Efficient Approaches to Cleaning with Mobile Robots

Jürgen Michael Hess

Technische Fakultät
Albert-Ludwigs-Universität Freiburg im Breisgau

Dissertation zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften

Betreuer: Prof. Dr. Wolfram Burgard

Mai 2015

Efficient Approaches to Cleaning with Mobile Robots

Jürgen Michael Hess

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

Dekan:	Prof. Dr. Georg Lausen
Erstgutachter:	Prof. Dr. Wolfram Burgard
Zweitgutachter:	Prof. Dr. Bernhard Nebel

Tag der Disputation:	12. Mai 2015

Zusammenfassung

Haushaltsroboter sollen in Zukunft immer mehr Aufgaben im Haushalt übernehmen, wie etwa Aufräumen, Putzen oder Wäsche waschen. Könnten Roboter diese Aufgaben selbstständig ausführen, würden deren Anwender von einer großen Zeitersparnis profitieren. Des Weiteren könnten solche Roboter ältere oder körperlich eingeschränkte Menschen dabei unterstützen, länger in ihrer angestammten Umgebung zu wohnen und auf weniger Unterstützung durch Pflegepersonal angewiesen zu sein.

Reinigungungstätigkeiten gehören zu denjenigen Aufgaben im Haushalt, die derzeit am ehesten von Robotern durchgeführt werden können. Im häuslichen Umfeld sind Reinigungsroboter auch schon weit verbreitet. Bereits 2008 führten Prassler und Kosuge [120] insgesamt 13 Reinigungsroboter auf, die zum Einsatz im Haushalt angeboten wurden. Ihre Verbreitung wird auch dadurch deutlich, dass die Firma iRobot angibt, bis zum Jahr 2013 etwa zehn Millionen Haushaltsroboter weltweit verkauft zu haben [73]. Die meisten der derzeit angebotenen Roboter sind zur Bodenreinigung gedacht.

Obwohl schon sehr gebräuchlich, sind Reinigungsroboter oft nicht sehr effizient. Frühe Staubsaugroboter etwa, bewegten sich eher zufällig in der Umgebung. Neuere Modelle hingegen versuchen, systematisch die gesamte Umgebung abzudecken. Dieses Vorgehen dauert allerdings sehr lange und ist häufig nicht notwendig. Eine längere Laufzeit führt zu einer längeren Lärmbelastung des Anwenders sowie zu einem höheren Energieverbrauch. Auch stellt ein einmaliges Überfahren einer Stelle, etwa mit einem Staubsauger, nicht sicher, dass diese auch zufriedenstellend gereinigt wurde. Deshalb kann ein mehrfaches Staubsaugen derselben Stelle erforderlich sein. Bisweilen ist auch die zum Reinigen verfügbare Zeit begrenzt, wenn etwa kurzfristig Besuch erwartet wird. Darüber hinaus sind Reinigungsroboter derzeit auf die Bodenreinigung beschränkt und nicht in der Lage, andere Aufgaben, wie etwa Abstauben, durchzuführen.

Das Ziel der vorliegenden Arbeit ist es, neue innovative Techniken zu entwickeln, um bisherige Reinigungsroboter effizienter zu machen. Des Weiteren sollen Algorithmen für Manipulationsroboter erforscht werden, die es diesen ermöglichen, Oberflächen beliebiger Struktur zu reinigen, und somit Anwendungen wie Staubwischen auszuführen.

Im Folgenden werden die Fragestellungen dieser Arbeit an zwei typischen Aufgaben, die in jedem Haushalt anfallen, motiviert und veranschaulicht: dem Staubwischen und dem

Reinigen der Böden. Als erste Aufgabe wird das Staubwischen betrachtet. Dazu wird typischerweise ein trockenes Tuch oder ein Schwamm verwendet und die gesamte Oberfläche abgewischt. Staub ist sehr leicht zu entfernen, die abzuwischenden Oberflächen können aber sehr verschieden sein. In einem Haushalt findet man häufig viele ebene Flächen, wie etwa die von Regalen, Tischen oder Schränken. Des Weiteren kommen aber auch sehr komplizierte Freiformflächen vor, wie sie beispielsweise Vasen oder Lampen besitzen. Für einen Manipulationsroboter birgt die Aufgabe des Abstaubens deshalb mehrere Herausforderungen. Für die Lokalisierung sowie für die Planung kollisionsfreier Bewegungen des Roboters ist zunächst eine Karte der Umgebung erforderlich. Für das Säubern der Oberfläche eines Objektes benötigt der Roboter zum einen ein Modell der Oberflächenstruktur. Zum anderen muss er einen Pfad für seinen Endeffektor bzw. Werkzeug planen, der die gesamte Oberfläche abdeckt und Kollisionen mit anderen Gegenständen vermeidet. Zusätzlich sollte die für die Reinigung benötigte Zeit minimal gehalten werden.

Für andere Aufgaben, wie etwa das Wegwischen eines Kaffeeflecks auf einem Tisch, muss nicht die gesamte Oberfläche gereinigt werden. Könnte der Roboter mithilfe seiner Sensoren den Fleck erkennen, könnte er sich bei der Reinigung auf diesen Teil der Oberfläche konzentrieren, die Reinigung schneller abschließen und das Reinigungsergebnis anschließend überprüfen.

Die zweite Aufgabe, die betrachtet wird, ist das Staubsaugen. Beim Staubsaugen wird manchmal die gesamte Wohnung gereinigt, oft aber auch nur ein kleiner Teil. Einige Stellen müssen dabei häufiger gereinigt werden als andere. Der Eingangsbereich muss eventuell jeden Tag oder sogar mehrmals am Tag gereinigt werden. Für andere Teile der Wohnung hingegen, wie z.B. das Schlafzimmer, kann eine einmalige Reinigung pro Woche ausreichend sein. Manchmal muss dieselbe Fläche abhängig von der Art des Schmutzes und des Bodens mehrfach bearbeitet werden um sie vollständig zu säubern. Wie auch beim Beispiel des Manipulationsroboters könnte ein Staubsaugroboter effizienter reinigen, sofern ihm bekannt wäre, welche Teile der Umgebung derzeit verschmutzt sind.

