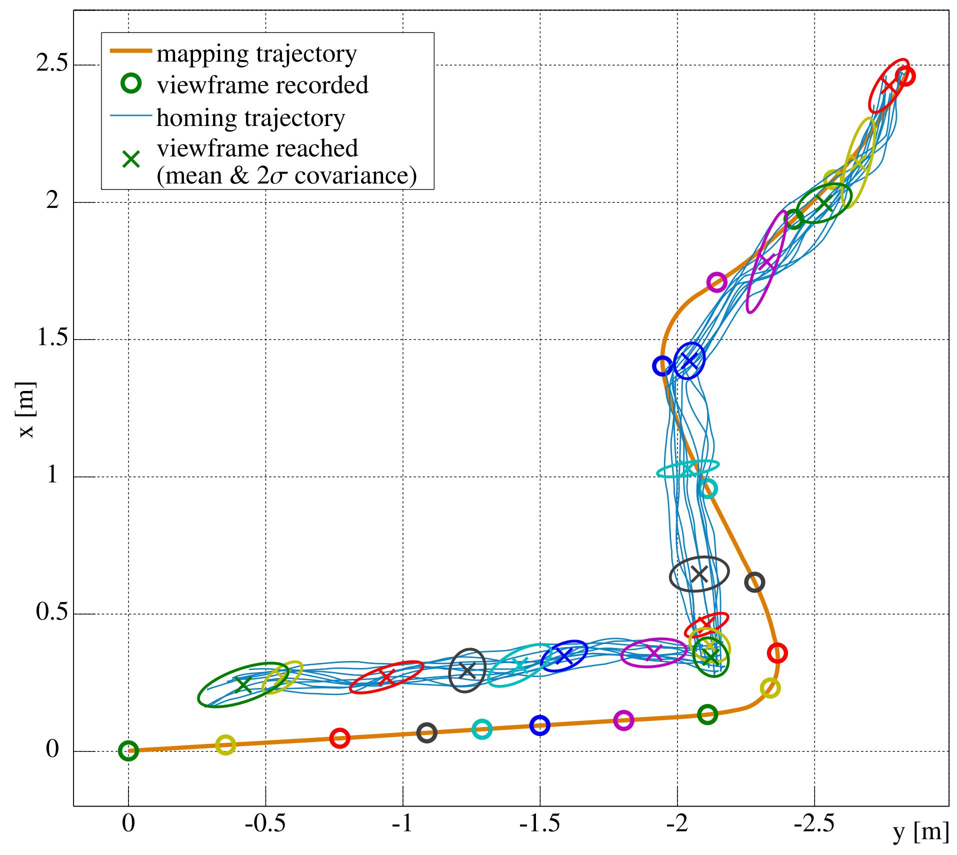


Figure 3.46.: Mapping and homing trajectories using the full Trail-Map in the indoor laboratory experiment

Validation of Homing

Next, the robot performed homing using the recorded Trail-Map. We placed the robot back to the goal position after each homing run and recorded 10 homing trajectories. For homing we used a dissimilarity threshold of $\xi^{\text{hom}}_{\delta^{\text{ang}}_{\text{diss}}} = 0.03\text{rad} \approx 1.7^\circ$ average angle error for detecting when a viewframe is reached. The robot computed the homing vectors using the improved

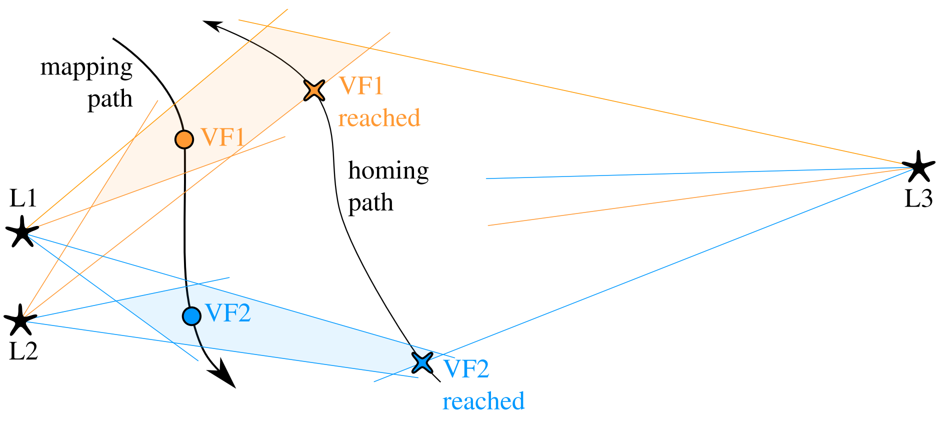


Figure 3.47.: Shape of the areas in which the dissimilarity measures for the corresponding viewframes fall below the threshold $\xi^{\text{hom}}_{\delta^{\text{ang}}_{\text{diss}}}$ depending on the landmark distribution

difference vector model using Eq. (3.7) at a rate of about 2.5Hz. Fig. 3.46 shows the resulting trajectories with the centers and covariance ellipses of the coordinates where the robot reached a viewframe. It can be seen that all homing paths were close together and the robot followed the mapping trajectory. The deviation from the recorded trajectory can be explained by the area in which the dissimilarity measure for a viewframe falls below the threshold $\xi^{\text{hom}}_{\delta^{\text{ang}}_{\text{diss}}}$. The shape and size of this area depends on the landmark configuration in the environment. This relation is illustrated in Fig. 3.47, which shows a mapping path with two viewframes VF1 and VF2 in an environment with three landmarks L1, L2 and L3. Two landmarks on one side are close to the mapping path, the other landmark on the opposite side is far away – that represents the landmark distribution in our experiment for the viewframes VF1 to VF9, where the robot had close objects to its right and only distant objects to its left hand side (ref. Fig. 3.44). The areas in which the dissimilarity measures for the corresponding viewframes fall below the threshold are the shaded polygons that result from the intersections of sectors originating at the landmarks. The opening angle of the sectors is $2\xi^{\text{hom}}_{\delta^{\text{ang}}_{\text{diss}}}$ and the viewframes are in the centers of the sectors[4]. It can be seen that the shape of the shaded area is stretched in the direction of the far landmark. Now, as soon as the robot enters the shaded area of a viewframe, it detects the viewframe as reached and switches to the next viewframe. That explains, why the deviation from the mapping path in Fig. 3.46 had an offset in the direction of the far landmarks.

[4]Please note that this is a simplification. The dissimilarity is also lower than the threshold in some border areas, in which the angle error to one landmark is higher than $\xi^{\text{hom}}_{\delta^{\text{ang}}_{\text{diss}}}$, but which is compensated by the other landmarks having errors lower than the threshold.

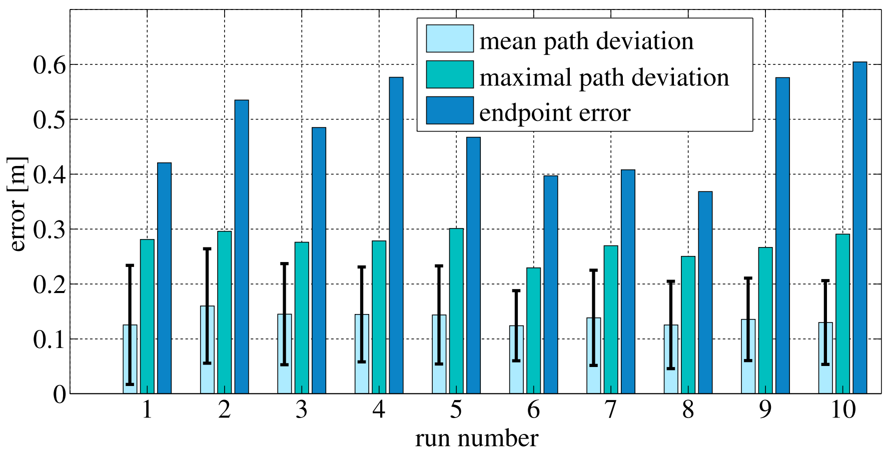


Figure 3.48.: Path errors and endpoint errors for the single homing runs in the indoor laboratory experiment

We evaluated the paths by computing the mean path error and the corresponding standard deviation, the maximal path error and the endpoint error. We computed the path error at a certain point of the homing trajectory as the shortest distance of this point to the mapping trajectory. The path error statistics were then computed from the path errors of all points of the homing trajectory. The statistics of the 10 homing runs are shown in Fig. 3.48. The average of all mean path errors was 0.14m, the maximal path deviation was 0.27m on average and the average endpoint error was about 0.48m. These values, especially the endpoint error seem to be large compared to the path length of 5.1m. However, these values do *not* depend on the length of the path, but only on the configuration of landmarks in the environment. Thus, also for longer paths with the same home position, the endpoint error would be approximately 0.5m. For this reason, we will not give any errors as percentage of the driven path length, but will only state the absolute errors.

Landmark Matching

Fig 3.49 shows how the landmark views in a viewframe were distributed among the levels and how many of the landmark views in each level were matched during the homing process. It becomes obvious that only 8% of the landmarks in level 1 could be matched, while in higher levels this ratio was about 20%. This fact is caused by many *spurious* landmarks being added to the map. A spurious landmark is a landmark which does not correspond to a specific object in the environment but appears as characteristic point in the image at object borders due to the high contrast between the object in the foreground and the background. Thus, it is only visible within a relatively small angle and changes or disappears if the viewpoint shifts. For this reason, the landmarks cannot be matched from other viewpoints, which explains the low matching ratio in level 1 and also the high number of landmark views in this level.

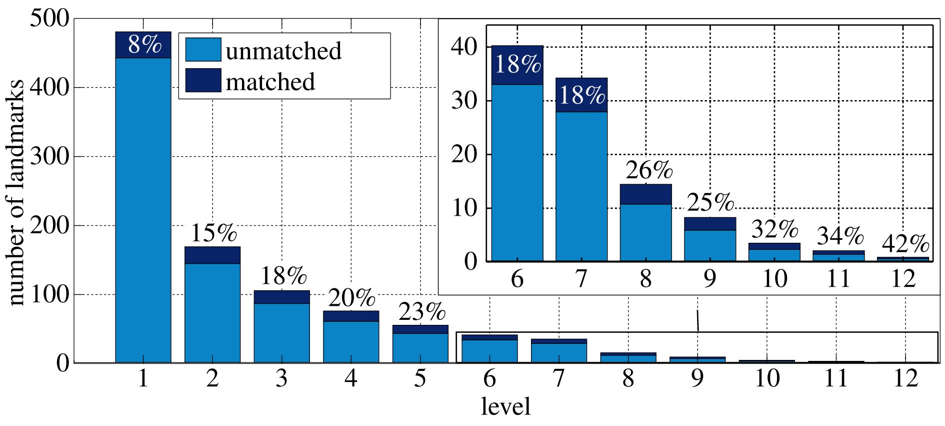


Figure 3.49.: Average number of matched and unmatched landmarks per viewframe and their distribution over the levels in the indoor laboratory experiment. The percentage corresponds to the ratio of matched landmarks.

Fig. 3.50 shows how the landmark matches of the different levels were distributed in the panorama image. It can be seen that the landmarks in the higher levels were located in the direction of motion and on distant objects.

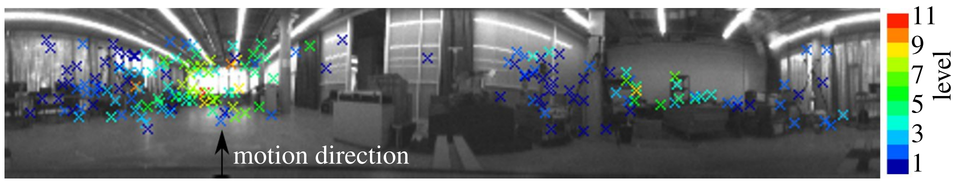


Figure 3.50.: Distribution of the landmark matches of the different levels in the image for the indoor laboratory experiment

Pruning Behavior

To evaluate the homing performance when pruning the recorded Trail-Map at different levels, we performed 10 homing runs for each pruning level. To ensure that the robot reached the home position, the home viewframe was never pruned. The resulting statistics are shown in Fig. 3.51. When pruning one level, already 2/3 of all landmark views were deleted. However, the homing performance degraded only slightly. When pruning the Trail-Map to less than 20% of its original size, also only slightly higher path errors occurred. The endpoint errors were independent from the pruning level since the home viewframe was not pruned. We stopped pruning after 7 levels, because the robot frequently started the searching behavior at pruning level 7. However, it still reached the home position in all trials.

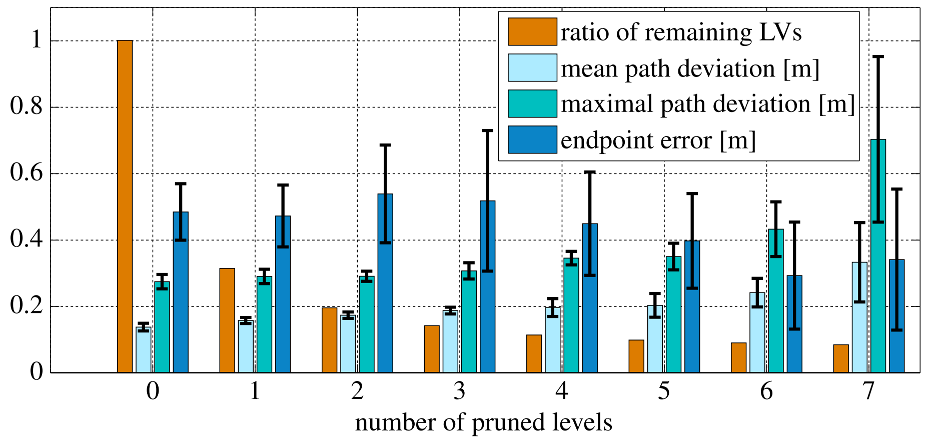


Figure 3.51.: LVs, path and endpoint errors for different pruning levels in the indoor laboratory experiment

To further show how the homing paths change depending on the pruning level, we recorded another Trail-Map for a meandering path of 17.7m length. The Trail-Map contained 64 viewframes and 29586 landmark views spread over 23 levels. Then, we let the robot perform single homing runs at different pruning levels and recorded the homing trajectories. The results are shown in Fig. 3.52. It can be observed that the robot took shortcuts the more the Trail-Map was pruned. However, with only 975 landmark views left (which are only 3.3% of the full Trail-Map size), the robot still found the home position. Of these 975 landmark views, 711 corresponded to viewframe 1, which was not pruned to ensure that the robot reached the home position.

For a better understanding of how much memory is saved, we should note that one landmark view needs about 100 bytes of memory. In detail, a landmark view consists of the landmark's ID (4 bytes integer), the landmark's descriptor (64 bytes for BRISK), a 3D unit vector (24 bytes for 3 double numbers[5]) and the number of the viewframe when the landmark view was added to the Trail-Map (4 bytes integer). Thus, the landmark descriptor is the main memory consumer. The Trail-Map consists of a list of levels, in which each level contains a list of landmark views and the level number (4 bytes integer). In the list of landmark

[5]By knowing that it is a unit vector, only 2 elements would also be sufficient.

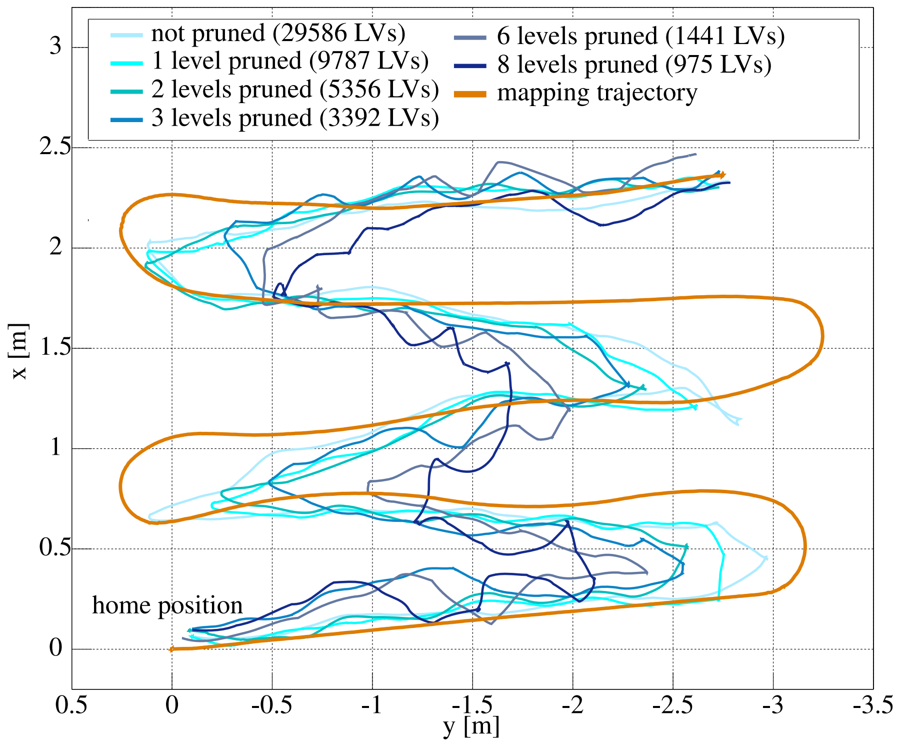


Figure 3.52.: Homing paths for different pruning levels in the indoor meandering experiment

views, each node has some additional memory requirements, depending on the programming environment. Thus, the full Trail-Map of the meandering path of 17m length requires about 3MB, which would extrapolate to about 18MB for 100m. This value is in the region of the memory consumption reported for Visual Teach and Repeat [27] (35MB for 100m) and the method introduced by Krajník et al. [51] (10MB per 100m). However, by pruning the map by 2 levels, the path deviation is still acceptable, but the memory requirements decrease to about 500kB for 17m, which corresponds to approximately 3MB per 100m. Furthermore, when pruning 8 levels of the map, only 100kB are sufficient to still reach the home position – where most of the required memory stems from the first viewframe that has not been pruned. This corresponds to less than 600kB per 100m. Whether this kind of extrapolation holds for longer paths will be discussed in the next section on long-range outdoor experiments.