Diese Überlegungen führen zu den folgenden wissenschaftlichen Fragestellungen, die in dieser Arbeit behandelt werden:

- Wie kann ein Roboter ein genaues Modell seiner Umgebung erstellen, das es ihm ermöglicht, kollisionsfreie Pfade zu planen und die Oberflächenstruktur der Objekte abzubilden?
- Wie kann ein Manipulationsroboter einen Pfad für sein Werkzeug planen, der die gesamte Oberfläche eines Gegenstandes abdeckt und dabei die Arbeitszeit minimiert?
- Wie kann ein Roboter lernen, die Teile der Umgebung zu erkennen, die gereinigt werden müssen, und diese Information nutzen, um die Aufgabe effizienter zu erfüllen?

- Wie kann ein Staubsaugroboter seinem Besitzer eine saubere Umgebung garantieren?

Diese Arbeit behandelt und beantwortet diese Fragen und ist wie folgt strukturiert. Kapitel 1 gibt eine Einführung in die Fragestellungen und erläutert die verwendeten mathematischen Schreibweisen. Die grundlegenden Techniken der wahrscheinlichkeitstheoretischen Zustandsschätzung sowie der kombinatorischen Optimierung, die in dieser Arbeit verwendet werden, werden in Kapitel 2 vorgestellt. Insbesondere werden der Bayes'sche Filter und seine Anwendung für die Roboterlokalisierung, Verfahren zur simultanen Lokalisierung und Kartenerstellung, das Problem des Handlungsreisenden, sowie das Mengenüberdeckungsproblem besprochen.

Viele Robotikanwendungen benötigen ein präzises Umgebungsmodell. In Kapitel 3 wird dafür ein Ansatz zur Rekonstruktion von 3D Objekten sowie der Erstellung von 3D Karten der Umgebung mittels RGB-D Kameras vorgestellt. Dieser basiert auf dem Registrieren visueller Merkmale von zu verschiedenen Zeitpunkten aufgenommenen Bilddaten. Auf Grund der registrierten Bildmerkmale und der dazugehörigen Tiefeninformationen ist es möglich, die lokale Bewegung der Kamera zwischen diesen Zeitpunkten zu schätzen. Im Falle der Kartierung der Umgebung mittels eines mobilen Roboters wird zusätzlich die Odometrieinformation für die Bewegungsschätzung verwendet. Die lokalen Bewegungsschätzungen bilden einen Graphen von Kameraposen, dessen Optimierung eine global konsistente Schätzung der Kameratrajektorie ergibt. Im letzten Schritt kann daraus eine 3D Repräsentation der Umgebung erzeugt werden. Eine ausführliche experimentelle Auswertung auf Referenzdatensätzen zeigt, dass der vorgestellte Algorithmus eine genaue Rekonstruktion der Kameratrajektorie in vielfältigen realen Umgebungen erlaubt. Die Experimente zeigen außerdem, dass die Verwendung von Odometrieinformationen eines mobilen Roboters die Genauigkeit, gerade in Umgebungen mit wenigen visuellen Merkmalen, häufig deutlich erhöhen kann.

Basierend auf einer 3D Punktwolkenrepräsentation der Oberfläche eines Gegenstandes wird in Kapitel 4 ein Ansatz zur Abdeckung komplexer Oberflächen für redundante Manipulationsroboter vorgestellt. Zunächst wird dabei die Oberfläche in lokale Ebenen diskretisiert. Auf diesen basierend wird ein Graph erstellt, der kollisionsfreie Pfade des Endeffektors des Roboters über die Oberfläche repräsentiert. Das Problem, einen Pfad durch diesen Graph zu finden, wird dann als verallgemeinertes Problem des Handlungsreisenden formuliert. Dies erlaubt, benutzerdefinierte Kostenfunktionen im Konfigurationsraum des Roboters während der Pfadplanung zu berücksichtigen und zu minimieren. Zusätzlich wird eine recheneffiziente hierarchische Approximation des generellen Ansatzes entwickelt. Einem redundanten Manipulationsroboter ermöglicht dieser Ansatz, Anwendungen wie etwa das Abstauben oder das Abwischen von Gegenständen mit komplizierten Oberflächenformen auszuführen. Dabei wird außerdem der benötigte Aufwand beziehungsweise die vom Roboter benötigte Zeit minimiert.

In bestimmten Fällen, klassischerweise etwa beim Entfernen eines Kaffeeflecks, muss nicht die gesamte Oberfläche gereinigt werden. In Kapitel 5 wird ein Ansatz vorgestellt, mit dem ein Manipulationsroboter lernen kann, unbekannte Oberflächen durch die Wahrnehmung der Ergebnisse seiner Aktionen zu reinigen. Dafür lernt der Roboter die Transitionsfunktion eines Markov-Entscheidungsprozesses. Dies ermöglicht ihm, eine Vorhersage über das Resultat der Reinigung zu machen und somit die zu schmutzigen Teile der Oberfläche zu erkennen. Des Weiteren kann das Ergebnis der Reinigung überprüft werden.

In Kapitel 6 wird diese Idee auf Staubsaugroboter übertragen. Einige dieser Roboter besitzen einen Sensor innerhalb der Saugeinheit, der den eingesaugten Schmutz messen kann. Der vorgestellte Ansatz zeigt, wie mit einem solchen Roboter die Verteilung des Schmutzes sowie deren Entwicklung über die Zeit mittels Poissonprozessen gelernt werden kann. Das ermöglicht es dem Roboter, Vorhersagen über die absolute Schmutzmenge in der Umgebung zu machen. Auf Basis dieser Vorhersagen werden zusätzlich zwei effiziente Reinigungsstrategien entwickelt. Die erste Strategie zielt darauf ab, den Schmutz in der gesamten Umgebung unter einen benutzerdefinierten Wert zu reduzieren und dabei die Reinigungszeit zu minimieren. Die zweite Strategie verringert den Schmutz in der Umgebung so weit wie möglich innerhalb einer vorgegebenen Reinigungszeit.