3.10.2. Long-Range Outdoor Experiments

To show the long-range performance of Trail-Map-based homing, we performed outdoor experiments on untraveled roads on the DLR Oberpfaffenhofen site, using the same Pioneer 3-DX robot as in the indoor laboratory experiments. Ground truth was obtained by a tachymeter, which automatically tracked a prism mounted on the robot and recorded the x, y and z coordinates using an infrared laser beam.

Carpark Experiment

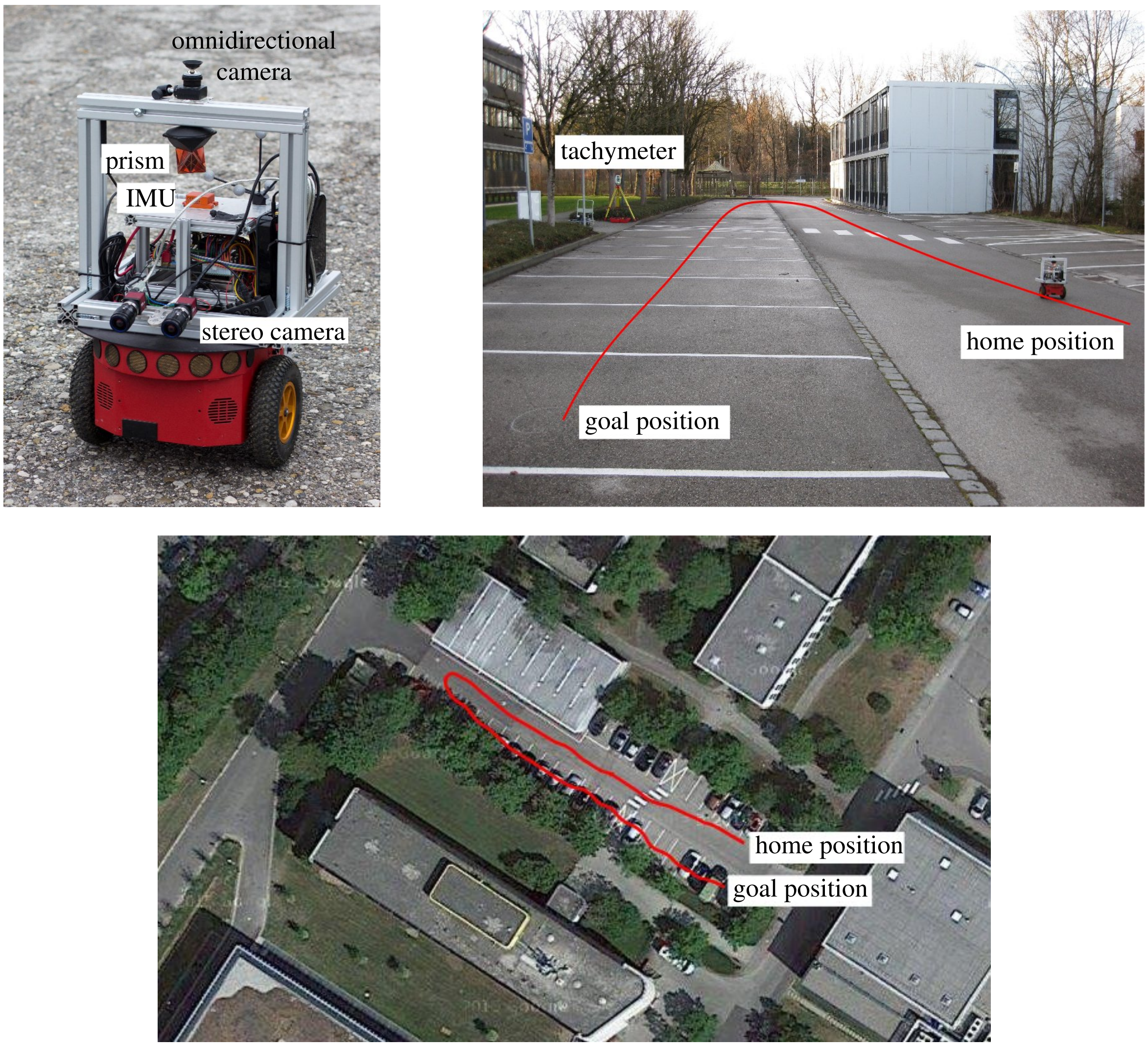


Figure 3.53.: Pioneer robot and experimental setup with path for the outdoor carpark experiment (satellite imagery ©2015, DigitalGlobe, GeoBasis-DE/BKG)

In a first experiment, we remotely controlled the robot along a U-shaped path of 98.7m length in an empty carpark between two buildings. Fig. 3.53 shows the experimental setup and visualizes the robot path in a satellite image.

For mapping, we chose the average angle error dissimilarity measure (ref. Eq. (3.4)) and a dissimilarity threshold of $\xi^{\text{map}}_{\delta^{\text{ang}}_{\text{diss}}} = 0.12\text{rad}$ (approx. 6.9°) for creating a new viewframe. The angle threshold for creating the Trail-Map was $\delta_{\text{ang}} = 10°$. With these parameters, the robot recorded 49 viewframes and 36024 landmark views in total. The resulting Trail-Map had 31323 landmark views spread over 17 levels, which means memory savings of 15% compared to storing all landmark views. Again, the majority of landmark views (88%) was in level 1 of the Trail-Map. Although each viewframe had 735 landmark views on average, only a mean of 122 of them could be matched between successive viewframes. Thus, all the unmatched landmarks remained in level 1 of the map, additionally to the landmark views that changed their bearing by more than δ_{ang} compared to the last viewframe. Statistics of the mapping process are summarized in Table 3.7. The mapping trajectory is shown in Fig. 3.54.

Table 3.7.: Mapping statistics of the carpark experiment

Trail-Map angle threshold δ_{ang}	10°
mapping dissimilarity threshold $\xi^{\text{map}}_{\delta^{\text{ang}}_{\text{diss}}}$	0.12rad ≈ 6.9°
path length	98.7m
observed LVs	36024
recorded viewframes	49
levels in Trail-Map	17
LVs in in full Trail-Map	31323
avg. LVs per viewframe	735 ± 25.5
avg. matches between viewframes	122 ± 24.7
LVs in Level 1	27681 (88.4%)
LVs in Level 2	1967 (6.3%)
LVs in Level 3	720 (2.3%)
LVs in Level 4	383 (1.2%)
LVs in Level 5	253 (0.8%)

For homing, we pruned the Trail-Map – except for the home viewframe – by 2 levels, which proved to be a good trade-off between memory consumption and path accuracy in the indoor experiments. As a result 2355 landmark views remained in the Trail-Map, which corresponds to approximately 250kB of memory. On average, each viewframe had 155 remaining landmark views after pruning. The dissimilarity measure for detecting a viewframe as reached was set to $\xi^{\text{hom}}_{\delta^{\text{ang}}_{\text{diss}}} = 0.05\text{rad}$ average angle error (approx. 2.9°) and the robot computed homing vectors using the normalized improved difference vector model in Eq. (3.9), which proved to be very robust against outliers in the simulations (ref. Section 3.7.2). The hom-

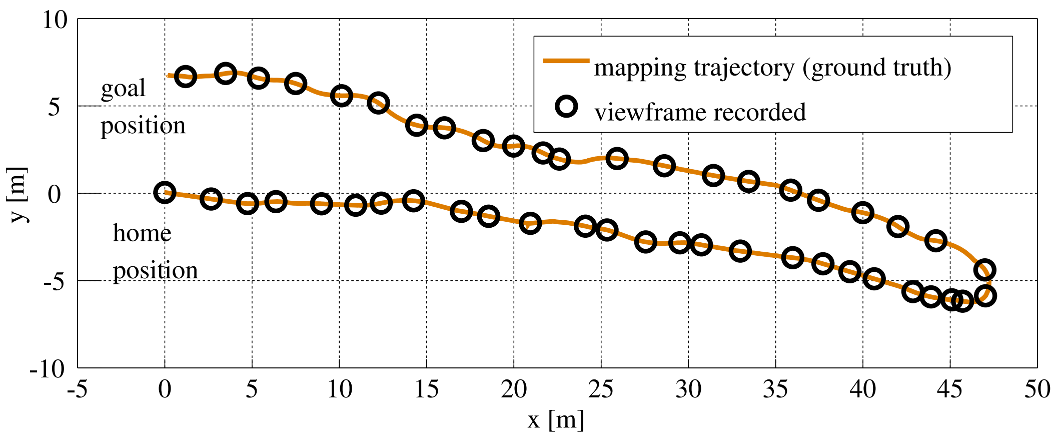


Figure 3.54.: Mapping trajectory for the outdoor carpark experiment

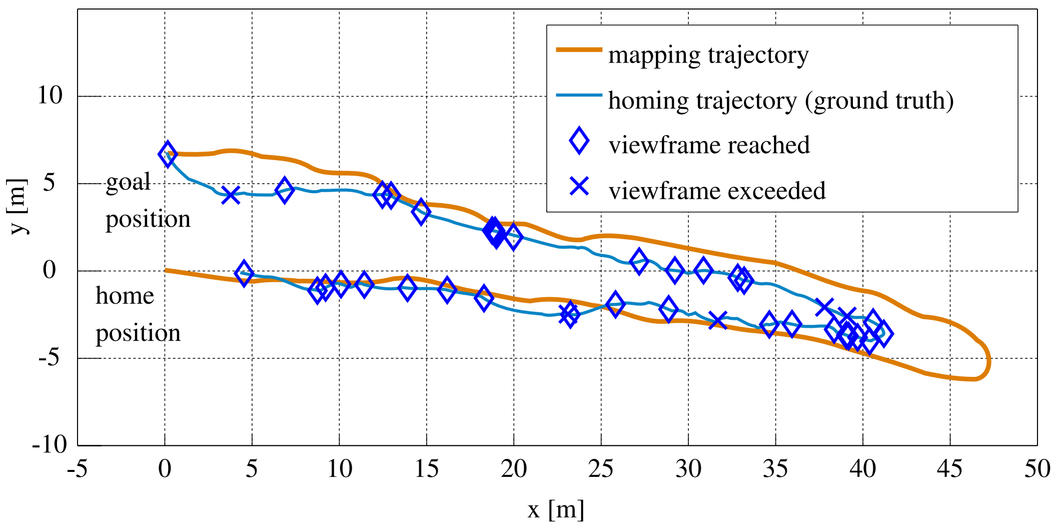


Figure 3.55.: Homing trajectory for the Trail-Map pruned by 2 levels in the outdoor carpark experiment

ing process took 14.4 minutes and the robot could drive at a maximal velocity of 0.2m/sec. The robot's homing trajectory is shown in Fig. 3.55. This figure shows the locations where the robot assumed to have reached or exceeded a viewframe. As described above, the robot assumes to have exceeded a viewframe when the homing vector jumps by more than 90°. As the figure shows, the robot performed homing successfully, but took a short-cut near the curve. That could be caused by the bigger size of the viewframes after pruning two levels of the map. We observed a similar behavior in our indoor experiments (ref. Section 3.10.1). During homing, the robot could match 20 landmark views on average, which is only 12.9% of the available landmark views in each viewframe. The robot's homing trajectory had an average path error of 0.88m±0.60m and a maximal deviation from the mapping path of 2.5m.

The endpoint error when the robot assumed to have reached the home position was 4.33m. This large deviation from the home position could be caused by the lack of nearby landmarks. The homing statistics are summarized in Table 3.8.

Table 3.8.: Homing statistics of the carpark experiment

homing dissimilarity threshold $\xi^{\text{hom}}_{\delta^{\text{ang}}_{\text{diss}}}$	0.05rad $\approx 2.9°$
pruned levels	2
remaining LVs in Trail-Map	2355
memory requirements	$\approx$ 250kB
average number of LVs per viewframe	155 ± 29.0
average number of matched LVs per viewframe	20 ± 6.4 (12.9%)
average path deviation	0.88m±0.60m
maximal path deviation	2.5m
endpoint error	4.33m

This experiment shows that the robot is able to retrace a path of nearly 100m length with a Trail-Map size of about 250kB. However, the robot moved through a bounded environment and was theoretically able to detect the majority of landmarks it observed at the home position along the whole path.

Urban Experiment

In another experiment, we remotely controlled the robot along a curved path of 87.4m length on a road between buildings, so that the landmarks which were visible from the home position could not be observed from the goal position. Fig. 3.56 shows the commanded path in a satellite image. The robot used the same mapping parameters as in the previous experiment ($\xi^{\text{map}}_{\delta^{\text{ang}}_{\text{diss}}} = 0.12$rad average angle error and $\delta_{\text{ang}} = 10°$) and recorded 49 viewframes with an average of 601 landmark views per viewframe. The resulting Trail-Map had 16 levels and 25130 landmark views. Again, 87% of all landmark views were in level 1 of the Trail-Map, and only an average of 107 landmarks could be matched between successive viewframes. Table 3.9 shows the statistics of the urban mapping results. The mapping trajectory with the viewframes is shown in Fig. 3.57. As can be seen, the viewframes were closer together near the curve in the area where the robot left the carpark, because in this region the landmarks were closer to the robot trajectory than in the rest of the environment.

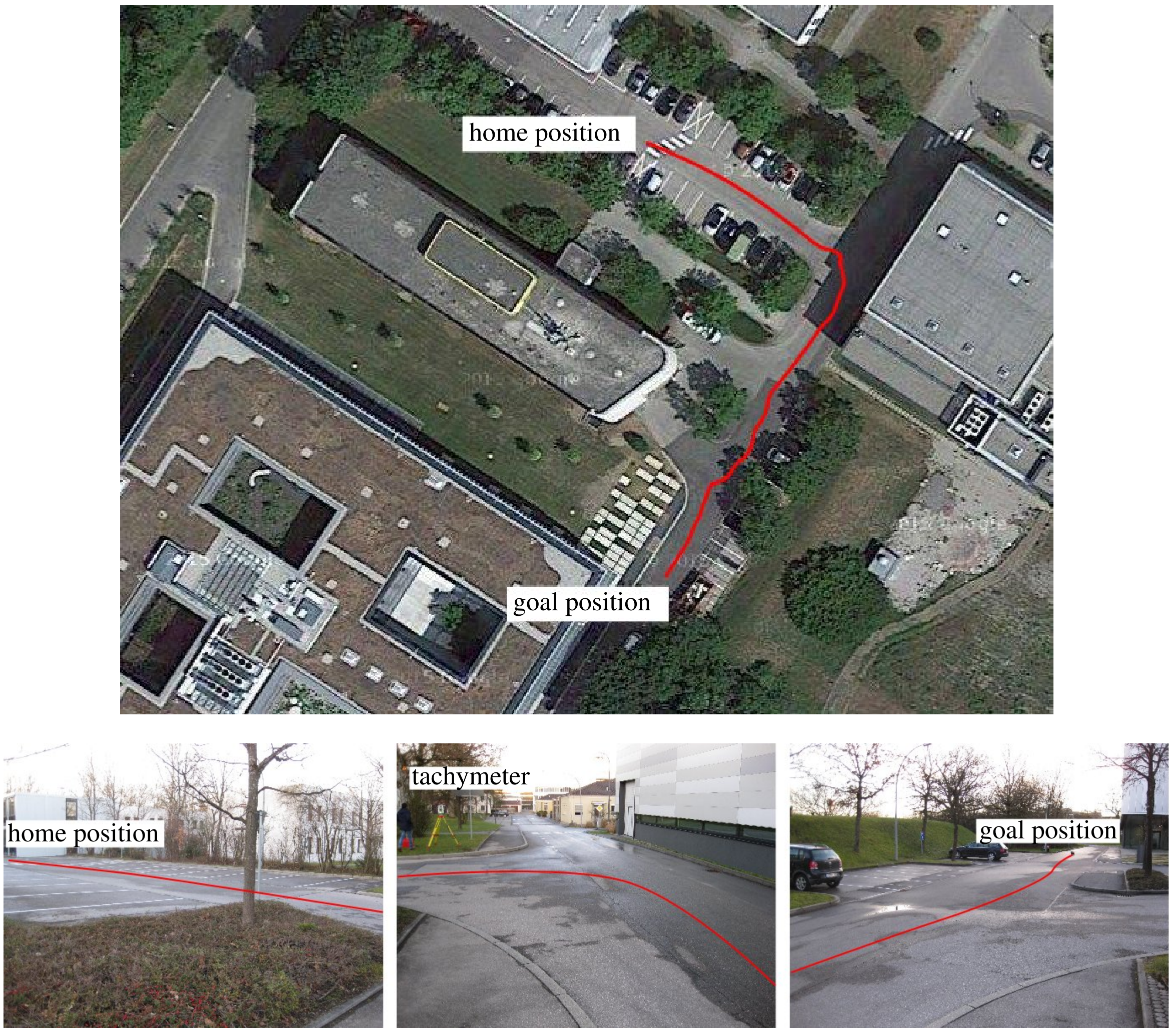


Figure 3.56.: Experimental setup and robot path for the outdoor urban experiments (satellite imagery ©2015, DigitalGlobe, GeoBasis-DE/BKG)

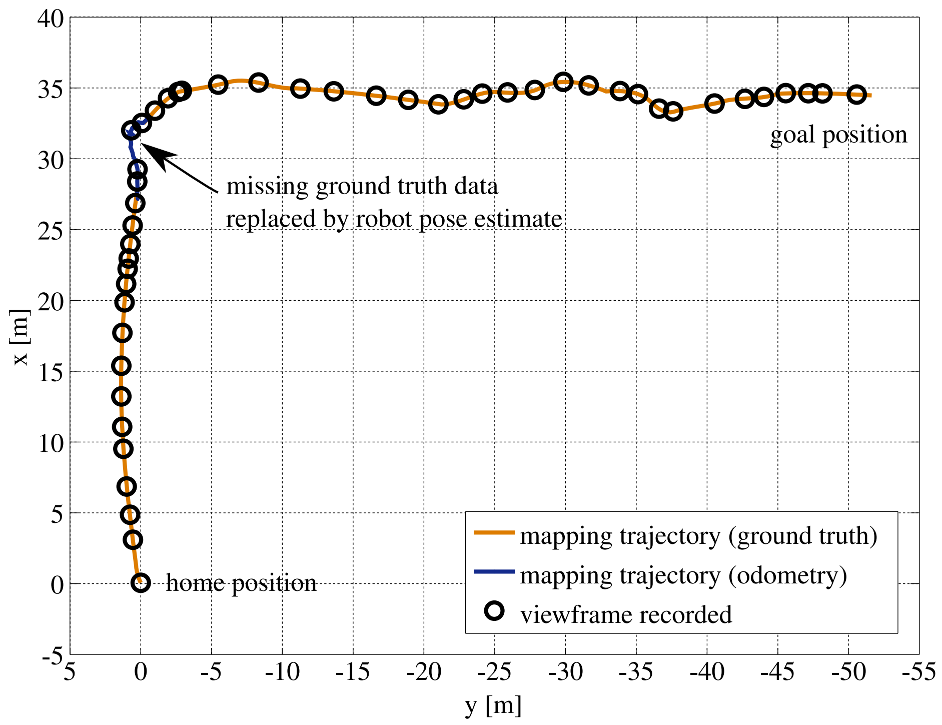


Figure 3.57.: Mapping trajectory for the outdoor urban experiment

Table 3.9.: Mapping statistics of the urban experiments

Trail-Map angle threshold δ_{ang}	10°
mapping dissimilarity threshold $\xi^{\text{map}}_{\delta^{\text{ang}}_{\text{diss}}}$	0.12rad ≈ 6.9°
path length	87.4m
observed LVs	29437
recorded viewframes	49
levels in Trail-Map	16
LVs in in full Trail-Map	25130
avg. LVs per viewframe	601 ± 82.2
avg. matches between viewframes	107 ± 29.5
LVs in Level 1	21845 (86.9%)
LVs in Level 2	1780 (7.1%)
LVs in Level 3	725 (2.9%)
LVs in Level 4	301 (1.2%)
LVs in Level 5	189 (0.8%)

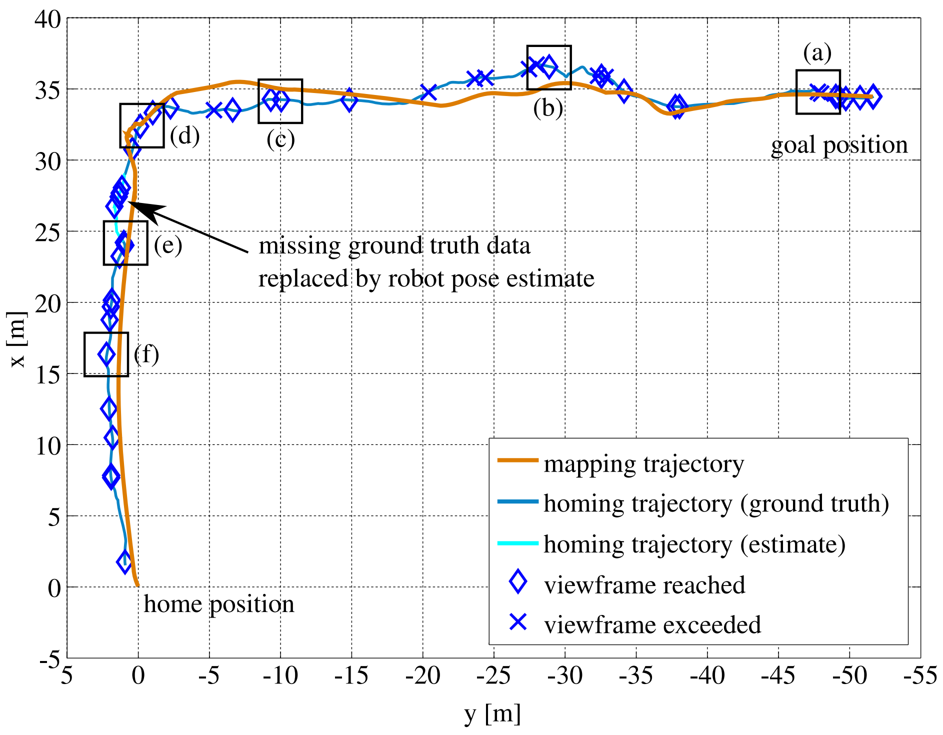


Figure 3.58.: Homing trajectory for the Trail-Map pruned by 2 levels (2195 LVs) with marked image locations for Fig. 3.59 in the outdoor urban experiment

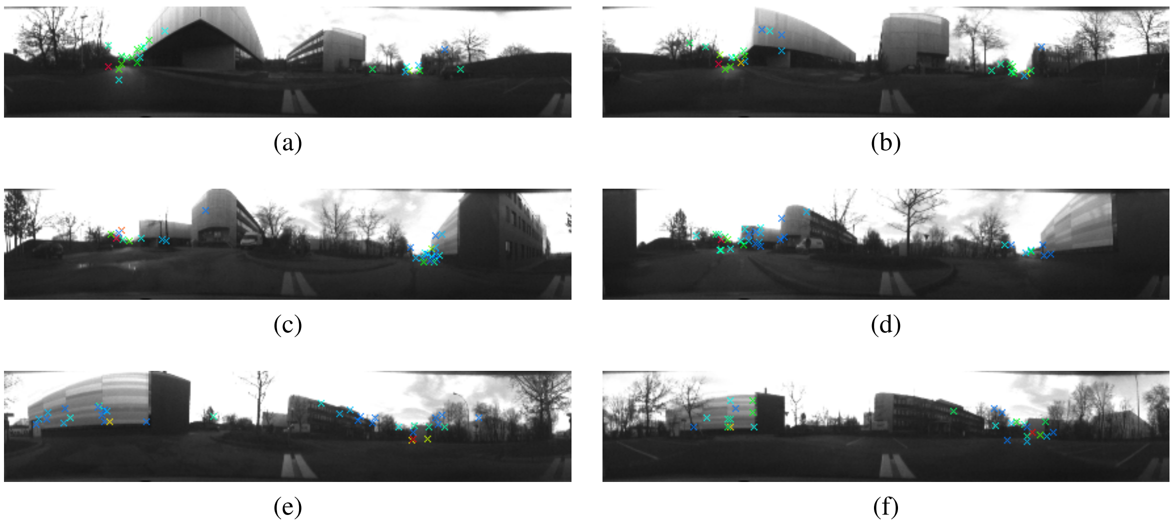


Figure 3.59.: Sample panorama images with matched landmarks during homing in the urban experiment with the Trail-Map pruned by 2 levels for robot positions shown in Fig. 3.58. Color code: blue-green-yellow-red corresponds to landmarks in levels 3-16 in this order.

For homing, we first pruned the Trail-Map by 2 levels but did not prune the home viewframe, so that the resulting Trail-Map had 2195 landmark views. The homing parameters were also equal to the carpark experiment ($\xi^{\text{hom}}_{\delta^{\text{hom}}_{\text{diss}}} = 0.05$rad average angle error). The homing vectors were computed using the normalized improved difference vector model in Eq. (3.9) at a rate of about 3.4Hz parallel to the pose estimation process on the Pioneer's Intel Core i7 CPU with 2.7GHz. The homing trajectory is shown in Fig. 3.58. We had a tracking drop out of a few meters, when the tachymeter lost the connection to the prism. We filled the gap in the plot with the pose estimate that the robot obtained by fusing IMU data and visual odometry. The robot successfully retraced the path with an average path error of 0.77 ± 0.51m, a maximal path error of 2.3m and an endpoint error of 1.74m. On average, the robot could match 22 landmark views in each image with the landmark views in the corresponding goal viewframe. Fig. 3.59 presents some sample images of the homing run with color-coded landmark matches. It can be seen that after pruning 2 levels mainly landmarks in the direction of motion were left. Furthermore, the figure shows that the red landmarks, which are in the highest levels of the Trail-Map, could be observed over a large range of viewframes.

To evaluate if homing is still possible with even less memory, we pruned the same Trail-Map by 3 levels, resulting in 1484 remaining landmark views. The robot performed homing for nearly 60 meters but then failed to compute stable homing vectors. On average, the robot matched only 17 landmarks in each image with the corresponding goal viewframe. The homing vectors started jumping, so that the robot assumed it had exceeded the viewframes and started homing to the next one. In the end, the current goal viewframe was too far away from the robot's current position, so that not enough landmark matches could be found. We stopped the homing process at this point. Fig. 3.60 shows the robot trajectory. Table 3.10 summarizes the statistics for the two homing experiments.

Table 3.10.: Homing statistics of the urban experiments

homing dissimilarity threshold $\xi^{\text{hom}}_{\delta^{\text{ang}}_{\text{diss}}}$	0.05rad ≈ 2.9°	0.05rad ≈ 2.9°
pruned levels	2	3
remaining LVs in Trail-Map	2195	1484
memory requirements	≈ 220kB	≈ 150kB
avg. number of LVs per viewframe	137 ± 22.7	116 ± 29.7
avg. matched LVs per viewframe	22 ± 7.1 (16.0%)	17 ± 7.2 (14.7%)
avg. path deviation	0.77 ± 0.51m	homing failed
maximal path deviation	2.3m	homing failed
endpoint error	1.74m	homing failed

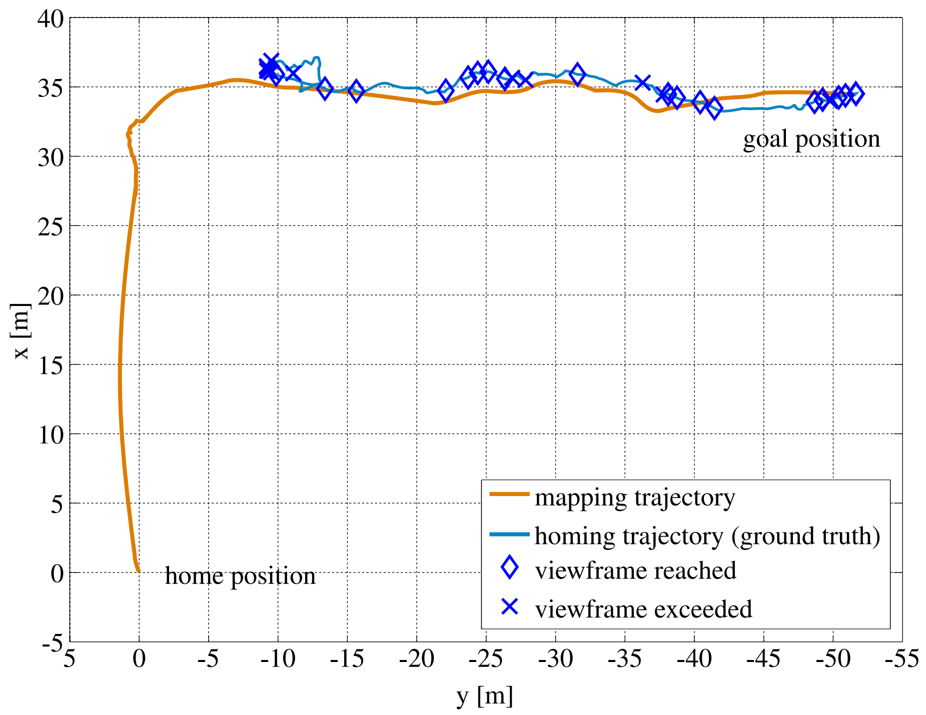


Figure 3.60.: Homing trajectory for the Trail-Map pruned by 3 levels (1484 LVs) in the outdoor urban experiment. In this experiment, homing failed because the number of landmark matches was not sufficient to compute stable homing vectors.

These experiments show that a Trail-Map size of about 220kB for 87m length is sufficient for the robot to retrace a path through a segmented environment, in which the visibility of landmarks is restricted to certain parts of the trajectory. Thus, we can assume that these memory requirements scale approximately linearly with the length of the traversed path, even for longer traverses of more than a kilometer in length, for example. The memory requirements in this experiment also correspond to about 250kB for 100m as in the previous experiment in the bounded carpark environment. However, stronger pruning of the Trail-Map can lead to a failure of the homing process.

Open Terrain Experiment

In a final outdoor experiment, we remotely controlled the robot along a path of about 86.3m length on a road through wide open terrain. The environment and the path are shown in Fig. 3.61. The robot recorded a Trail-Map using an angle threshold of $\delta_{\text{ang}} = 10°$ and a viewframe dissimilarity threshold of $\xi^{\text{map}}_{\delta^{\text{map}}_{\text{diss}}} = 0.15\text{rad}$ (approx. 8.6°) average angle error.

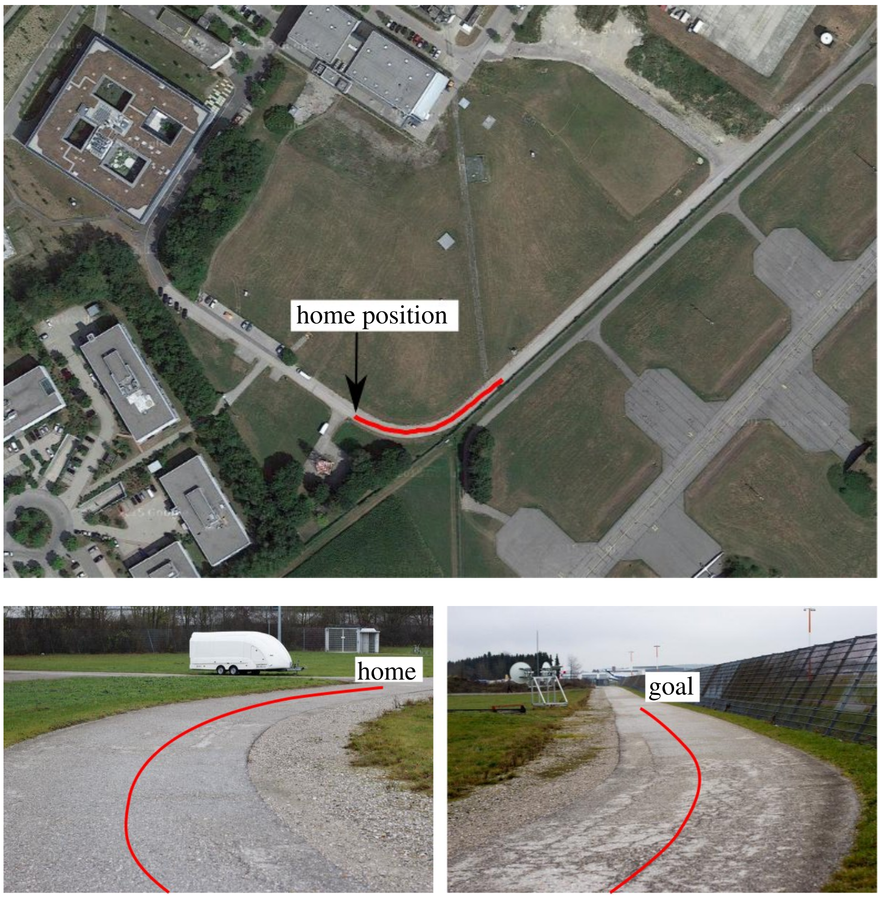


Figure 3.61.: Open terrain environment and robot path (satellite imagery ©2015, DigitalGlobe, GeoBasis-DE/BKG/GeoContent)

The ground truth robot trajectory is given in Fig. 3.62. The robot recorded 61 viewframes with a mean of 412 landmark views per viewframe, where 113 landmark views could be matched between successive viewframes on average. The full Trail-Map had 20953 landmark views spread over 12 levels, which saved about 17% compared to storing all landmark views of all viewframes. Again, the majority of landmark views (more than 83%) was in level 1 of the map, which can be seen in Table 3.11.