Preiswerte Reinigungsroboter für Privathaushalte sind häufig anfällig für hohe Mess- und Bewegungsunsicherheiten. Dies kann in einer größeren Abweichung der Positionsschätzung des Roboters resultierten, sodass angefahrene Zielpunkte nicht mit der nötigen Genauigkeit erreicht werden. Möglich ist außerdem, dass nicht der gesamte Schmutz im ersten Reinigungsdurchgang entfernt wird. In Kapitel 7 wird ein Verfahren entwickelt, das die Lokalisierung des Roboters sowie die Unsicherheit der Sensoren und Aktuatoren während der Ausführung des Reinigungsvorgangs explizit berücksichtigt. Dabei werden zunächst statistische Modelle für den Schmutzsensor sowie die Saugeinheit entwickelt. Ein neuer vollständig wahrscheinlichkeitstheoretischer Ansatz ermöglicht es, die gemeinsame Verteilung des Schmutzes in der Umgebung und der Position des Roboters zu schätzen. In einem letzten Schritt wird diese Schätzung verwendet, um den Reinigungspfad des Roboters während des Reinigungsvorgangs neu zu planen. Mittels dieses Ansatzes kann ein Staubsaugroboter den Schmutz in der Umgebung mit hoher Wahrscheinlichkeit unter einen durch den Benutzer festgelegten Wert reduzieren und dabei die Reinigungszeit minimieren. Das Ergebnis ist eine signifikant sauberere Umgebung im Vergleich zu bisherigen Verfahren. Die Arbeit schließt mit einem Fazit und einem Ausblick über mögliche zukünftige Ansätze in Kapitel 8.

Alle entwickelten Ansätze wurden sorgfältig ausgewertet und getestet. Dafür wurden umfangreiche Experimente in Simulationen und auf echten Daten durchgeführt. Die Experimente auf verschiedenen Roboterplattformen demonstrieren zusätzlich die Anwendbarkeit der vorgestellten Algorithmen.

Abstract

In the future, service robots are expected to perform many useful tasks in domestic environments such as cleaning or tidying up. If they performed these tasks autonomously, users would profit from a large amount of saved time. Furthermore, elderly or handicapped people could live more independent of care personnel.

Most currently available service robots address the task of floor cleaning and are often not very efficient. This thesis presents several innovative approaches, which make cleaning robots more efficient and extend their capabilities.

For many tasks in domestic environments, such as cleaning the floor or wiping a table, service robots require access to a precise model of the environment. We therefore investigate an approach to object reconstruction and 3D mapping for RGB-D cameras, which results in an accurate, globally consistent reconstruction of the camera trajectory. Given this trajectory, our system allows for computing different 3D environment representations such as voxel maps or point clouds. For robot mapping, we integrate the odometry information as an additional motion estimator. In extensive experiments on benchmark datasets, we show that our approach is highly accurate in real-world scenarios.

Using a 3D point cloud representation of an object as a prerequisite, we present techniques for cleaning 3D surfaces. Our algorithm for coverage path planning enables manipulation robots to perform tasks such as dusting. In addition, it considers cost functions in the joint space of the robot at the time of planning. As a result, a manipulation robot can cover complex 3D surfaces while minimizing the time or effort needed for the task.

A cleaning robot that knows which parts of the surface are dirty can focus on them and thus finish faster. We therefore propose a vision-based approach that allows a manipulation robot to identify the dirty parts of a surface by observing the outcome of its actions. As a result, the robot becomes more efficient and can verify its cleaning success.

We extend the idea of selectively cleaning the dirty parts to floor cleaning robots and present techniques for modeling and learning the distribution of dirt in the environment and its dynamics. This allows a robot to make predictions about the amount of dirt in the environment before it starts to clean. Furthermore, we propose a fully probabilistic model for jointly estimating the distribution of dirt as well as the pose of the robot during a cleaning run. This enables a vacuum cleaning robot to reduce the dirt in the environment below a user-defined threshold with high confidence. In addition, it minimizes the required cleaning time.

The techniques presented in this thesis were implemented and evaluated in simulation as well as with real-world data obtained from mobile robots or benchmark datasets. We performed extensive experiments to support our claims. The implementation of our approaches on several robotic platforms demonstrate their applicability in practice.

Acknowledgments

I would like to thank the many people without whose support this thesis would not have been possible. First of all, I thank my advisor Wolfram Burgard for giving me the opportunity to pursue my PhD, for the professional and enjoyable working environment and for the freedom to pursue my own ideas. I very much appreciated his support when finishing papers (even late at night), his encouragement in case of a rejection and the chance to visit interesting conferences and meet great people, which I enjoyed very much.

I would also like to thank the second corrector Bernhard Nebel for examining this thesis. Furthermore, I thank Matthias Teschner and Christian Schindelhauer for being on the examination committee.

A special thanks goes to my co-authors for the fruitful discussions, collaborations and paper writing. In particular, I would like to thank Felix Endres, Maximilian Beinhofer, Diego Tipaldi, Jürgen Sturm, Philipp Ruchti and Daniel Kuhner.

I thank Christoph Sprunk for his support of PGFPlots and Jörg Müller for his insights into C++. For his efforts in keeping the PR2 up and running and presenting it on numerous occasions with me, I thank Nichola Abdo. For proof reading early versions of this thesis, I thank Felix Endres, Christoph Sprunk, Andreas Lars Wachaja, Tobias Schubert, Henrik Kretzschmar, Jörg Röwekämper, Markus Kuderer, Michael Ruhnke, Nichola Abdo, Bastian Steder, Benjamin Suger, and Manfred Krapf.

I would like to thank all of my colleagues from the Autonomous Intelligent Systems Lab for the many scientific discussions about different research topics, the software they provided and the nice working atmosphere in general.

For administrative and technical help, I thank Michael Keser, Evelyn Rusdea, Susanne Bourjaillat, Bettina Schug, Dagmar Sonntag and Kristine Haberer.

Finally, I thank my wife Marina and my family for their support and love during this time and during my life in general.

This work has partially been supported by the German Research Foundation (DFG) within the Research Training Group 1103 under contract number SFB/TR 8 as well as through the Leibniz program. It has also been supported by the European Commission in the First-MM project under the contract number FP7-ICT-248258-First-MM and in the Squirrel project under contract number 621532.