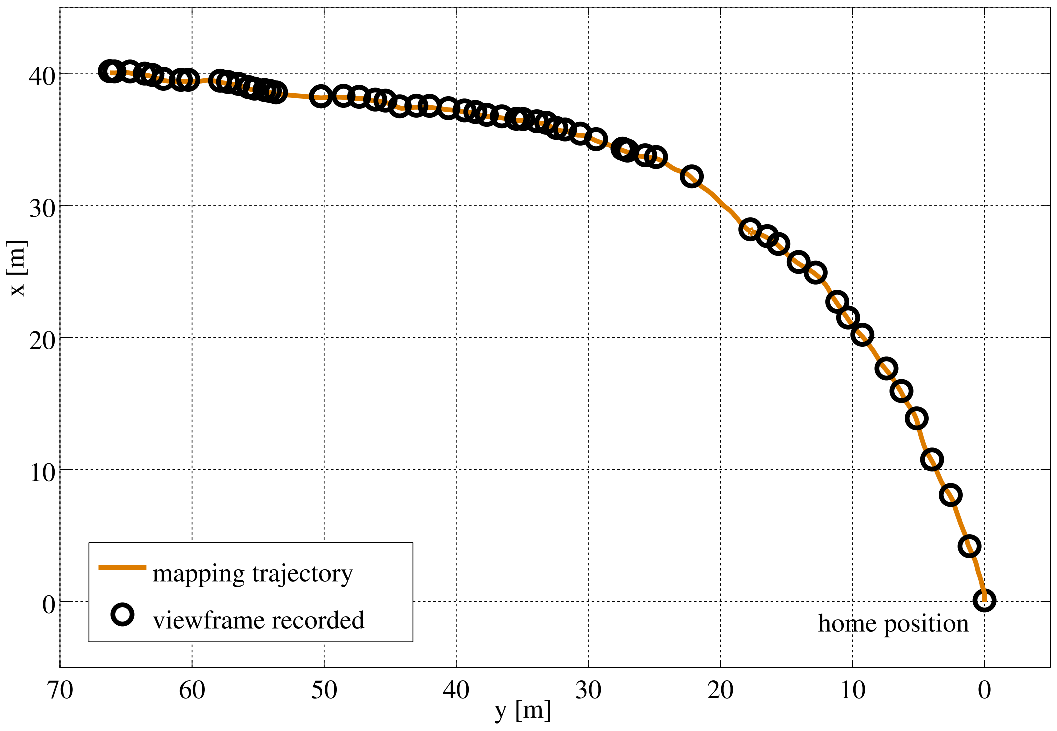


Figure 3.62.: Mapping trajectory for the open terrain experiment

Table 3.11.: Mapping statistics of the open terrain experiment

Trail-Map angle threshold δ_{ang}	10°
mapping dissimilarity threshold $\xi^{\text{map}}_{\delta^{\text{ang}}_{\text{diss}}}$	0.15rad ≈ 8.6°
path length	86.3m
observed LVs	25126
recorded viewframes	61
levels in Trail-Map	12
LVs in in full Trail-Map	20953
avg. LVs per viewframe	412 ± 39.0
avg. matches between viewframes	113 ± 28.8
LVs in Level 1	17536 (83.6%)
LVs in Level 2	1316 (6.3%)
LVs in Level 3	839 (4.0%)
LVs in Level 4	656 (3.1%)
LVs in Level 5	287 (1.4%)

After mapping, we pruned the recorded Trail-Map by 2 levels, which resulted in 2544 remaining landmark views. That corresponds to approximately 250kB of memory and 12% of the full Trail-Map size. The robot was placed near the end of the mapping trajectory and performed homing using the pruned Trail-Map. For this, the robot used a dissimilarity threshold of $\xi^{\text{hom}}_{\delta^{\text{hom}}_{\text{diss}}} = 0.05$rad average angle error and calculated homing vectors using the normalized improved difference vector model in Eq. (3.9). The resulting homing trajectory is visualized in Fig. 3.63. As the figure shows, the robot exceeded many viewframes in this experiment, because the environment was short of distinctive landmarks and the lighting conditions were challenging (ref. Fig. 3.64 for sample panorama images of the homing run). Therefore, there were many false landmark matches that affected the homing vector. Nevertheless, the robot was still able to retrace most of the learned path. Since the robot did not perform obstacle avoidance, we had to intervene twice to bring the robot back on the road when it left the paved area. These sections are marked in the plot. The robot stopped 13.6m away from the home position while we were moving it back to the road. During this operation, we occluded large parts of the camera image, so that only 5 matching landmarks were visible and the resulting dissimilarity measure for reaching the home viewframe dropped below the threshold.

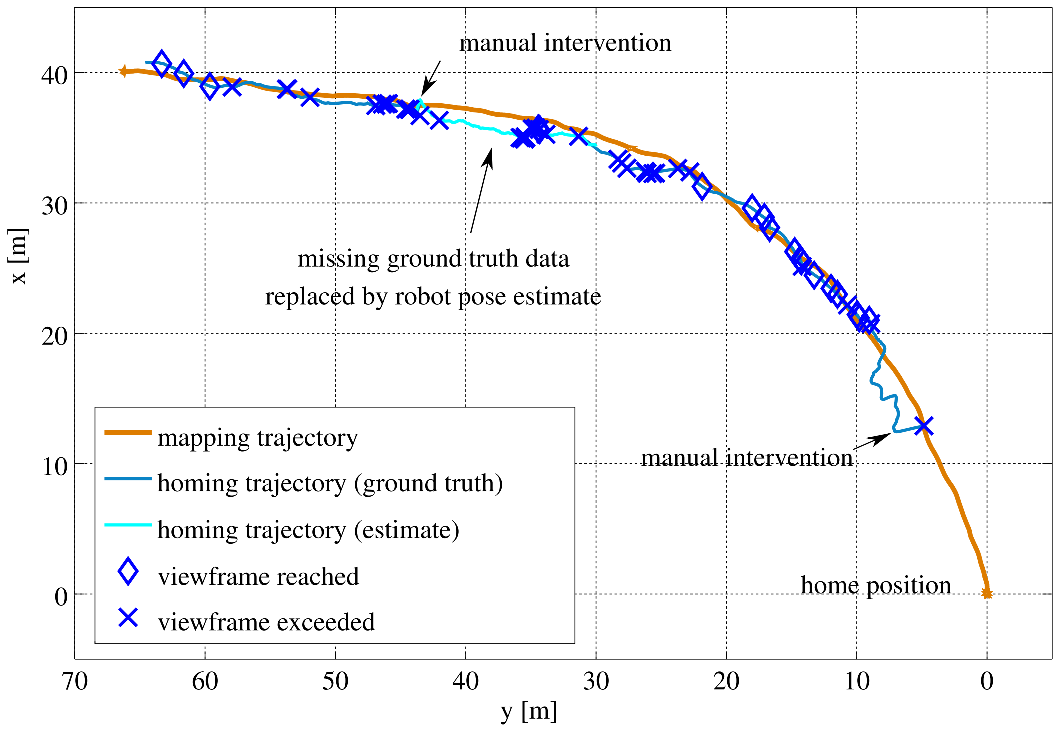


Figure 3.63.: Homing trajectory using the Trail-Map pruned by 2 levels for the open terrain experiment

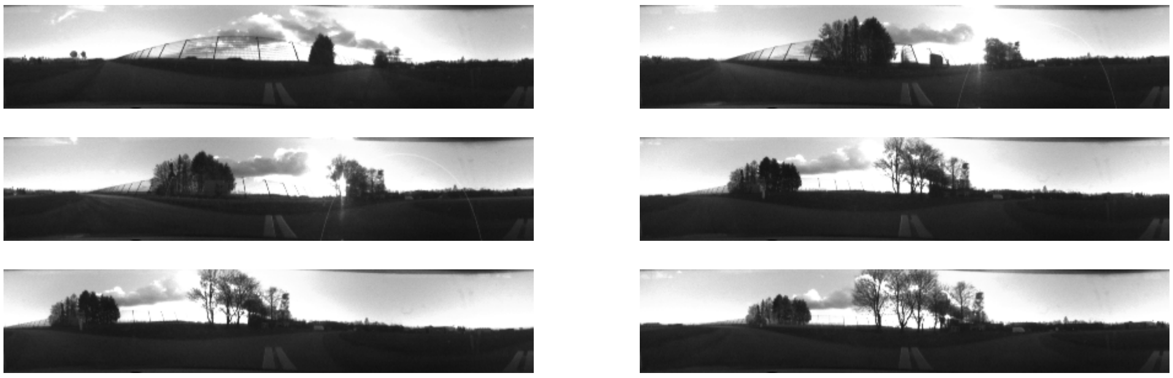


Figure 3.64.: Sample panorama images during homing in the open terrain experiment

To have a closer look at the landmark matching results, we evaluated the number of matched landmarks during homing compared to the number of landmark views and their levels in the corresponding viewframe. Fig. 3.65 shows the average distribution of the landmark views in a viewframe over the levels of the Trail-Map. As this figure shows, only a small fraction of the landmark views in each viewframe can be matched during homing, while in general the matching ratio is higher in higher levels of the Trail-Map. Compared to the landmark matching ratios of the indoor experiments, even less landmarks could be matched in this open terrain experiment.

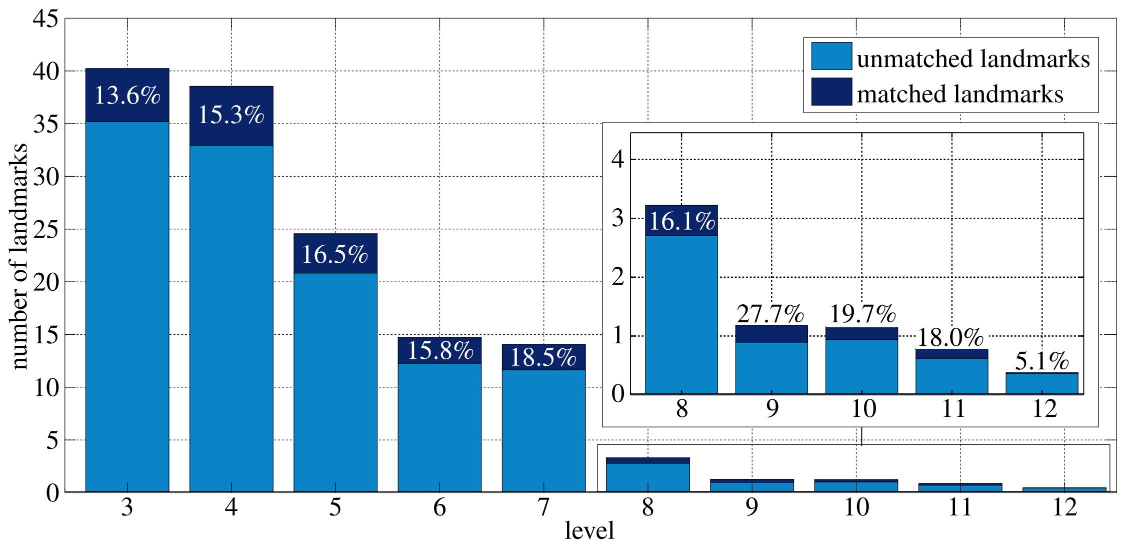


Figure 3.65.: Average landmark distribution and matches per viewframe in the open terrain experiment

The outdoor experiments have shown that a robust landmark matching method is the key component for robust homing using the Trail-Map. Nevertheless, we could demonstrate in different environments that successful homing is possible with less than 300kB of memory for a path of 100 meters length. Extrapolating the memory requirements for longer traverses results in an average memory requirement of about 3MB per kilometer. Compared to the Visual Teach and Repeat approach described by Furgale and Barfoot [27], which is based on overlapping metric submaps and requires about 348MB of data per kilometer, the Trail-Map-based homing approach requires two orders of magnitude less memory, while the computational complexity is constant during mapping and homing. The Trail-Map also significantly outperforms the method proposed by Krajník et al. [51], who reported memory requirements of 848MB for a path of 8km (ref. Table 3.12).

Table 3.12.: Comparison of memory requirements for different homing methods

method	memory requirements	memory requirements per 1km
Visual Teach and Repeat [27]	348MB per 1km	348MB
method by Krajník et al. [51]	848MB per 8km	106MB
Trail-Map-based homing	300kB per 100m	3MB

3.11. Conclusion

In this chapter we described a landmark-based navigation method for autonomous homing, which is solely based on landmark bearing angles and does not use metric distance information. Landmark observations and their angular configurations are stored in so-called viewframes at certain locations along the robot's path. The acquisition of a new viewframe is triggered when the dissimilarity to the previous recorded viewframe exceeds a threshold. For this, we analyzed different viewframe dissimilarity measures and discussed their behavior under the presence of observation errors. The robot can retrace a path by computing homing vectors, which point in the direction to align the robot's current landmark configuration with the one stored in the goal viewframe. In this chapter, we gave an overview of several existing homing vector computation methods and introduced new methods, which outperform the existing ones in case of landmark outliers and nonisotropic landmark distributions. Next, we analyzed the LT-Map, which was the first data structure that provided an efficient and scalable storage of viewframe-based paths. After pointing out some weaknesses of the LT-Map, we developed the Trail-Map, which is a novel data structure for scalable and nonredundant viewframe storage that also allows constant-time mapping and homing. It is based on the insight that the bearing angles to distant landmarks and landmarks in the direction of travel hardly change during motion (translation invariant landmarks), while the bearing angles of nearby landmarks change significantly as the robot passes them (translation variant landmarks). Thus, the Trail-Map stores the landmark observations in a hierarchical order of their level of translation invariance and avoids redundant storage of landmark observations that do not change significantly between subsequent viewframes. In case of memory shortage, the Trail-Map can be pruned by deleting the landmark views in the lower levels of the map. By doing so, only the information of quickly changing and possibly unstable landmarks is discarded, and the information of stable, translation invariant landmarks is preserved. In simulations we could show the superior performance of the Trail-Map compared to the LT-Map. Furthermore, we analyzed the Trail-Map data structure in the presence of noise. We could validate the approach in indoor and outdoor experiments with a robot using natural landmarks extracted from omnidirectional camera images, where we could show the scalability and memory efficiency of the method. In particular, we demonstrated in long-range outdoor experiments that a Trail-Map pruned by 2 levels with less than 300kB is sufficient to store and retrace a path of 100 meters length, which outperforms state-of-the-art methods by two orders of magnitude, while still offering constant runtimes for mapping and homing. Since the method achieved these values also in a segmented environment, in which the home and at the goal location had no common landmarks, we can assume that the memory requirements can be approximately linearly extrapolated for longer traverses of more than a kilometer in length, for example. Due to these properties, we also claim that a Trail-Map-based method is a promising alternative to state-of-the-art feature-based SLAM approaches for the task of robot homing without loop closures in the mapping path.

The bottleneck of the whole method is the landmark detection and matching process, because Trail-Map-based homing heavily depends on correct and reliable landmark matches. Unstable landmarks unnecessarily inflate the Trail-Map, and false matches affect the correctness of the homing vector. Furthermore, to make Trail-Map-based homing applicable to real-world scenarios, it must be combined with a local obstacle detection and avoidance method. This problem is tackled in the following chapter.

4. Hybrid Navigation

In the previous chapters we developed a method for local metric navigation based on stereo disparity images, and a method for global topological homing based on landmarks extracted from an omnidirectional camera. For homing in unstructured environments, the robot needs a combination of both skills to return to the home position by moving safely through the environment. Blindly following the homing vector computed from the global navigation method could lead the robot into obstacles, because the homing path deviates from the learned path depending on the landmark distribution in the environment and on the pruning level of the Trail-Map. Although the robot most likely observed the obstacles during the first traversal of the path, there is no benefit in storing this information for the homing process. First of all, this would require additional memory. Second, the robot would need an accurate metric localization method to align the currently perceived terrain patch with the stored obstacle. Furthermore, the robot would have to fuse the old obstacle information and the new observations to have a consistent obstacle representation. However, metric localization and fusion of environment information would decrease the efficiency of the homing method. In fact, storing the obstacle positions during the path learning phase is simply not necessary because the robot would observe and detect the obstacles during homing anyway. By doing so, the robot can even avoid obstacles which were not present during the path learning phase, in case these obstacles do not block the learned path completely.

For obstacle avoidance during homing, the robot maintains a geometric representation of its immediate surroundings, which allows estimating the traversability of the terrain. Using this local metric map, the robot can choose the homing direction that is close to the computed homing vector but does not put the robot to any risk.

An overview of the hybrid navigation method is presented in Fig. 4.1. The local and global navigation components have already been described as separate modules in the previous chapters. The homing vector calculated by the global navigation method is given as input for the local path planner, which uses the local traversability cost map to adjust the homing direction to only lead through safe cells. This chapter explains what is necessary to combine the local metric navigation approach with the global homing method and experimentally demonstrates the performance of the hybrid homing method.