To my wife and family.

Contents

Chapter 1

Introduction

Service robots are envisioned to perform many relevant functions in domestic environments in the future such as cleaning, tidying up or doing the laundry. These tasks typically consume large amounts of time in everyone's daily routine. If a robot could perform these tasks autonomously, this development would have a significant impact on our everyday life. Users of service robots would benefit from a substantial amount of saved time. In addition, service robots could enable elderly or physically handicapped people to stay longer in their own homes and live more autonomously, less dependent on daily care personnel.

Most commercially available service robots address the task of floor cleaning. In fact, robotic floor cleaning is the most popular robotic application in domestic environments today. In 2008 Prassler and Kosuge [120] listed thirteen commercially available floor cleaning robots, and by now they are sold as a mass product. The company iRobot, for example, sold more than 10 million home robots worldwide [73]. They are also popular in industry. One of the earliest navigation systems for autonomous floor cleaning robots that were targeted for cleaning at chain stores is the SINAS system, which started operation as early as 1996 [92].

Although floor cleaning robots are very common, most cleaning strategies currently implemented are not very efficient. Early vacuum cleaning robots mostly performed random movements. They could thus not guarantee that every part of the environment was visited. More recent models aim at covering the entire floor surface. This, however, takes a long time and is often unnecessary as only few spots may be dirty. Due to imperfect actuator units, covering an area once also does not mean that it has been thoroughly cleaned. The long running time of a robot results in high power consumption and further disturbance of the user from the noise of the robot. Often, the available cleaning time is also limited, e.g., if visitors are coming on short notice. In addition, available cleaning robots are mostly constrained to work on two-dimensional surfaces and are not able to dust or wipe furniture or decorations, for example.

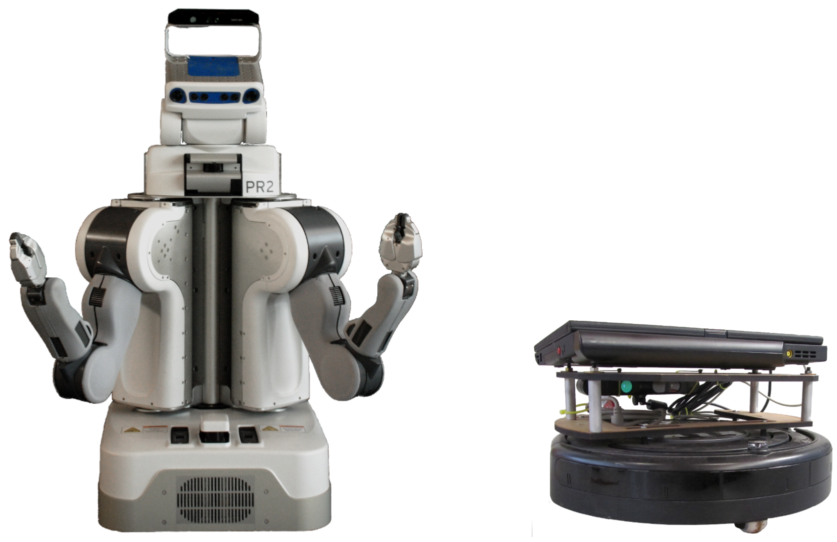
The goal of this thesis is to enable service robots to perform cleaning tasks efficiently and guarantee the user a clean environment. Beyond floor cleaning, we also aim at providing novel approaches for cleaning arbitrary surfaces with a manipulation robot.

Robot cleaning is a complex task. Its completion requires the robot to cope with many challenges. We illustrate these challenges and motivate our work with two typical tasks performed in every household: dusting and cleaning the floors. When dusting, one typically uses a towel or sponge. Dust is removed easily but one usually aims at covering the entire surface of an object. Typically, there are many objects with very different types of surfaces in a household environment. Tables, for example, often possess simple, largely planar surfaces while statues or vases may have complex free-form surfaces. For a manipulation robot the task of dusting poses several challenges. The robot requires a map of the environment, which it can use to localize in and plan collision free traveling paths to reach the object. For wiping a specific object, the robot needs to first acquire a model of the object surface, which can be arbitrarily complex. Second, it must plan a path such that the tool used covers the entire surface of the object and the robot avoids collisions. Furthermore, it should minimize the time needed. Other tasks, such as removing a stain left by a coffee mug from a counter top, for example, do not require cleaning the entire surface. If the robot uses its sensors and learns to identify parts that need to be cleaned, it becomes much more efficient and is able to verify its cleaning result.

The next task we consider is vacuuming an apartment. When vacuuming, one sometimes cleans the entire apartment but, more often, only specific parts. Some areas are cleaned very often to several times a day, the entrance area for example. Other areas, such as under the bed, may only need cleaning once a week or even less often. In addition, some types of dirt may be removed on the first trial, while other types require several sweeps, e.g., crumbs on a carpet. A robot that knows where the dirty parts are, can reduce its runtime and guarantee the user a clean home. A vacuum cleaning robot, however, can only observe the area in which it is currently located. Some vacuum cleaning robots possess a sensor for measuring the dirt currently cleaned. This sensor is located inside the cleaning unit of the robot. It can thus not directly observe the result of its cleaning actions on the environment but only the change it induced, namely the dirt cleaned. To guarantee a clean environment, a robot therefore needs an initial estimate about the location of the dirty areas. In addition it has to consider the uncertainty in its position as well as the uncertainty of its dirt sensor and actuator.

These considerations lead to the following research questions that we address in this thesis:

- How can a robot obtain an accurate model of its environment that it can use to check collisions against as well as use for path planning?



(a) PR2 (Willow Garage) **(b)** Roomba 560 (iRobot)

Figure 1.1.: Robots used for performing the practical experiments presented in this thesis.

- How can a manipulation robot plan a path for its tool that covers the entire surface of an object and minimizes the execution time?
- How can a robot learn to identify those parts of the environment that currently need cleaning and exploit this information to clean efficiently?
- How can a vacuum cleaning robot guarantee the user a clean environment?

In this thesis, we address these research questions and present solutions to them. To test and evaluate our approaches, we used two robotic platforms. The first robot is the PR2 from Willow Garage. It includes two seven degrees of freedom arms, an omni-directional base and a rich sensor configuration including two laser range finders, two stereo cameras and an RGB-D camera (see Figure 1.1a). The second robot is an iRobot Roomba 560 vacuum cleaning robot that includes a sensor for measuring the dirt cleaned inside its suction unit. The robot has been extended with an RGB-D camera for perceiving its environment (see Figure 1.1b). In experiments with these robots as well as in simulation, we show that our approaches enable service robots to effectively cover 3D surfaces, to identify the dirty areas, to clean these efficiently and to verify the results.

This thesis is organized as follows. In Chapter 2, we present the basic techniques from the fields of probabilistic state estimation and combinatorial optimization that we use throughout the thesis. In the following chapters, we present the scientific contributions of this thesis, which we detail in the next section. Finally, we give a conclusion and an outlook to future work in Chapter 8.

1.1. Key Contributions

In this thesis, we contribute innovative approaches to several fields of robotics research including mapping, coverage path planning, and state estimation. Each of them addresses one key challenge service robots have to overcome. With the goal of enabling robots to efficiently perform cleaning tasks, we combined techniques from robotics and combinatorial optimization. Our approaches enable robots to build consistent 3D models, estimate probabilistic models of actuators and sensors, learn the effects of their actions, and estimate dynamically changing properties of the environment. In summary, in this thesis, we provide these main contributions:

- a SLAM system for creating globally consistent 3D models of the environment with an RGB-D camera (Chapter 3),
- a general framework for coverage path planning on 3D surfaces, which allows for minimizing user defined cost functions in the joint space of a redundant manipulation robot (Chapter 4),
- a vision-based method that enables a manipulation robot to learn the effect of its actions on the environment (Chapter 5),
- a solution for modeling and estimating the dynamics of the generation of dirt in an environment (Chapter 6), which allows us to develop more efficient cleaning strategies,
- probabilistic models for the sensor and actuator of a vacuum cleaning robot and a probabilistic framework for jointly estimating its pose as well as the distribution of dirt in the environment (Chapter 7), which allows the robot to guarantee with high confidence that the dirt in the environment is reduced below a user-provided threshold.

1.2. Publications

Parts of this thesis were published in the following workshop, conference, and journal articles:

- J. Hess, M. Beinhofer, and W. Burgard. A probabilistic approach to high-confidence cleaning guarantees for low-cost cleaning robots. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 5600–5605, 2014

- J. Hess, M. Beinhofer, D. Kuhner, P. Ruchti, and W. Burgard. Poisson-driven dirt maps for efficient robot cleaning. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 2245–2250, 2013

- F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard. 3D mapping with an RGB-D camera. *IEEE Transactions on Robotics*, pages 177–187, 2 2014

- J. Hess, D. Tipaldi, and W. Burgard. Null space optimization for effective coverage of 3D surfaces using redundant manipulators. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 1923–1928, 2012

- F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evaluation of the RGB-D SLAM system. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 1691–1696, 2012

- F. Endres, J. Hess, N. Engelhard, J.Sturm, and W. Burgard. 6D visual SLAM for RGB-D sensors. *at - Automatisierungstechnik*, 60:270–278, 2012

- N. Engelhard, F. Endres, J. Hess, J. Sturm, and W. Burgard. Real-time 3D visual SLAM with a hand-held camera. In *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum*, 2011

- J. Hess, J. Sturm, and W. Burgard. Learning the state transition model to efficiently clean surfaces with mobile manipulation robots. In *Proc. of the Workshop on Manipulation under Uncertainty at the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011

1.3. Collaborations

Parts of this thesis resulted from collaborations with others, which we detail in this section.

- Chapter 3: The system for 3D mapping with an RGB-D camera is the result of a collaboration with Felix Endres and Nikolas Engelhard, supervised by Jürgen Sturm, Wolfram Burgard and Daniel Cremers. The idea and implementation of the basic system as presented in this thesis was a joint work of the author of this thesis and Felix Endres who both contributed equally. Solely the author's work are the sections about integrating the robot odometry into the estimation problem (see Section 3.2.4, Section 3.1.1.5). Felix Endres solely proposed the heuristic for selecting which frames to compare for detecting loop closures in Section 3.1.1.4. See Endres et al. [34, 35, 36], Engelhard et al. [37] for related publications.

- Chapter 4: The research on coverage path planning was supervised by Gian Diego Tipaldi and Wolfram Burgard. See Hess et al. [65] for the related publication.
- Chapter 5: The research on how a manipulation robot learns the effects of its actions was supervised by Jürgen Sturm and Wolfram Burgard. See Hess et al. [64] for a related workshop publication.
- Chapter 6: The concept of how a robot learns the dynamics of dirt in the environment over time was developed in cooperation with Maximilian Beinhofer, Daniel Kuhner, and Philipp Ruchti and supervised by Wolfram Burgard. Daniel Kuhner and Philipp Ruchti contributed by conducting preliminary experiments in the context of a Master team project that was co-supervised by Maximilian Beinhofer and the author of this thesis. The results presented in this thesis as well as the corresponding publication (see Hess et al. [66]) are based on an entirely new and substantially different implementation.
- Chapter 7: The joint estimation of the robot pose and the distribution of dirt in the environment is the result of a collaboration with Maximilian Beinhofer, supervised by Wolfram Burgard. Maximilian Beinhofer contributed to the derivation of the sensor and actuator model of the robot. See Hess et al. [67] for the related publication.

1.4. Notation

The following table states the symbols and notations used in this work.