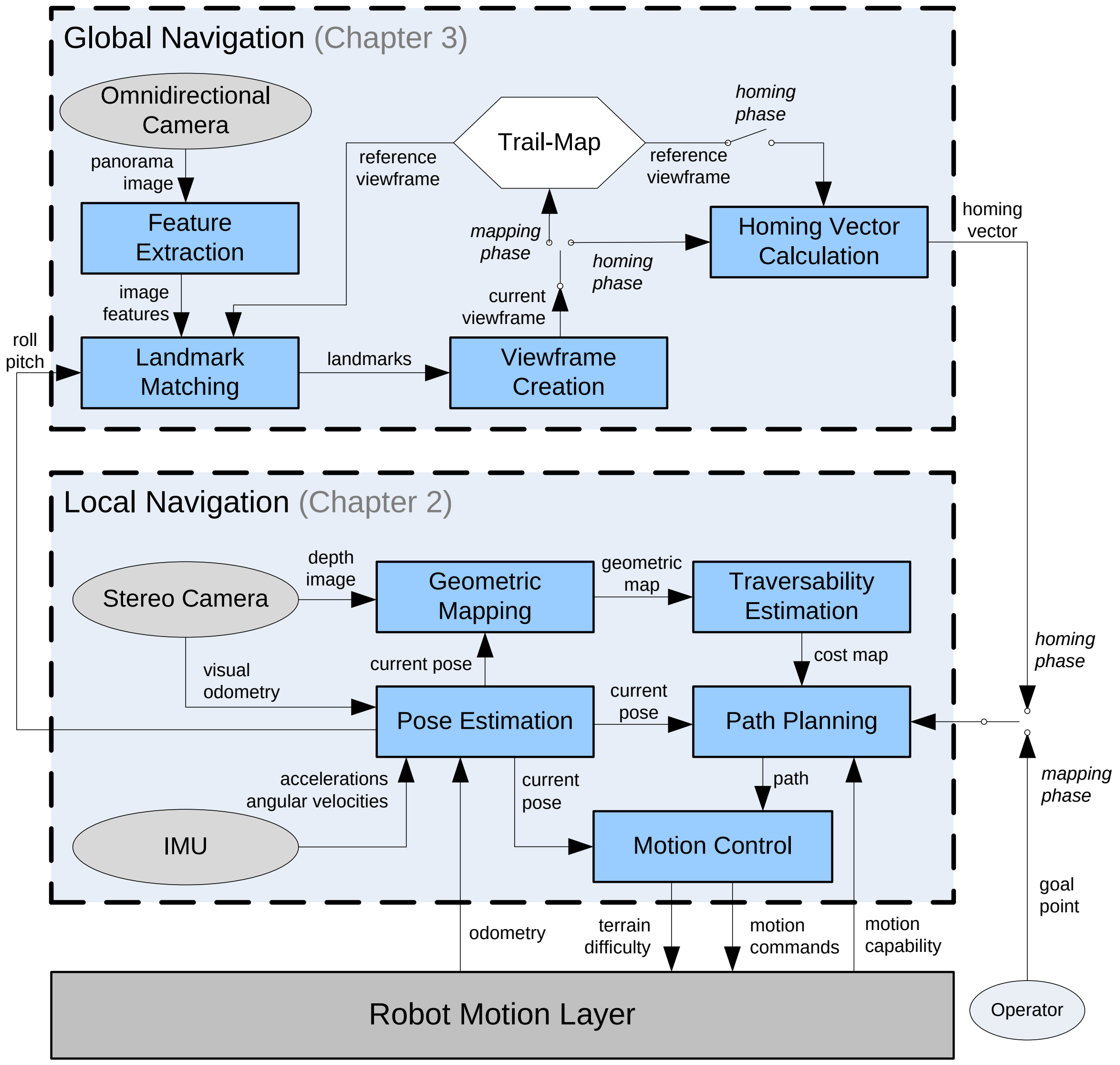


Figure 4.1.: Overview of the hybrid navigation method

4.1. Creation of a Moving Local Metric Map

The robot needs detailed information about the structure of the environment only in its immediate surroundings to find traversable regions and obstacles. Thus, we limit the size of the local map and let the map move with the robot. For this, the map origin is shifted every time a new single-view map is attached, so that the robot always stays in the center of the local map. Terrain information outside the map bounds is deleted. This ensures constant memory requirements and also constant runtimes for creating and maintaining the local traversability cost map.

The size of the local map should be chosen according to the available computational power and according to the pose estimation and environment modeling uncertainties. The map should be big enough to allow meaningful obstacle detection, but small enough to prevent pose estimation errors from affecting the map quality.

4.2. Homing with Obstacle Avoidance

The homing vector calculated from the landmark correspondences between the viewframes should lead the robot along the learned path to its home position. Assuming a static environment and constant motion capabilities of the robot, this path is known to be safe, because it has been traversed by the robot earlier. However, small deviations from this trajectory can already lead into regions which are not safe for the robot. As the simulations and the indoor and outdoor experiments of Trail-Map-based homing have shown, such deviations occur depending on the landmark configuration of the environment and on the pruning level of the Trail-Map. To make sure that the robot performs homing along a safe path, the raw homing vector has to be adjusted considering the estimated traversability of the terrain in the local metric map. This could be achieved by using the same D* path planner as described in Section 2.4 and setting a goal point in the direction of the homing vector. For this, it must be ensured that the goal point is located in a safe cell. Otherwise the path planning would fail. The effort of finding a safe goal cell can be high, and since the path planner only has to work in a very small region, we chose a different and easier approach than D* path planning, which will be explained in the following paragraphs.

4.2.1. Adjusting the Homing Vector

The idea of realizing obstacle avoidance with homing vectors is to adjust the homing direction as little as possible with minimal robot turning while achieving lower costs for the intended direction of movement. For this, three measures are evaluated:

- **Homing direction costs:** A straight path of a fixed length l_{vec} is projected on the cost map along a possible homing direction ϕ_i and subsampled in equal distances. The costs of the underlying cells are then summed up and give the homing direction costs d_i. The lower the costs, the safer is the corresponding homing direction.
- **Path deviation penalty:** We introduced a penalty proportional to the angular deviation from the desired homing direction ϕ_h, which ensures that the new direction is as close as possible to the computed homing direction.
- **Robot turning penalty:** The turning penalty is proportional to the angular deviation from the current yaw angle of the robot α and ensures that the new direction is close to the current bearing of the robot.

Summing up these three measures gives the overall costs c_i for each possible homing direction ϕ_i:

$$c_i = d_i + |\phi_i - \phi_h| p_{\text{dev}} + |\phi_i - \alpha| p_{\text{turn}}, \tag{4.1}$$

where p_{dev} and p_{turn} are constant parameters for computing the path deviation penalty and the robot turning penalty, respectively. Algorithm 2 shows how the best homing direction is found. The robot starts computing the overall costs of the desired homing direction ϕ_h. Then it repeatedly changes the possible homing direction in fixed angular steps $\Delta\phi$ in either direction and recomputes the overall costs of these directions. The direction corresponding to the first local minimum of the overall costs is chosen by the robot. Additional parameters for finding the best homing direction are the size of the angular steps $\Delta\phi$, the length of the homing vector l_{vec} and the number of steps n_{samples} for subsampling this vector. The approach is visualized in Fig. 4.2.

4.2.2. Robot Motion Control

The motion control of the robot is similar to the method described in Section 3.9.5. The robot chooses a forward and a turn velocity that lead it in the direction of the adjusted homing vector. However, we enforce the robot to travel a distance of at least l_{vec} before it requests a new homing direction. This ensures that the robot does not get stuck during homing in case the computed homing vector alternates between two possible directions. Nevertheless, it constantly checks whether the intended direction of travel is safe using Algorithm 2.

Algorithm 2: Finding the best homing direction

```
Input  : robot pose $p_x, p_y, \alpha$; desired homing direction $\phi_h$; cost map $C$
Output: best homing direction $\phi_{\text{best}}$
$c_{\min} \leftarrow 10^5$
noMinCount$\leftarrow$ 1
for $i = 0$ to $180/\Delta\phi$ do
    if $i \mod 2 = 0$ then
        $\phi_i \leftarrow \phi_h + i\Delta\phi$                ▷ alternate between left and right direction
    else
        $\phi_i \leftarrow \phi_h - i\Delta\phi$
    $\Delta l \leftarrow l_{\text{vec}}/n_{\text{samples}}$                ▷ subsample homing direction
    $\Delta x \leftarrow \Delta l \cos(\phi_i)$
    $\Delta y \leftarrow \Delta l \sin(\phi_i)$
    $d_i \leftarrow 0$
    for $j = 1 : n_{\text{samples}}$ do
        $p_{x,j} \leftarrow p_x + j\Delta x$
        $p_{y,j} \leftarrow p_y + j\Delta y$
        $c_j \leftarrow C(p_{x,j}, p_{y,j})$
        if $c_j = UNKNOWN$ then
            $c_j \leftarrow 1$                ▷ set cost for unknown cells to 1
        else if $c_j = UNTRAVERSABLE$ then
            $c_j \leftarrow 10n_{\text{samples}}$                ▷ set high costs for untraversable cells
        $d_i \leftarrow d_i + c_j$
    $c_i \leftarrow d_i + |\phi_i - \phi_h|p_{\text{dev}} + |\phi_i - \alpha|p_{\text{turn}}$                ▷ compute overall cost with penalties
    if $c_i < c_{\min}$ then
        $c_{\min} \leftarrow c_i$
        $\phi_{\text{best}} \leftarrow \phi_i$
        noMinCount $\leftarrow$ 0
    else
        noMinCount++
    if noMinCount=5 & $c_{\min} < 1$ then
        break                ▷ end algorithm if local minimum is found
```

If the computed homing direction leads through cells with a low certainty, the robot has to collect more information about the surrounding terrain before moving in this direction. For this, it performs exploration turns as described in Section 2.5.

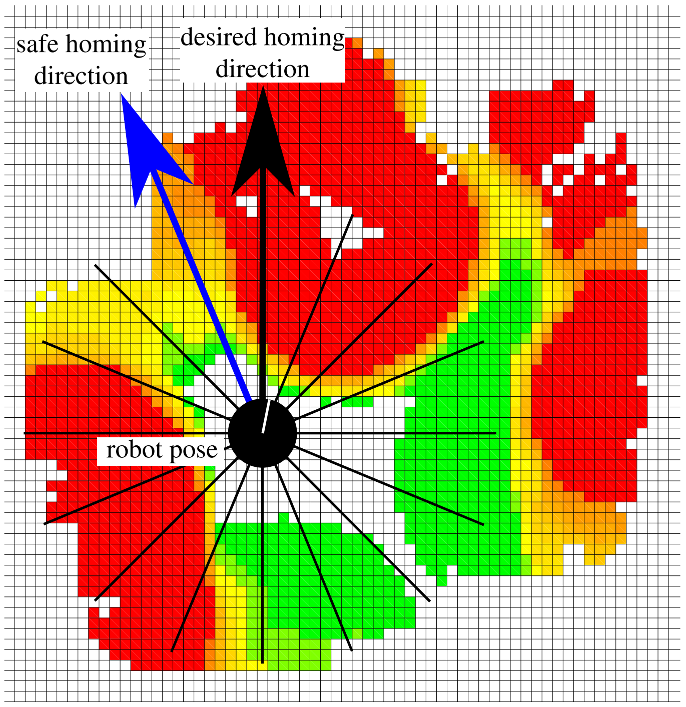


Figure 4.2.: Local adjustment of the homing vector: Cost map with traversability costs between 0 and 1 for traversable cells (white – green – orange) and > 1 for untraversable cells (red). The desired homing vector leads through untraversable cells and is adjusted for safe homing with minimal robot turning.

4.3. Experimental Evaluation

To demonstrate the combination of local obstacle avoidance and Trail-Map-based homing, we set up an indoor test environment with obstacles such as stones and slopes as shown in Fig. 4.3. We used the Pioneer 3-DX robot as mobile test platform, which is equipped with a wide angle stereo camera, an Xsens MTi10 IMU and an omnidirectional camera. The computational hardware is an Intel Core i7-3740QM CPU with 2.70GHz and a Spartan 6 LX75 FPGA Eval Board to perform dense stereo matching using SGM [39] at a rate of more than 10Hz. The stereo camera is a Guppy-Pro grayscale camera pair with a resolution of 1292×964 pixels and a baseline of 9cm. It is equipped with wide angle lenses resulting in a focal length of 635 pixels, which is equal to a horizontal opening angle of about $90°$. The omnidirectional camera is a catadioptric system with a Point Grey Chameleon USB 2.0 color camera. It has a vertical opening angle of $-30°$ to $40°$. We used an infrared tracking system which tracked a reflecting target object mounted on the robot for obtaining ground truth robot poses.

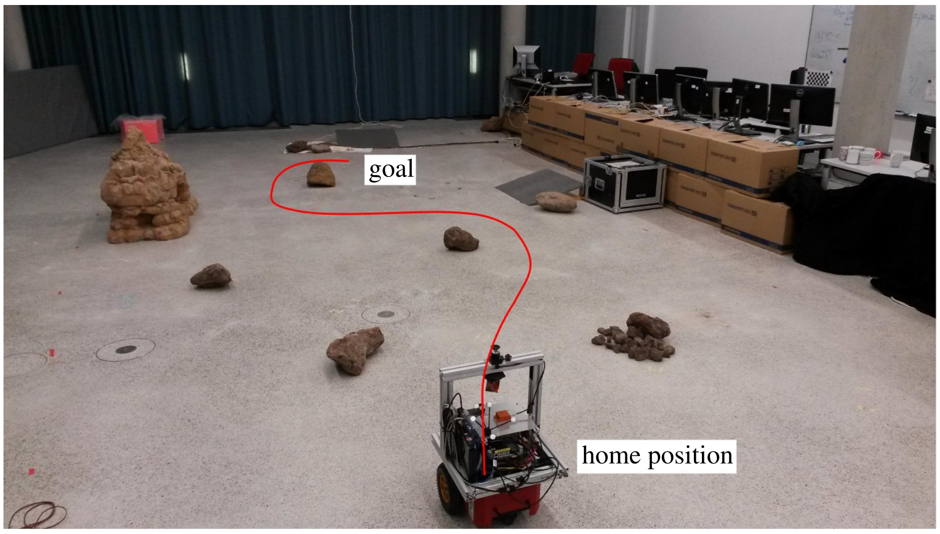


Figure 4.3.: Experimental setup for demonstrating homing with obstacle avoidance

4.3.1. Mapping

The robot was remotely controlled along a path of about 9.1m length, which is sketched in Fig. 4.3. During this traverse, the robot estimated its pose with respect to the starting position and maintained a moving local traversability cost map using the disparity images computed from the stereo camera data. However, since the robot was remotely controlled, it did not make use of the metric map, but only displayed it to the operator. Using the omnidirectional image data, the robot recorded viewframes using a dissimilarity threshold of $\xi^{\text{map}}_{\delta^{\text{ang}}_{\text{diss}}} = 0.05\text{rad}$ average angle error and built a Trail-Map with an angle threshold of $\delta_{\text{ang}} = 10°$. The resulting Trail-Map consisted of 39 viewframes and 12919 landmark views spread over 15 levels as given in Table 4.1. The robot detected on average 474 ± 42.2 landmarks in each image, and could match 157 ± 34.2 landmarks between subsequent viewframes. The mapping trajectory is shown in Fig. 4.4.

Table 4.1.: Distribution of landmark views (LVs) over the levels for the obstacle avoidance experiment

level	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LVs	9213	1548	817	491	349	196	139	80	34	17	20	6	6	1	2

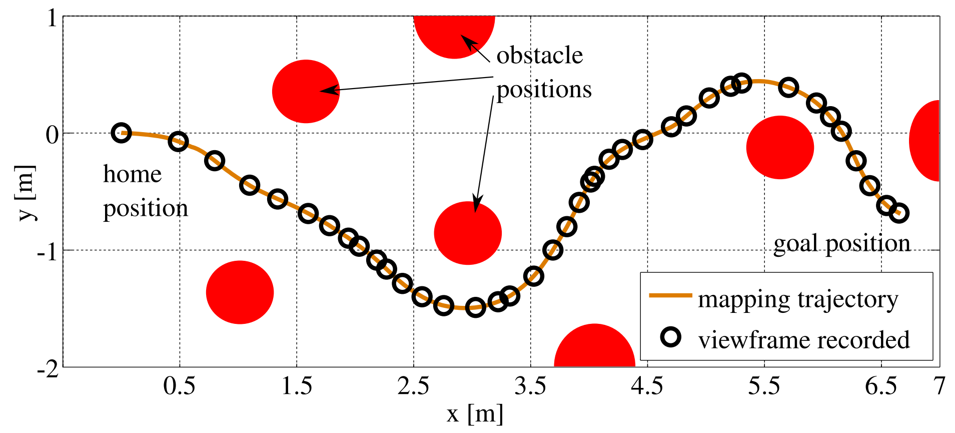


Figure 4.4.: Mapping trajectory with viewframes and obstacle positions

4.3.2. Homing with Obstacle Avoidance

To demonstrate homing with obstacle avoidance, the robot performed homing using the recorded Trail-Map as described in Chapter 3. It computed homing vectors using the normalized improved difference vector model and the dissimilarity threshold for detecting a viewframe as reached was set to $\xi^{\text{hom}}_{\delta^{\text{ang}}_{\text{diss}}} = 0.03\text{rad}$ average angle error. During homing, parallel to computing homing vectors from the observed landmark configurations, the robot maintained a local moving digital elevation map (DEM) and estimated the traversability of the terrain as described in Chapter 2. The parameters for the local mapping and traversability estimation process are given in Table 4.2. The resulting cost map was then used to correct the current homing direction as described in Section 4.2. The parameters for adjusting the homing direction are given in Table 4.3.

parameter	value
map resolution r	5cm
single view map size	1.2m×1.2m
local map size	2.5m×2.5m
robot diameter	1.0m
critical step height h_{crit}	5cm
critical slope s_{crit}	15°
critical roughness r_{crit}	7cm
slope weight α_1	0.25
roughness weight α_2	0.25
step height weight α_3	0.5

Table 4.2.: Cost map parameters

parameter	value
homing vector length l_{vec}	0.5m
homing vector samples n_{samples}	25
angular step size $\Delta\phi$	5°
path deviation penalty p_{dev}	0.2
robot turning penalty p_{turn}	0.2

Table 4.3.: Obstacle avoidance parameters

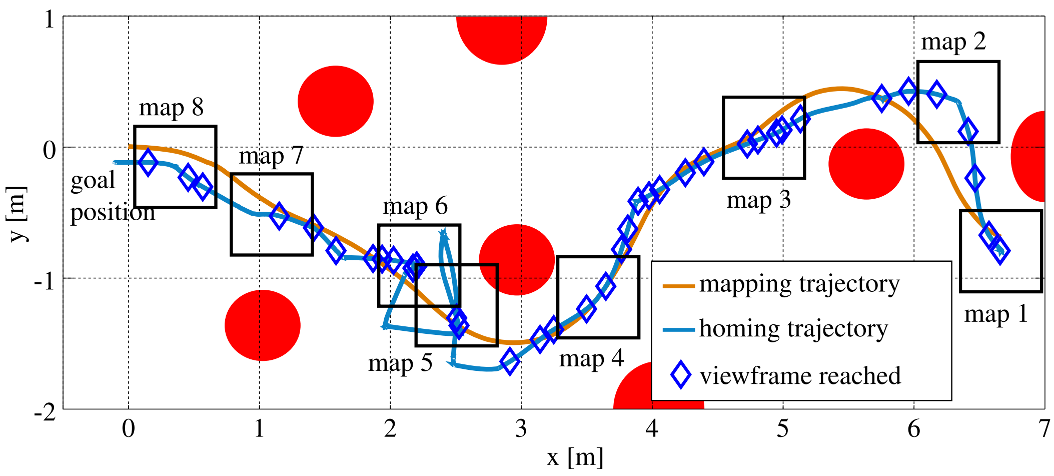
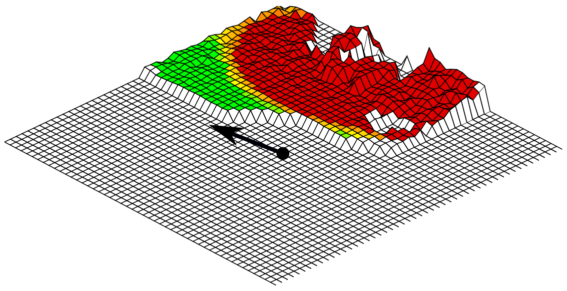
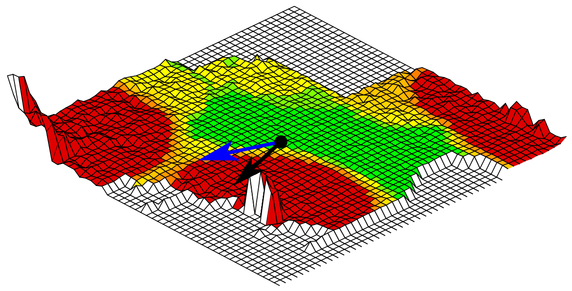


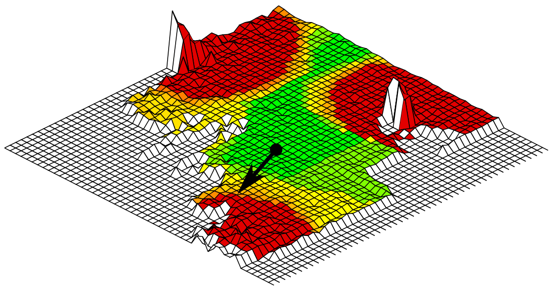
Figure 4.5.: Homing trajectory with obstacle avoidance using the full Trail-Map. The positions of selected local maps (ref. Fig. 4.6) are marked in the plot.

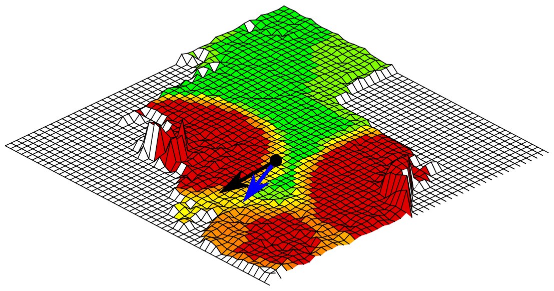
The robot performed several homing runs, each with a different pruning level of the Trail-Map. Fig. 4.5 shows the homing path of the robot when it used the full Trail-Map. As the figure shows, the robot made a detour after half of the homing trajectory. This detour was caused by noisy homing vectors in combination with the motion control method we implemented. Since the robot has to travel at least a distance of l_{vec} before it requests a new homing direction, a wrong homing vector can cause the robot to significantly deviate from its path. Some selected local maps of the homing run are shown in Fig. 4.6 along with the homing vector computed from the landmark configurations and the adjusted homing vector in case the desired direction was not safe for the robot. The plots show that the homing vector was always corrected if necessary, so that it led the robot through safe cells. Mapping and homing ran at a frequency of about 2Hz on the Pioneer's Intel Core i7 CPU with 2.7GHz. The computation time did not increase with the length of the path. The robot moved with a maximum velocity of 0.15m/sec.

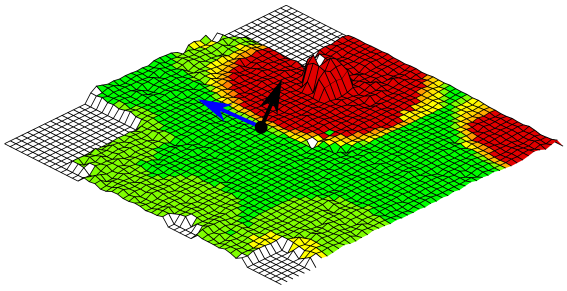
The homing trajectories of all homing runs with Trail-Map pruning levels from 0 to 8 are shown in Fig. 4.7. In all runs up to pruning level 7, the robot reached the home position without putting itself to any risk. We never had to intervene to prevent the robot from running into an obstacle. Only in the homing run using the Trail-Map pruned at 8 levels we stopped the homing process when the robot left the area covered by the tracking system. As the figure also shows, when the Trail-Map was pruned by 7 levels and only 735 LVs were left, the robot passed one obstacle on the other side than in the learning run. This shows that the

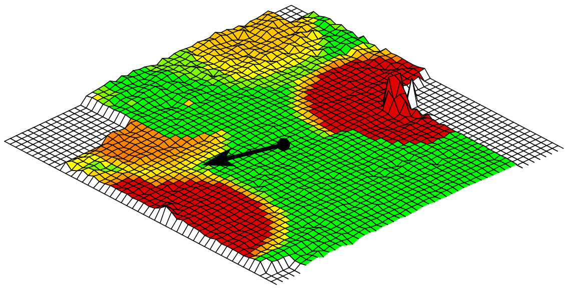

(a) map 1

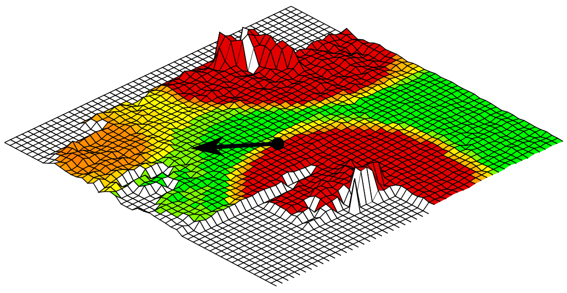

(b) map 2


(c) map 3


(d) map 4


(e) map 5


(f) map 6


(g) map 7

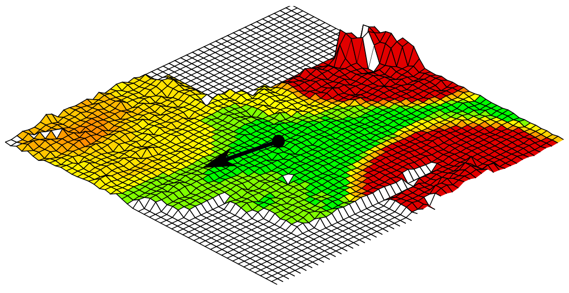

(h) map 8

Figure 4.6.: Local maps of the homing run with the full Trail-Map. Green – orange: traversable. Red: untraversable. Black arrow: desired homing direction. Blue arrow: adjusted homing direction.

robot is able to take a different path during homing, when its intended direction is blocked by an obstacle. A detailed visualization of the homing vectors in this area is given by Fig. 4.8.

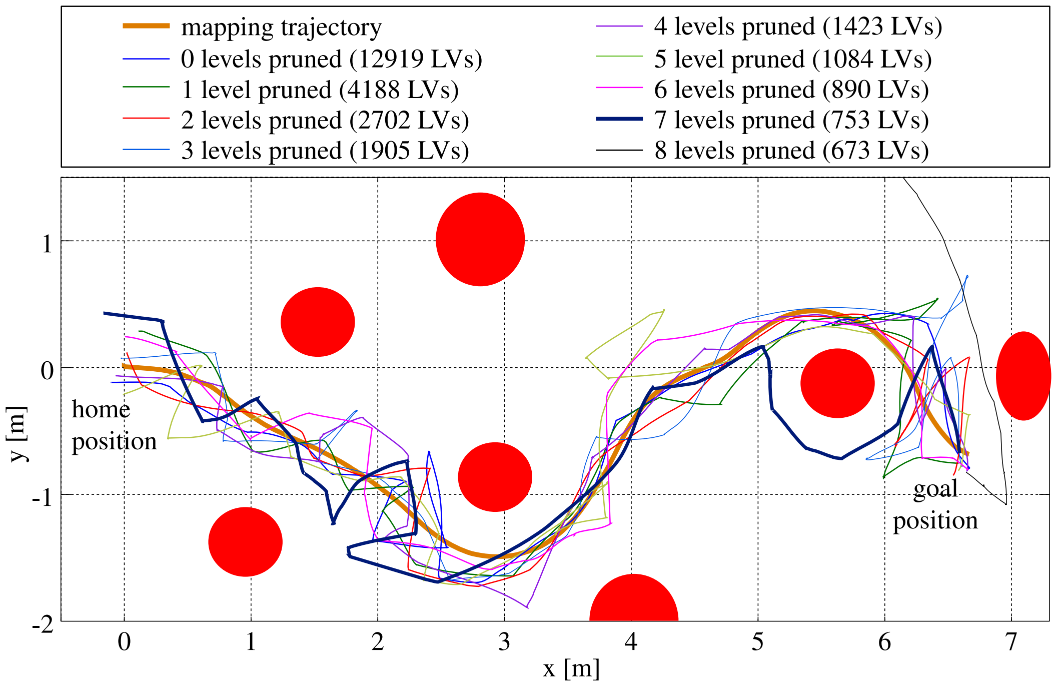


Figure 4.7.: Homing paths with the Trail-Map pruned by different levels. The homing path using the Trail-Map pruned by 7 levels is highlighted, because here the robot took a different path than during mapping.

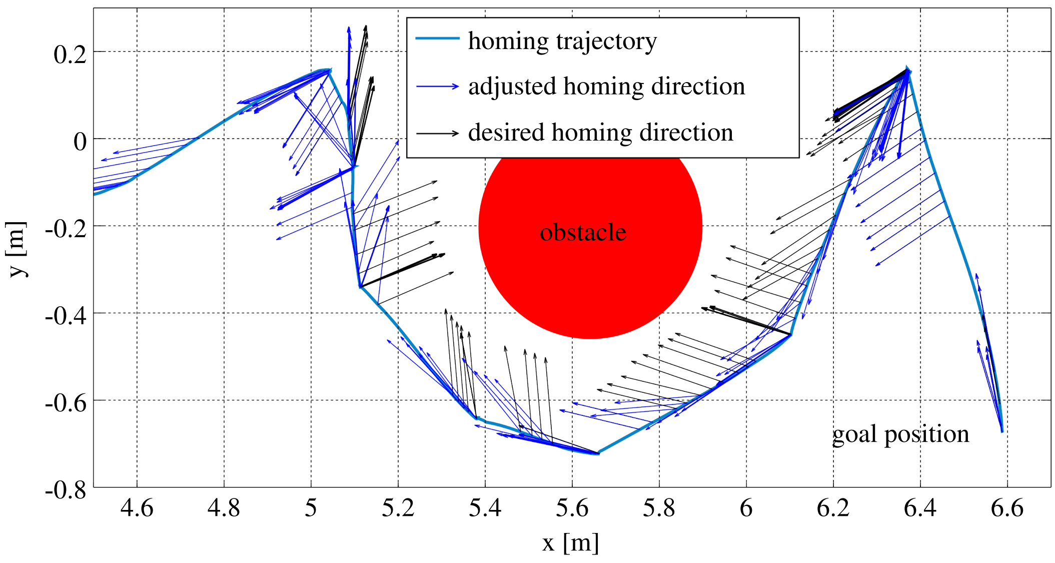


Figure 4.8.: Desired and adjusted homing vectors for passing an obstacle during homing with the Trail-Map pruned by 7 levels

4.4. Conclusion

In this chapter we showed how to combine the local metric and the global topological navigation approaches to a hybrid navigation method that enables the robot to retrace learned paths in rough terrain without putting itself to any risk. The robot creates and maintains a moving local cost map of its immediate surroundings and uses this data to locally adjust the homing direction, which is computed from the landmark configuration of the current position and the stored viewframes. The experimental results have shown that the robot can retrace learned paths without putting itself to any risk, even if the Trail-Map is pruned very strongly. However, the homing performance is not optimal yet. The robot makes detours caused by noisy homing vectors and the applied motion control law. This behavior needs to be improved. Nevertheless, this chapter could show that the proposed approach is applicable to visual robot homing in unstructured environments.

5. Conclusion

In this thesis we developed a method for efficient visual homing in rough terrain. Visual homing describes the ability of a mobile robot to return to its starting position after having traveled away from it. This is an essential skill in many applications in which robots serve as tools for researchers or rescue workers. For example, in a sample-return scenario on a foreign planet the rover is commanded to scientifically interesting places by an operator, takes samples and then autonomously returns the samples to the lander. Also when robots are used to distribute sensor instruments, e.g. for seismic surveillance, they will have to return to a base station after placing each instrument. Furthermore, the stored path information cannot only be used for homing, but can also be transferred to other robots for retracing the path in the same direction. As an example, in disaster scenarios, robots could provide medical aids for trapped people, which were previously found by a scouting robot that provided the path information for its robotic colleagues.

The presented approach to the visual homing problem is only based on lightweight, passive proprioceptive and exteroceptive sensors, namely a stereo camera, an IMU, an omnidirectional camera and odometry sensors. Furthermore, it does not depend on external infrastructure such as GPS or artificial landmarks. The method is divided into a local metric navigation and a global topological navigation method, which have different requirements and are combined to achieve efficient visual homing in rough terrain.

The task of the local metric navigation method is to guide the robot along a safe path in the desired direction, which is computed by the global navigation method. For this, we developed a multisensor data fusion method for robust pose estimation from IMU data, visual odometry and leg odometry. The measurements are fused in an indirect information filter framework and the estimated visual odometry errors are taken into account. Thus, the estimated poses are robust against visual disturbances and leg slip. We demonstrated the accuracy and robustness of the pose estimation method in experiments with a six-legged walking robot in a gravel testbed. Furthermore, we adapted a method for generating dense elevation maps to work with stereo disparity images. Using this method, digital elevation maps of arbitrary resolution can be built from stereo data. We could show that this method outperforms the straightforward point cloud approach, especially when wide angle lenses are used. We used the estimated poses to combine the elevation maps from single disparity images to a local digital elevation map. To show the overall performance of the local metric navigation method, we applied traversability estimation to create a traversability cost map of the environment and implemented a method for planning paths based on the estimated traversability of the terrain. We demonstrated the functionality of the method in experiments with a six-legged walking robot and could show that the local navigation method is robust against visual disturbances and against slip on loose ground. Since metric navigation is only required in the immediate surroundings of the robot, the size of the local metric map is limited to a small region around the robot, resulting in runtimes which are constant with respect to the length of the traveled path.

The global navigation method aims at building an efficient representation of the robot path and provides a framework for retracing this path. For this, it does not require any metric distance information, but is only based on viewframes, which contain the bearing angle configurations of natural landmarks at certain locations in the environment. The robot records a new viewframe along its path whenever the dissimilarity to the previously recorded viewframe exceeds a threshold. Furthermore, for retracing the path, the robot computes homing vectors that lead it in the direction to align the currently perceived landmark configuration with the configuration of a goal viewframe. In this work, we described and discussed a variety of different methods for calculating dissimilarities and for computing homing vectors between viewframes. Additionally, we proposed new homing vector calculation methods that outperform the existing ones in cases of non-uniform landmark distributions and false landmark matches. Furthermore, we developed the Trail-Map, a novel scalable data structure for the efficient storage of a robot path consisting of a sequence of viewframes. It is based on the insight that the bearing angles to distant landmarks and landmarks in the direction of movement hardly change during the motion of the robot (translation invariant landmarks), while the bearing angles to close landmarks change quickly (translation variant landmarks). The Trail-Map avoids redundancies by only inserting new landmark observations, if they have changed significantly compared to the last viewframe. The Trail-Map stores the landmark views in their hierarchical order of translation invariance, with long-term stable landmarks in the higher levels and quickly-changing, volatile landmarks in the lower levels. This structure not only avoids redundancies, but also allows easy scaling of the map. In case of memory shortage, the Trail-Map can be pruned by deleting landmark observations in the lower levels of the Trail-Map. This means discarding volatile landmark information and retaining the important, stable translation invariant landmark observations. We demonstrated in simulations that the Trail-Map significantly outperforms the LT-Map [2], which is the only existing scalable data structure for viewframe-based navigation, in terms of memory requirements, path accuracy and runtime behavior. We evaluated the homing performance with the Trail-Map in simulations and in real-world experiments and also analyzed the effect of different pruning levels. In the experiments we used natural landmarks extracted from omnidirectional camera images. In long-range outdoor experiments we could demonstrate that homing using the Trail-Map requires less than 300kB of memory per 100m path length, while its runtime is constant with respect to the path length. Compared to the state-of-the-art methods for retracing taught paths [27, 51], this means memory savings of more than 97%.

By combining the local and the global navigation methods, we developed a visual homing approach for rough terrain, which is suitable for mobile robots with limited computational and memory resources. For this, we proposed a simple obstacle avoidance method that adjusts homing vectors so that they point to a safe direction. We showed in experiments that this method enables the robot to safely retrace previously traveled paths in rough terrain.