Symbol	Meaning
a, b	Scalar values
$\mathbf{a}, \mathbf{b}$	Column vectors
A, B	Matrices
I	Identity matrix
$\mathcal{A}, \mathcal{B}$	Sets
$\mathbf{a}^\top, B^\top$	Transpose of a vector or matrix
$\|\mathbf{a}\|$	L_2 norm of vector $\mathbf{a}$
$\mathrm{tr}(A)$	Trace of a matrix A
$\det(A)$	Determinant of a matrix A
$p(a)$	Probability density function evaluated at a
$p(a \mid b)$	Conditional probability density function at a conditioned on b
$\hat{a}$	Estimated value of a
a^*	Optimal value of a
$\mathbb{E}[X]$	Expected value of a random variable X
$Var(X)$	Variance of a random variable X
$\mathcal{N}(\mu, \Sigma)$	Gaussian distribution with mean μ and variance Σ
$\{\cdots\}$	Set
$\langle\cdots\rangle$	Tuple (ordered set)
$(\cdots)$	Row vector
$x_{0:t}$	Sequence of states x from time 0 to time t, i.e., $x_0, \dots, x_t$

Chapter 2

Basic Techniques

In this chapter, we describe the basic techniques from robotics and the field of combinatorial optimization that we apply throughout this thesis. In the first section, we cover the basics from probability theory. We then describe recursive Bayesian state estimation using the Bayes' filter and discuss how to apply its particle filter implementation to robot localization in Section 2.2. In Section 2.3, we present the general SLAM problem and briefly introduce nonlinear least-squares optimization and its application to pose graph optimization. We detail the traveling salesman problem and its generalized version in Section 2.4 and finally present the set cover problem and the greedy algorithm that is commonly used to query for a solution in Section 2.5.

2.1. Probability Theory

In this section, we briefly introduce basic definitions and formulations from probability theory that build the basis of the probabilistic models used in this thesis. For describing uncertainties and random processes, we make use of random variables. Let (Ω, Σ, P) be a probability space, where Ω denotes the set of distinct outcomes, Σ a σ-algebra, a set of events defined on the space Ω, and P a probability measure. A probability measure is a function $P : \Sigma(\Omega) \to [0,1]$ that assigns a probability between zero and one to each event. A random variable X is then a measurable function $X : \Omega \to \mathcal{X}$ that maps the set of possible outcomes Ω to the measurable space $\mathcal{X}$, which can be discrete or continuous. If the random variable is discrete, a probability $P(X = x) = P(X^{-1}(x))$ is associated to each value $x \in \mathcal{X}$. These probabilities sum to 1,

$$\sum_{x \in \mathcal{X}} P(X = x) = 1. \tag{2.1}$$

If it is clear from the context, we use the abbreviation $p(x) := p(X = x)$ in this thesis. In the following, we consider continuous random variables. If a random variable X is absolutely continuous with respect to the Lesbegue measure, it is fully described by its probability density function. For each $x \in \mathcal{X}$, the probability density function takes on the value $p(x)$. It always integrates to 1:

$$\int_{x \in \mathcal{X}} p(x)\,dx = 1. \tag{2.2}$$

Let D be a subset of $\mathcal{X}$. The probability that the outcome of a random variable X lies in set D is then given as the integral

$$P(X \in D) = \int_{x \in D} p(x)\,dx. \tag{2.3}$$

For two random variables X and Y, their joint distribution is denoted as

$$p(x,y). \tag{2.4}$$

They are called independent, if their joint distribution is given as

$$p(x,y) = p(x)\,p(y). \tag{2.5}$$

If they are not independent, then knowledge about the value of Y influences the value of X. This fact is denoted by a conditional probability distribution. For $p(y) > 0$, it is defined as

$$p(x|y) = \frac{p(x,y)}{p(y)}. \tag{2.6}$$

In this thesis, we apply the following properties and rules that make use of conditional probability distributions.

Marginalization

$$p(x) = \int p(x,y)dy. \tag{2.7}$$

Law of Total Probability

$$p(x) = \int p(x|y)p(y)dy. \tag{2.8}$$

Bayes' Theorem In robotics, the posterior $p(x|y)$ can often not be computed directly. If $p(y) > 0$, Bayes' theorem allows for computing this quantity as

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}. \tag{2.9}$$

For continuous valued random variables, $p(y)$ is given as

$$p(y) = \int p(y|x)p(x)\,dx. \tag{2.10}$$

If the value of y is known, we can simplify Bayes' theorem by replacing $1/p(y)$ by a normalizer η. It is then written as

$$p(x|y) = \eta\, p(y|x)p(x). \tag{2.11}$$

Markov Property The Markov property in time-discrete stochastic systems denotes the assumption that the future state x_t at time t only depends on the present state x_{t-1} and is independent of the entire state sequence preceding it:

$$p(x_t|x_0,\ldots,x_{t-1}) = p(x_t|x_{t-1}). \tag{2.12}$$

Gaussian Distribution A multivariate Gaussian distribution over a vector valued random variable $\mathbf{X}$ in $\mathbb{R}^n$ is denoted as

$$\mathbf{X} \sim \mathcal{N}(\mu, \Sigma), \tag{2.13}$$

where $\mu \in \mathbb{R}^n$ denotes the mean vector and $\Sigma \in \mathbb{R}^{n\times n}$ is symmetric positive definite and also called the covariance matrix of the multivariate Gaussian. Its probability density function takes the following form:

$$p(\mathbf{X} = \mathbf{x}) = ((2\pi)^n \det(\Sigma))^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^\top \Sigma^{-1}(\mathbf{x}-\mu)\right), \tag{2.14}$$

where $\det(\Sigma)$ denotes the determinant of matrix Σ.

Binomial Distribution The binomial distribution is a discrete probability distribution that models the number of successes in a series of independent Bernoulli trials. A Bernoulli trial is a random experiment with two possible outcomes, success or failure. Let $p \in (0,1)$

be the probability of success of a single trial. The probability for k successes in n independent trials is then given as

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}. \tag{2.15}$$

The expected value of a discrete random variable X under a binomial distribution is $\mathbb{E}[X] = np$ and its variance is $Var(X) = (np(1-p))$. The special case of a binomial distribution with $n = 1$ is also called a Bernoulli distribution.

Poisson Distribution The Poisson distribution is a discrete probability distribution that models the number of events that occur independently of each other at a constant average rate in a fixed time interval. The probability that a specific number k of events occur is given as

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}, \tag{2.16}$$

with $\lambda > 0$. The parameter λ is equal to the expected value as well as the variance of the distribution, i.e., $\mathbb{E}[X] = \lambda$, $Var(X) = \lambda$.