The proposed method for visual homing in rough terrain is a powerful alternative to metric navigation methods in applications in which a global metric representation of the environment is not required and the robot should only retrace a previously traversed path. Thus, the method is another step towards more autonomous and efficient mobile robots to support human operators, for example in search-and-rescue or planetary exploration scenarios.

Future Perspective

There are several possibilities to improve the proposed visual homing method.

In future work, a representation of uncertainty should be added to the local mapping and traversability estimation process. The pose estimation method already estimates the covariances of the robot pose, but they are not used to create a probabilistic map so far. Additionally, the confidence of the stereo data should be considered for creating the local map. Then, the traversability estimation process should also be adjusted to work with uncertain geometric map information. Another extension to the presented work would be a GPU or FPGA implementation of the mapping and traversability estimation method, as a parallelization of these processes is possible and is expected to decrease the computation time significantly.

A major improvement of the proposed viewframe-based homing method would be a more robust landmark detection and matching method. Our experiments showed that the matching ratios of the BRISK features were very low, so that the majority of landmark observations in the Trail-Map could not be used for homing. Furthermore, since in its current implementation the Trail-Map cannot be corrected, every landmark view which is added to the Trail-Map will remain there, except the corresponding level of the Trail-Map is deleted during pruning. For retracing a path several times, a method for correcting the Trail-Map might be beneficial. Then, unstable landmarks could be deleted from the map and angular observation errors can be corrected.

In the current implementation of Trail-Map-based homing, the robot does not try to find any shortcuts. Thus, it will also travel along loops in the homing phase. To optimize the robot's homing behavior, a topological network of viewframes that corresponds to a kind of *cognitive map* would be beneficial. However, since this requires the computation of dissimilarities between all viewframes, such a method would lead to a higher computational complexity. A first work on the creation of a topological map of viewframes is presented by Vayugundla [96]. In this work, dimensionality reduction techniques are used to compute the 2D distribution of viewframes based on their dissimilarities. This approach is closely related to another possible extension, which is the development of a Trail-Map-based SLAM method. In the current implementation, no metric information is used to create the Trail-Map, although metric pose estimates and covariances are available. These viewframe poses could potentially be corrected using the dissimilarities between the viewframes as computed during the creation of the topological network of viewframes mentioned before.

Additionally, the proposed hybrid homing method with local obstacle avoidance can still be improved. The experiments showed that the robot made unnecessary detours caused by noisy homing vectors in combination with the motion control law. Here, a better motion control method could lead to shorter homing paths. Furthermore, the obstacle avoidance method assumes that the homing vectors computed from the landmark configurations are sufficiently accurate, and that the environment does not change significantly. In case the previously traveled path is blocked so that the robot runs into a dead end which is bigger than the size of the local map, a strategy to find an alternative way must be implemented.

A. Appendix: Serial RANSAC for Landmark Outlier Rejection

This appendix gives the equations and algorithms for the landmark outlier rejection method proposed by Jäger [42] [43].

A.1. Rotation Estimation with Unknown Roll and Pitch

Wahba [97] first posed the problem of finding the rotation matrix $\boldsymbol{R}$ that minimizes the cost function

$$J(\boldsymbol{R}) = \frac{1}{2} \sum_{i=1}^{n} w_i \, |\boldsymbol{l}_i' - \boldsymbol{R}\, \boldsymbol{l}_i|^2, \tag{A.1}$$

where $\boldsymbol{l}_i'$ is a landmark observation unit vector in a fixed body frame, $\boldsymbol{l}_i$ is the corresponding landmark unit vector in a known reference frame and w_i are the weights.

Markley [64] solved this problem by singular value decomposition as follows: First, it can be shown that

$$J(\boldsymbol{R}) = \frac{1}{2} \sum_{i=1}^{n} w_i \, |\boldsymbol{l}_i' - \boldsymbol{R}\, \boldsymbol{l}_i|^2 = 1 - \sum_{i=1}^{n} w_i \, \boldsymbol{l}_i'^T \boldsymbol{R}\, \boldsymbol{l}_i = 1 - \mathrm{tr}(\boldsymbol{R}\, \boldsymbol{B}^T) \tag{A.2}$$

$$\boldsymbol{B} = \sum_{i=1}^{n} w_i \, \boldsymbol{l}_i' \, \boldsymbol{l}_i^T, \tag{A.3}$$

assuming $\sum_{i=1}^{n} w_i = 1$. Matrix $\boldsymbol{B}$ can be factorized by singular value decomposition into

$$\boldsymbol{B} = \boldsymbol{U}_B \, \boldsymbol{S}_B \, \boldsymbol{V}_B^T, \tag{A.4}$$

$$\boldsymbol{S}_B = \mathbf{diag}[s_1, s_2, s_3], \tag{A.5}$$

where $\boldsymbol{S}_B$ is a diagonal matrix with $s_1 \geq s_2 \geq s_3 \geq 0$ being the singular values of $\boldsymbol{B}$, and $\boldsymbol{U}_B$ and $\boldsymbol{V}_B$ are orthogonal matrices. Then it can be shown that

$$\boldsymbol{R} = \boldsymbol{U}_B \, \mathbf{diag}[1, 1, \det(\boldsymbol{U}_B) \det(\boldsymbol{V}_B)] \, \boldsymbol{V}_B^T \tag{A.6}$$

minimizes the cost function $J(\boldsymbol{R})$. The weights w_i for calculating $\boldsymbol{B}$ were computed as $w_i = |\boldsymbol{l}_i' - \boldsymbol{l}_i^T|$. Using these weights, landmark vectors with large displacements have higher influence on the result.

A.2. Translation Estimation

The method for calculating the translation $\boldsymbol{t}$ between two sets of unit vectors $\boldsymbol{l}_i$ and $\boldsymbol{l}_i'$ was introduced by Jäger et al. [43]. The cost function to be minimized is

$$E(\boldsymbol{t}) = \sum_{i=1}^{n} w_i \, ((\boldsymbol{l}_i' \times \boldsymbol{l}_i)^T \, \boldsymbol{t})^2 - \lambda \, (|\boldsymbol{t}|^2 - 1), \tag{A.7}$$

where w_i are weights and λ is a Lagrange multiplier ensuring $|\boldsymbol{t}| = 1$. With $\boldsymbol{m}_i = \boldsymbol{l}_i' \times \boldsymbol{l}_i$ we get

$$E(\boldsymbol{t}) = \sum_{i=1}^{n} w_i \, (\boldsymbol{m}_i^T \, \boldsymbol{t})^2 - \lambda \, (| \, \boldsymbol{t} \, |^2 - 1) \tag{A.8}$$

$$= \boldsymbol{t}^T \sum_{i=1}^{n} (w_i \, \boldsymbol{m}_i \boldsymbol{m}_i^T) \, \boldsymbol{t} - \lambda \, (| \, \boldsymbol{t} \, |^2 - 1). \tag{A.9}$$

By setting $\boldsymbol{M} = \sum_{i=1}^{n} (w_i \, \boldsymbol{m}_i \boldsymbol{m}_i^T)$ we get

$$E(\boldsymbol{t}) = \boldsymbol{t}^T \, \boldsymbol{M} \, \boldsymbol{t} - \lambda \, (| \, \boldsymbol{t} \, |^2 - 1). \tag{A.10}$$

For minimizing $E(\boldsymbol{t})$ we set the derivative of $E(\boldsymbol{t})$ with respect to $\boldsymbol{t}$ to 0 and get

$$\frac{\partial E(\boldsymbol{t})}{\partial \boldsymbol{t}} = 2\boldsymbol{M} \, \boldsymbol{t} - 2\lambda \boldsymbol{t} = 0 \qquad \rightarrow \quad \lambda \boldsymbol{t} = \boldsymbol{M} \boldsymbol{t}, \tag{A.11}$$

which is an eigenvalue equation. Since $\boldsymbol{M}$ is symmetric, the singular values are equal to the eigenvalues. Thus, the sign-free vector $\tilde{\boldsymbol{t}}$ that solves this equation can be determined by singular value decomposition of $\boldsymbol{M}$:

$$\boldsymbol{M} = \boldsymbol{U}_M \boldsymbol{S}_M \boldsymbol{V}_M^T \quad \text{and} \quad \tilde{\boldsymbol{t}} = \boldsymbol{V}_{M;:,3}, \tag{A.12}$$

where $\boldsymbol{V}_{M;:,3}$ denotes the third column of the matrix $\boldsymbol{V}_M$ corresponding to the smallest eigenvalue. To determine the sign for $\tilde{\boldsymbol{t}}$, we use

$$\boldsymbol{t} = \begin{cases} -\ \tilde{\boldsymbol{t}} & \text{if} \quad \sum_{i=1}^{n} (\boldsymbol{l}'_i - \boldsymbol{l}_i^T)\, \tilde{\boldsymbol{t}} < 0 \\ \tilde{\boldsymbol{t}} & \text{else.} \end{cases} \tag{A.13}$$

The weights for calculating $\boldsymbol{M}$ are also computed as $w_i = |\boldsymbol{l}'_i - \boldsymbol{l}_i^T|$.

A.3. Serial RANSAC Pseudocode

```
Algorithm 3: Reject Outliers
Input   : set of initial matches $V = \{< \boldsymbol{l}_i, \boldsymbol{l}'_i > | i = 1 \dots n\}$
Output: outlier-free set of matches $V'$
1 $[V^{\text{rot}}_{\text{best}}, \boldsymbol{R}_{\text{best}}]$ = Reject_Rotational_Outliers ($V$)
2 $V = V \setminus V^{\text{rot}}_{\text{best}}$
3 foreach $< \boldsymbol{l}_i, \boldsymbol{l}'_i >$ in $V$ do
4     $\boldsymbol{l}_i = \boldsymbol{R}_{\text{best}} \boldsymbol{l}_i$
5 $[V^{\text{trans}}_{\text{best}}, \boldsymbol{t}_{\text{best}}]$ = Reject_Translational_Outliers ($V$)
6 $V' = V^{\text{rot}}_{\text{best}} \cup V^{\text{trans}}_{\text{best}}$
```

```
Algorithm 4: Update_Inliers_Translation
Input   : initial matches $V = \{< \boldsymbol{l}_i, \boldsymbol{l}'_i > | i = 1 \dots n\}$, initial inlying matches $V_{\text{in}}$
Output: refined inlying matches $V_{\text{in}}$
1 $\boldsymbol{t}$ = Compute_Translation($V_{\text{in}}$)
2 foreach $< \boldsymbol{l}_i, \boldsymbol{l}'_i >$ in $V \setminus V_{in}$ do
3     $\epsilon_i = |(\boldsymbol{l}_i \times \boldsymbol{l}'_i)^T \boldsymbol{t}|$                ▹ Compute error
4     if $\epsilon_i < \epsilon_{thresh}$ and $(\boldsymbol{l}_i - \boldsymbol{l}'_i)^T \boldsymbol{t} < 0$ then
5         $V_{\text{in}} = V_{\text{in}} \cup < \boldsymbol{l}_i, \boldsymbol{l}'_i >$
```

Algorithm 5: Update_Inliers_Rotation

```
Input  : initial matches $V = \{< \boldsymbol{l}_i, \boldsymbol{l}'_i > | i = 1 \ldots n\}$, initial inlying matches $V_{\text{in}}$
Output: refined inlying matches $V_{\text{in}}$
1  $\boldsymbol{R}$ = Compute_Rotation($V_{\text{in}}$)
2  foreach $< \boldsymbol{l}_i, \boldsymbol{l}'_i >$ in $V \setminus V_{in}$ do
3      $\epsilon_i = |\boldsymbol{R}\boldsymbol{l}_i - \boldsymbol{l}'_i|$                              ▹ Compute error
4      if $\epsilon_i < \epsilon_{thresh}$ then
5          $V_{\text{in}} = V_{\text{in}} \cup < \boldsymbol{l}_i, \boldsymbol{l}'_i >$
```

Algorithm 6: Reject_Translational_Outliers

```
Input  : initial matches $V = \{< \boldsymbol{l}_i, \boldsymbol{l}'_i > | i = 1 \ldots n\}$
Output: inlying matches $V_{\text{best}}$, estimated translation $\boldsymbol{t}_{\text{best}}$
1  $V_{\text{best}} = \emptyset$
2  for i=1:maxIterations do
3      $V_{\text{in}}$ = randomly sample 3 elements from $V$
4      $V_{\text{in}}$ = Update_Inliers_Translation($V_{\text{in}}$, $V$)
5      if size($V_{in}$) > size($V_{best}$) then
6          $V_{\text{in}}$ = Update_Inliers_Translation($V_{\text{in}}$, $V$)
7          $\boldsymbol{t}_{\text{best}}$ = Compute_Translation($V_{\text{in}}$)
8          $V_{\text{best}} = V_{\text{in}}$
9      if size($V_{best}$) > 0.9·size($V$) then
10         break for
```

Algorithm 7: Reject_Rotational_Outliers

```
Input  : initial matches $V = \{< \boldsymbol{l}_i, \boldsymbol{l}'_i > | i = 1 \ldots n\}$
Output: inlying matches $V_{\text{best}}$, estimated rotation $\boldsymbol{R}_{\text{best}}$
1  $V_{\text{best}} = \emptyset$
2  for i=1:maxIterations do
3      $V_{\text{in}}$ = randomly sample 3 elements from $V$
4      $V_{\text{in}}$ = Update_Inliers_Rotation($V_{\text{in}}$, $V$)
5      if size($V_{in}$) > size($V_{best}$) then
6          $V_{\text{in}}$ = Update_Inliers_Rotation($V_{\text{in}}$, $V$)
7          $\boldsymbol{R}_{\text{best}}$ = Compute_Rotation($V_{\text{in}}$)
8          $V_{\text{best}} = V_{\text{in}}$
9      if size($V_{best}$ > 0.9 size($V$) then
10         break for
```

Bibliography

[1] A. A. Argyros, K. E. Bekris, S. C. Orphanoudakis, and L. E. Kavraki. Robot homing by exploiting panoramic vision. *Autonomous Robots*, 19(1):7–25, 2005. Cited on page 63.

[2] M. Augustine, E. Mair, A. Stelzer, F. Ortmeier, D. Burschka, and M. Suppa. Landmark-Tree Map: A biologically inspired topological map for long-distance robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO) 2012, Guangzhou, China*, 2012. Cited on pages 8, 9, 10, 64, 85, and 165.

[3] T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (SLAM): Part II. *IEEE Robotics & Automation Magazine*, 13(3):108–117, 2006. Cited on page 3.

[4] H. Bay, T. Tuytelaars, and L. Van Gool. SURF: Speeded up robust features. In *Computer vision–ECCV 2006*, pages 404–417. Springer, 2006. Cited on page 122.

[5] J. L. Blanco, J. A. Fernandez-Madrigal, and J. Gonzalez. Toward a unified Bayesian approach to hybrid metric-topological SLAM. *IEEE Transactions on Robotics*, 24 (2):259–270, 2008. Cited on page 3.

[6] I. Bogoslavskyi, O. Vysotska, J. Serafin, G. Grisetti, and C. Stachniss. Efficient traversability analysis for mobile robots using the kinect sensor. In *European Conference on Mobile Robots (ECMR) 2013*, pages 158–163. IEEE, 2013. Cited on page 46.

[7] O. Booij, B. Terwijn, Z. Zivkovic, and B. Kröse. Navigation using an appearance based topological map. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 2007*, pages 3927–3932, 2007. Cited on page 3.

[8] J. Bortz. A new mathematical formulation for strapdown inertial navigation. *IEEE Transactions on Aerospace Electronic Systems*, 7:61–66, 1971. Cited on page 20.

[9] F. Caron, E. Duflos, D. Pomorski, and P. Vanheeghe. GPS/IMU data fusion using multisensor Kalman filtering: Introduction of contextual aspects. *Information Fusion*, 7(2):221–230, 2006. Cited on page 15.

[10] B. A. Cartwright and T. S. Collett. Landmark learning in bees: Experiments and models. *Journal of Comparative Physiology*, 151(4):521–543, 1983. Cited on pages 62, 69, and 77.

[11] B. A. Cartwright and T. S. Collett. Landmark maps for honeybees. *Biological Cybernetics*, 57(1-2):85–93, 1987. Cited on page 62.

[12] A. Cherubini and F. Chaumette. Visual navigation of a mobile robot with laser-based collision avoidance. *The International Journal of Robotics Research*, 32(2):189–205, 2013. Cited on pages 3, 4, and 64.

[13] A. Chilian. Stereokamerabasierte Navigation eines Krabbelroboters auf unebenem Gelände. Diplomarbeit, Technische Universität Ilmenau, Fakultät für Maschinenbau, 2008. Cited on pages 7, 46, 48, 49, and 50.

[14] A. Chilian and H. Hirschmüller. Stereo camera based navigation of mobile robots on rough terrain. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2009*, pages 4571–4576, 2009. Cited on pages 10, 37, 46, and 48.

[15] A. Chilian, H. Hirschmüller, and M. Görner. Multisensor data fusion for robust pose estimation of a six-legged walking robot. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS) 2011*, pages 2497–2504, 2011. Cited on page 9.

[16] D. M. Cole and P. M. Newman. Using laser range data for 3D SLAM in outdoor environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 2006*, pages 1556–1563, 2006. Cited on page 35.

[17] D. Dai and D. T. Lawton. Range-free qualitative navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 1993*, pages 783–790, 1993. Cited on page 63.

[18] F. Dellaert and M. Kaess. Square Root SAM: Simultaneous localization and mapping via square root information smoothing. *The International Journal of Robotics Research*, 25(12):1181–1203, 2006. Cited on page 117.