2.2. Recursive State Estimation

In this section, we introduce recursive Bayesian state estimation in general as well as its particle filter implementation [145]. We furthermore describe how the particle filter can be used for localizing mobile robots in their environment.

2.2.1. The Bayes' Filter

The Bayes' filter is a probabilistic technique that recursively estimates the state of a dynamical stochastic system over time. It is a time-discrete process that explicitly models the uncertainties involved.

For the derivation, we refer to the following example of a mobile robot moving through an environment. At each time step t, the robot is steered by a control command $\mathbf{u}_t$ and perceives the environment with a noisy measurement $\mathbf{z}_t$. The Bayes' filter then estimates a probability density function of the state $\mathbf{x}_t$ of the system. Concretely, it estimates the

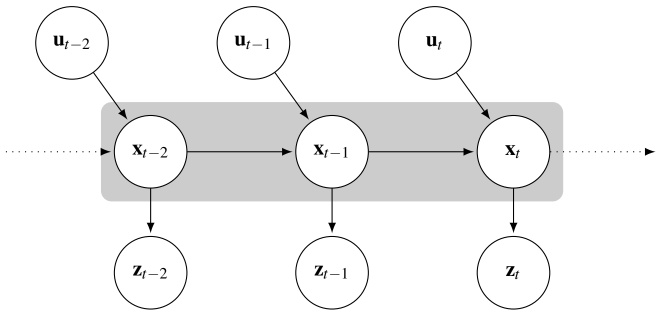


Figure 2.1.: Graphical model of the dynamical system describing the evolution of the state variable **x**. The nodes in the shaded area are the hidden nodes to be estimated. The other nodes correspond to variables that can be observed, the stochastic controls **u** and measurements **z**. The arrows denote the stochastic dependencies between the random variables.

posterior distribution of $\mathbf{x}_t$, also called belief $bel(\mathbf{x}_t)$, conditioned on all previously obtained measurements $\mathbf{z}_{1:t}$ and controls $\mathbf{u}_{1:t}$, that is

$$bel(\mathbf{x}_t) = p(\mathbf{x}_t|\mathbf{u}_{1:t},\mathbf{z}_{1:t}). \tag{2.17}$$

The dynamic Bayesian network corresponding to the described dynamical system is shown in Figure 2.1. This graphical model visualizes the random variables involved in the process as nodes and the assumed stochastic dependencies as arrows. The shaded area marks the variables that are unknown and cannot be measured directly, i.e., the states **x** of the system that are estimated. As one can see, the Markov assumption is encoded into the graphical model, the assumption being that the state at time t can be computed using the previous state $\mathbf{x}_{t-1}$ and the control $\mathbf{u}_t$. The measurement $\mathbf{z}_t$ is only generated by the state $\mathbf{x}_t$. A system that is modeled as a Markov process with hidden (unobserved) states is also called a hidden Markov model (HMM) [127].

Using Bayes' rule and applying the rules of d-separation [9] to the graphical model (see Figure 2.1), the following recursive update scheme can be derived from Eq. 2.17 as shown in Thrun et al. [145, p. 31–32]:

$$p(\mathbf{x}_t|\mathbf{z}_{1:t},\mathbf{u}_{1:t}) = \eta \quad \underbrace{p(\mathbf{z}_t|\mathbf{x}_t)}_{\text{Measurement Model}} \quad \int \underbrace{p(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{u}_t)}_{\text{Transition Model}} \underbrace{p(\mathbf{x}_{t-1}|\mathbf{z}_{1:t-1},\mathbf{u}_{1:t-1})}_{\text{Prior Belief } bel(x_{t-1})} d\mathbf{x}_{t-1}. \tag{2.18}$$

In this expression, η denotes a normalizing constant that ensures that the posterior obtained is a probability distribution, namely that $\int p(\mathbf{x}_t|\mathbf{z}_{1:t},\mathbf{u}_{1:t})\, d\mathbf{x}_t = 1$. To explicitly take the uncertainties of the control and the measurement into account, they are described as

Algorithm 1: Bayes' filter

```
Input: $bel(\mathbf{x}_{t-1}), \mathbf{u}_t, \mathbf{z}_t$
Output: $bel(\mathbf{x}_t)$
1 $\forall \mathbf{x}_t : \overline{bel(\mathbf{x}_t)} \leftarrow \int p(\mathbf{x}_t|\mathbf{z}_{t-1}, \mathbf{u}_t, \mathbf{x}_{t-1})\, bel(\mathbf{x}_{t-1})\, d\mathbf{x}_{t-1}$    (Prediction)
2 $\forall \mathbf{x}_t : bel(\mathbf{x}_t) \leftarrow \eta\, p(\mathbf{z}_t|\mathbf{x}_t)\, \overline{bel(\mathbf{x}_t)}$    (Correction)
3 return $bel(\mathbf{x}_t)$
```

probability distributions. The transition model is a probability distribution of the predicted state $\mathbf{x}_t$ given the previous state $\mathbf{x}_{t-1}$ and the control command $\mathbf{u}_t$:

$$p(\mathbf{x}_t|\mathbf{u}_t, \mathbf{x}_{t-1}). \tag{2.19}$$

Likewise, the measurement model is a probability distribution over the measurement $\mathbf{z}_t$ given the state estimate $\mathbf{x}_t$:

$$p(\mathbf{z}_t|\mathbf{x}_t). \tag{2.20}$$

The general Bayes' filter algorithm that implements the update equation (see Eq. 2.18) is given in Algorithm 1. It performs the following two steps to calculate the new belief $bel(\mathbf{x}_t)$ of the distribution of the state variable:

Prediction An intermediate belief $\overline{bel(\mathbf{x}_t)} = p(\mathbf{x}_t|\mathbf{u}_{1:t}, \mathbf{z}_{1:t-1})$ about the new state $\mathbf{x}_t$ is predicted by applying the control $\mathbf{u}_t$ to the previous belief $bel(\mathbf{x}_{t-1})$ in line 1.

Correction The belief $\overline{bel(\mathbf{x}_t)}$ about the state is reshaped according to the likelihood of the measurement $\mathbf{z}_t$ taken from the estimated state $\mathbf{x}_t$ in line 2. This step results in the final estimate $bel(\mathbf{x}_t)$.