[19] G. Dissanayake, S. Sukkarieh, E. Nebot, and H. Durrant-Whyte. The aiding of a low-cost strapdown inertial measurement unit using vehicle model constraints for land vehicle applications. *IEEE Transactions on Robotics and Automation*, 17(5):731–747, 2002. Cited on pages 15 and 16.

[20] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping (SLAM): Part I. *IEEE Robotics & Automation Magazine*, 13(2):99–110, 2006. Cited on page 3.

[21] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, 3(3):249–265, 1987. Cited on page 2.

[22] C. Estrada, J. Neira, and J. D. Tardós. Hierarchical SLAM: Real-time accurate mapping of large environments. *IEEE Transactions on Robotics*, 21(4):588–596, 2005. Cited on page 116.

[23] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. Cited on page 123.

[24] M. O. Franz, B. Schölkopf, H. A. Mallot, and H. H. Bülthoff. Where did I take that snapshot? Scene-based homing by image matching. *Biological Cybernetics*, 79(3): 191–202, 1998. Cited on page 69.

[25] F. Fraundorfer, C. Engels, and D. Nistér. Topological mapping, localization and navigation using image collections. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2007*, pages 3872–3877, 2007. Cited on page 3.

[26] C. Früh, S. Jain, and A. Zakhor. Data processing algorithms for generating textured 3D building facade meshes from laser scans and camera images. *International Journal of Computer Vision*, 61(2):159–184, 2005. Cited on pages 36 and 39.

[27] P. Furgale and T. D. Barfoot. Visual teach and repeat for long-range rover autonomy. *Journal of Field Robotics*, 27(5):534–560, 2010. Cited on pages 3, 63, 134, 148, and 165.

[28] J. Gasós and A. Saffiotti. Integrating Fuzzy Geometric Maps and Topological Maps for Robot Navigation. In *Proceedings of the 3rd International Symposium on Soft Computing*, 1999. Cited on page 3.

[29] T. Goedemé, T. Tuytelaars, L. Van Gool, G. Vanacker, and M. Nuttin. Feature based omnidirectional sparse visual path following. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2005*, pages 1806–1811, 2005. Cited on page 63.

[30] S. B. Goldberg, M. W. Maimone, and L. Matthies. Stereo vision and rover navigation software for planetary exploration. In *IEEE Aerospace Conference Proceedings 2002*, pages 2025–2036, March 2002. Cited on pages 46, 47, and 48.

[31] M. Görner and A. Stelzer. A leg proprioception based 6 DOF odometry for statically stable walking robots. *Autonomous Robots*, 34(4):311–326, 2013. Cited on pages 10 and 28.

[32] M. Görner, T. Wimböck, A. Baumann, M. Fuchs, T. Bahls, M. Grebenstein, C. Borst, J. Butterfass, and G. Hirzinger. The DLR-Crawler: A testbed for actively compliant hexapod walking based on the fingers of DLR-Hand II. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2008*, pages 1525 – 1531, September 2008. Cited on page 28.

[33] M. Görner, T. Wimböck, and G. Hirzinger. The DLR Crawler: Evaluation of gaits and control of an actively compliant six-legged walking robot. *Industrial Robot: An International Journal*, 36(4):344–351, 2009. Cited on pages 28 and 51.

[34] M. Görner, A. Chilian, and H. Hirschmüller. Towards an autonomous walking robot for planetary surfaces. In *Proceedings of the 10th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, September 2010. Cited on pages 10 and 50.

[35] R. Hadsell, J. A. Bagnell, D. Huber, and M. Hebert. Space-carving kernels for accurate rough terrain estimation. *The International Journal of Robotics Research*, 2010. Cited on page 36.

[36] D. Hähnel, W. Burgard, and S. Thrun. Learning compact 3D models of indoor and outdoor environments with a mobile robot. *Robotics and Autonomous Systems*, 44 (1):15–27, 2003. Cited on page 36.

[37] D. M. Helmick, Y. Cheng, D. S. Clouse, L. H. Matthies, and S. I. Roumeliotis. Path following using visual odometry for a mars rover in high-slip environments. In *IEEE Aerospace Conference Proceedings 2004*, pages 772–789, 2004. Cited on pages 15 and 16.

[38] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments. In *The 12th International Symposium on Experimental Robotics (ISER)*, volume 20, pages 22–25, 2010. Cited on page 35.

[39] H. Hirschmüller. Stereo processing by semi-global matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):328–341, February 2008. Cited on pages 126 and 156.

[40] A. Howard, D. F. Wolf, and G. S. Sukhatme. Towards 3D mapping in large urban environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2004*, volume 1, pages 419–424, 2004. Cited on page 35.

[41] D. F. Huber and M. Hebert. A new approach to 3-d terrain mapping. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 1999*, pages 1121–1127, 1999. Cited on pages 36 and 39.

[42] B. Jäger. Efficient vision-based navigation of mobile robots based on the LT-Map. Master's thesis, Technische Universität München, Lehrstuhl für Flugsystemdynamik, 2013. Cited on pages 122 and 167.

[43] B. Jäger, E. Mair, C. Brand, W. Stürzl, and M. Suppa. Efficient navigation based on the landmark-tree map and the Z^∞ algorithm using an omnidirectional camera. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2013*, pages 1930–1937, 2013. Cited on pages 103, 125, 167, and 168.

[44] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Transactions on Robotics*, 24(6):1365–1378, 2008. Cited on pages 3 and 117.

[45] K. Kawamura, A. B. Koku, D. M. Wilkes, R. A. Peters, and A. Sekmen. Toward egocentric navigation. *International Journal of Robotics and Automation*, 17(4):135–145, 2002. Cited on page 63.

[46] S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. In *Proceedings of the International Conference on Robotics and Automation (ICRA) 2002*, pages 968–975, 2002. Cited on page 49.

[47] J. Z. Kolter, Y. Kim, and A. Y. Ng. Stereo vision and terrain modeling for quadruped robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 2009*, pages 1557–1564, 2009. Cited on pages 2, 36, and 39.

[48] K. Konolige, M. Agrawal, and J. Sola. Large scale visual odometry for rough terrain. In *Proceedings of the International Symposium on Robotics Research*, 2007. Cited on page 15.

[49] K. Konolige, E. Marder-Eppstein, and B. Marthi. Navigation in hybrid metric-topological maps. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 2011*, pages 3041–3047, 2011. Cited on page 3.

[50] J. Kosecka, L. Zhou, P. Barber, and Z. Duric. Qualitative image based localization in indoors environments. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages II–3, 2003. Cited on page 62.

[51] T. Krajník, J. Faigl, V. Vonásek, K. Košnar, M. Kulich, and L. Přeučil. Simple yet stable bearing-only navigation. *Journal of Field Robotics*, 27(5):511–533, 2010. Cited on pages 3, 63, 134, 148, and 165.

[52] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 2011*, pages 3607–3613, 2011. Cited on page 117.

[53] I. S. Kweon and T. Kanade. High-resolution terrain map from multiple sensor data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 278–292, 1992. Cited on pages 2, 7, 9, 36, 39, 40, and 59.

[54] D. Lambrinos, T. Labhart, and R. Pfeifer. A mobile robot employing insect strategies for navigation. *Robotics and Autonomous Systems*, 30(1-2):39–64, 2000. Cited on pages 8, 69, 70, 71, 72, 82, and 84.

[55] P. Lamon and R. Siegwart. Inertial and 3d-odometry fusion in rough terrain-towards real 3d navigation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2004*, volume 2, pages 1716–1721, 2004. Cited

on page 16.

[56] R. C. Leishman, T. W. McLain, and R. W. Beard. Relative navigation approach for vision-based aerial GPS-denied navigation. *Journal of Intelligent & Robotic Systems*, 74(1-2):97–111, 2014. Cited on pages 3 and 116.

[57] J. Leonard and P. Newman. Consistent, convergent, and constant-time SLAM. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 1143–1150. Morgan Kaufmann Publishers Inc., 2003. Cited on page 116.

[58] S. Leutenegger, M. Chli, and R. Y. Siegwart. BRISK: Binary robust invariant scalable keypoints. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV) 2011*, pages 2548–2555, 2011. Cited on page 122.

[59] M. Liu, C. Pradalier, Q. Chen, and R. Siegwart. A bearing-only 2D/3D-homing method under a visual servoing framework. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 2010*, pages 4062–4067, 2010. Cited on pages 78, 83, and 84.

[60] M. Liu, C. Pradalier, F. Pomerleau, and R. Siegwart. Scale-only visual homing from an omnidirectional camera. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 2012*, 2012. Cited on pages 79 and 80.

[61] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004. Cited on page 122.

[62] E. Mair, G. D. Hager, D. Burschka, M. Suppa, and G. Hirzinger. Adaptive and generic corner detection based on the accelerated segment test. In *Computer Vision–ECCV 2010*, pages 183–196. Springer, 2010. Cited on page 122.

[63] E. Mair, M. Augustine, B. Jäger, A. Stelzer, C. Brand, D. Burschka, and M. Suppa. A biologically inspired navigation concept based on the Landmark-Tree Map for efficient long-distance robot navigation. *Advanced Robotics*, 28(5):289–302, 2014. Cited on pages 9, 11, and 66.

[64] F. L. Markley. Attitude determination using vector observations and the singular value decomposition. *The Journal of the Astronautical Sciences*, 36(3):245–258, 1988. Cited on page 167.

[65] Y. Matsumoto, M. Inaba, and H. Inoue. Visual navigation using view-sequenced route representation. In *Proceedings of the IEEE International Conference on Robotics and*

Automation (ICRA) 1996, pages 83–88, 1996. Cited on page 62.

[66] M. J. Milford and G. F. Wyeth. Mapping a suburb with a single camera using a biologically inspired SLAM system. *IEEE Transactions on Robotics*, 24(5):1038–1053, 2008. Cited on page 118.

[67] M. J. Milford, G. F. Wyeth, and D. F. Rasser. RatSLAM: A hippocampal model for simultaneous localization and mapping. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 2004*, pages 403–408, 2004. Cited on pages 3 and 117.

[68] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Edmonton, Canada, 2002. AAAI. Cited on pages 3 and 116.

[69] H. P. Moravec. Robot spatial perception by stereoscopic vision and 3D evidence grids. Technical report, CMU Robotics Institute, September 1996. Cited on pages 2 and 36.

[70] M. Müller and R. Wehner. The hidden spiral: Systematic search and path integration in desert ants, cataglyphis fortis. *Journal of Comparative Physiology A*, 175(5):525–530, 1994. Cited on page 103.

[71] R. R. Murphy. *Disaster robotics*. MIT Press, 2014. Cited on page 1.

[72] C. J. Ostafew, A. P. Schoellig, and T. D. Barfoot. Visual teach and repeat, repeat, repeat: Iterative learning control to improve mobile robot path tracking in challenging outdoor environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2013*, pages 176–181, 2013. Cited on page 3.

[73] P. Payeur, P. Hébert, D. Laurendeau, and C. M. Gosselin. Probabilistic octree modeling of a 3d dynamic environment. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 1997*, volume 2, pages 1289–1296, 1997. Cited on page 36.

[74] C. Plagemann, S. Mischke, S. Prentice, K. Kersting, N. Roy, and W. Burgard. A Bayesian regression approach to terrain mapping and an application to legged robot locomotion. *Journal of Field Robotics*, 26(10):789–811, 2009. Cited on page 36.

[75] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV) 2005*, volume 2, pages 1508–1515, 2005. Cited on page 122.

[76] S. I. Roumeliotis and J. W. Burdick. Stochastic cloning: A generalized framework for processing relative state measurements. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 2002*, volume 2, pages 1788–1795, 2002. Cited on pages 16 and 21.

[77] S. I. Roumeliotis, G. S. Sukhatme, and G. A. Bekey. Circumventing dynamic modeling: Evaluation of the error-state Kalman filter applied to mobile robot localization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 1999*, volume 2, pages 1656–1663, 1999. Cited on page 16.

[78] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz. Towards 3D point cloud based object maps for household environments. *Robotics and Autonomous Systems*, 56(11):927–941, 2008. Cited on page 36.

[79] K. Schmid, F. Ruess, M. Suppa, and D. Burschka. State estimation for highly dynamic flying systems using key frame odometry with varying time delays. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2012*, pages 2997–3004, 2012. Cited on page 16.

[80] S. Šegvić, A. Remazeilles, A. Diosi, and F. Chaumette. Large scale vision-based navigation without an accurate global reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2007*, pages 1–8, 2007. Cited on pages 3 and 63.

[81] S. Šegvić, A. Remazeilles, A. Diosi, and F. Chaumette. A mapping and localization framework for scalable appearance-based navigation. *Computer Vision and Image Understanding*, 113(2):172–187, 2009. Cited on pages 3 and 4.

[82] S. Singh, R. Simmons, T. Smith, A. Stentz, V. Verma, A. Yahja, and K. Schwehr. Recent progress in local and global traversability for planetary rovers. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 2000*, volume 2, pages 1194–1200, 2000. Cited on page 46.

[83] J. Sola, A. Monin, M. Devy, and T. Vidal-Calleja. Fusing monocular information in multicamera SLAM. *IEEE Transactions on Robotics*, 24(5):958–968, 2008. Cited on page 114.

[84] J. Sola, D. Marquez, J. Codol, and T. Vidal-Calleja. An EKF-SLAM toolbox for MATLAB, 2009. URL `http://www.iri.upc.edu/people/jsola/JoanSola/eng/toolbox.html`. Cited on pages 101, 113, and 115.

[85] J. Sola, T. Vidal-Calleja, J. Civera, and J. M. M. Montiel. Impact of landmark parametrization on monocular EKF-SLAM with points and lines. *International Journal of Computer Vision*, 97(3):339–368, 2012. Cited on page 114.

[86] A. Stelzer, H. Hirschmüller, and M. Görner. Stereo-vision-based navigation of a six-legged walking robot in unknown rough terrain. *The International Journal of Robotics Research*, 31(4):381–402, 2012. Cited on pages 9 and 27.

[87] A. Stelzer, E. Mair, and M. Suppa. Trail-Map: A scalable landmark data structure for biologically inspired range-free navigation. In *Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO) 2014, Bali, Indonesia*, pages 2138–2145, 2014. Cited on page 9.

[88] A. Stelzer, M. Suppa, and W. Burgard. Trail-Map-based homing under the presence of sensor noise. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2015*, pages 929–936, 2015. Cited on page 10.

[89] T. Suzuki, M. Kitamura, Y. Amano, and T. Hashizume. 6-DOF localization for a mobile robot using outdoor 3D voxel maps. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2010*, pages 5737–5743, 2010. Cited on pages 2 and 36.

[90] S. Thrun, Y. Liu, D. Koller, A. Y. Ng, Z. Ghahramani, and H. Durrant-Whyte. Simultaneous localization and mapping with sparse extended information filters. *The International Journal of Robotics Research*, 23(7-8):693, 2004. Cited on page 3.

[91] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press, Cambridge, 1. edition, 2005. Cited on pages 116 and 117.

[92] R. Triebel, P. Pfaff, and W. Burgard. Multi-level surface maps for outdoor terrain mapping and loop closing. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2006*, 2006. Cited on page 36.

[93] A. Vardy. Long-range visual homing. In *Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO) 2006*, pages 220–226, 2006. Cited on pages 3 and 62.

[94] A. Vardy and R. Möller. Biologically plausible visual homing methods based on optical flow techniques. *Connection Science*, 17(1-2):47–89, 2005. Cited on page 70.

[95] J. F. Vasconcelos, P. Oliveira, and C. Silvestre. Inertial navigation system aided by GPS and selective frequency contents of vector measurements. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference (GNC2005)*, San Francisco, USA, August 2005. Cited on page 19.

[96] M. Vayugundla. Experimental evaluation and improvement of a viewframe-based navigation method. Master's thesis, University of Applied Sciences Bonn-Rhein-Sieg, Department of Computer Science, 2015. Cited on page 166.

[97] G. Wahba. A least squares estimate of satellite attitude. *SIAM review*, 7(3):409, 1965. Cited on page 167.

[98] K. Weber, S. Venkatesh, and M. Srinivasan. Insect-inspired robotic homing. *Adaptive Behavior*, 7(1):65–97, 1999. Cited on pages 77, 81, 83, and 84.

[99] N. Winters and J. Santos-Victor. Omni-directional visual navigation. In *Proceedings of the 7th International Symposium on Intelligent Robotic Systems (SIRS) 1999*, pages 109–118, 1999. Cited on page 62.

[100] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *Proceedings of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, volume 2, 2010. Cited on page 36.

[101] C. Ye and J. Borenstein. A method for mobile robot navigation on rough terrain. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 2004*, volume 4, pages 3863–3869, 2004. Cited on page 46.

[102] C. Ye and J. Borenstein. A novel filter for terrain mapping with laser rangefinders. *IEEE Transactions on Robotics*, 20(5):913–923, 2004. Cited on pages 2 and 36.

[103] J. Zeil, M. I. Hofmann, and J. S. Chahl. Catchment areas of panoramic snapshots in outdoor scenes. *Journal of the Optical Society of America A*, 20(3):450–469, 2003. Cited on page 69.

[104] A. M. Zhang and L. Kleeman. Robust appearance based visual route following for navigation in large-scale outdoor environments. *The International Journal of Robotics Research*, 28(3):331–356, 2009. Cited on page 62.

[105] Z. Zhang. Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision*, 13(2):119–152, 1994. Cited on page 46.

[106] Z. Zivkovic, B. Bakker, and B. Kröse. Hierarchical map building using visual landmarks and geometric constraints. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2005*, pages 2480–2485, 2005. Cited on page 3.