As it is stated here, the Bayes' filter algorithm is very general as we have not detailed the state space or the type of probability distributions used or specified the physical models underlying the transition and measurement function.

There are several parametric and nonparametric implementations of the Bayes' filter known in the literature. Parametric filters represent the posterior distribution in a functional form, such as a Gaussian, that has a fixed number of parameters. In contrast, nonparametric filters approximate the posterior with a finite number of discrete values [145]. One of the earliest parametric implementations is the Kalman filter [79]. It assumes a linear Gaussian system, i.e., a linear transition and measurement function with Gaussian distributed noise. In this case, the integral in line 1 of the Bayes' filter algorithm can be computed in closed form and the Kalman filter results in the exact posterior distribution [79]. Although a large variety of problems can be solved using the Kalman filter, the assumption of a linear Gaussian system often does not hold for mobile robot localization. Robot motion models, for example, are often nonlinear. In the next section, we introduce the particle filter, a nonparametric implementation of the Bayes' filter that we apply for robot localization in Chapter 6 and Chapter 7. We chose the particle filter for robot localization as it can deal with non-Gaussian probability distributions. In the literature it is also widely used for mobile robot localization in grid maps [144].

2.2.2. The Particle Filter

The key idea of the particle filter [31, 145] is to represent the posterior distribution $p(\mathbf{x}_t)$ by a set of discrete weighted samples, called particles [145]:

$$\mathcal{X}_t = \left\{ \langle \mathbf{x}_t^{[1]}, w_t^{[1]} \rangle, \langle \mathbf{x}_t^{[2]}, w_t^{[2]} \rangle, \cdots, \langle \mathbf{x}_t^{[N]}, w_t^{[N]} \rangle \right\}. \tag{2.21}$$

Each particle holds a concrete instantiation $\mathbf{x}_t^{[n]}$ of the state $\mathbf{x}_t$ as well as an importance weight $w_t^{[n]} > 0$. The sum of all importance weights is one.

The particle filter is an implementation of the Bayes' filter and thus recursively estimates the posterior distribution $p(\mathbf{x}_t)$ of the state $\mathbf{x}_t$ at time t given the prior distribution $p(\mathbf{x}_{t-1})$. The posterior distribution is also called target distribution in the context of particle filtering. As these distributions are represented as particle sets, the filter generates a new particle set $\mathcal{X}_t$ of the same size from the set $\mathcal{X}_{t-1}$ at each time step. In general, we would compute the new particle set by directly sampling from the posterior distribution. In robotic applications, however, this is often not possible. In this thesis, we therefore apply the sampling importance resampling (SIR) particle filter [31, 124]. It allows us to compute a particle set distributed according to the posterior distribution, while only being able to obtain samples from a different distribution $q(\mathbf{x}_t)$, called proposal distribution. The SIR filter performs the following steps to update the particle set $\mathcal{X}_{t-1}$:

Sampling We first propagate each particle in the set $\mathcal{X}_{t-1}$ by sampling from the proposal distribution $q(\mathbf{x}_t)$. This step corresponds to the prediction step of the Bayes' filter. The updated particles form the set $\mathcal{X}_t'$.

Importance Resampling For each particle in the set $\mathcal{X}_t'$ we compute an importance weight $w_t^{[n]}$. The weight accounts for the difference between the target and the proposal distribution and is given as

$$w_t^{[n]} = \frac{p(\mathbf{x}_t^{[n]})}{q(\mathbf{x}_t^{[n]})}. \tag{2.22}$$

Thereby it is assumed that the proposal distribution has the same support as the target distribution, i.e., $p(\mathbf{x}_t) > 0 \implies q(\mathbf{x}_t) > 0$. The set $\mathcal{X}_t$ is created by drawing particles with replacement from the particle set $\mathcal{X}'_t$ in proportion to their weight $w_t^{[n]}$. This is also called importance sampling [51]. The goal is to ensure that the density of the particle set is related to and thus represents the posterior distribution. This step for updating the particle set by resampling corresponds to the correction part of the Bayes' filter.

In this thesis, we apply the particle filter to mobile robot localization. In this context, the particle filter is also known as the Monte Carlo localization [27]. Monte Carlo localization estimates the posterior over the full trajectory of the robot:

$$p(\mathbf{x}_{0:t} | \mathbf{u}_{1:t}, \mathbf{z}_{1:t}). \tag{2.23}$$

We thus extend each particle to hold not only its current pose but also its entire history:

$$\mathbf{x}_{0:t}^{[n]} = \left\{ \mathbf{x}_0^{[n]}, \cdots, \mathbf{x}_t^{[n]} \right\}. \tag{2.24}$$

To derive the computation of the importance weights, we proceed similar to Thrun et al. [145, p. 103–104]. As we can usually not directly sample the target distribution, we factorize it as follows:

$$\begin{aligned} p(\mathbf{x}_{0:t} | \mathbf{u}_{1:t}, \mathbf{z}_{1:t}) &= \eta \, p(\mathbf{z}_t | \mathbf{x}_t) \, p(\mathbf{x}_{0:t} | \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}) \\ &= \eta \, p(\mathbf{z}_t | \mathbf{x}_t) \, p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{u}_t) \, p(\mathbf{x}_{0:t-1} | \mathbf{u}_{1:t-1}, \mathbf{z}_{1:t-1}). \end{aligned} \tag{2.25}$$

In the factorization above, we first use Bayes' rule, factoring out the measurement $\mathbf{z}_t$, and then apply the Markov property, observing the fact that the current measurement only depends on the current state $\mathbf{x}_t$. In a second step, we again apply Bayes' rule on the state $\mathbf{x}_t$ and again observe the Markov property. For deriving the weight of each particle we also have to choose the proposal distribution. This is a crucial step in particle filtering that can substantially influence its performance. For mobile robot localization it is common to choose the motion model together with the recursive term as proposal distribution as suggested by Dellaert et al. [27]:

$$q(\mathbf{x}_t | \mathbf{u}_{1:t}, \mathbf{z}_{1:t}) = p(\mathbf{x}_t | \mathbf{x}_{0:t-1}, \mathbf{u}_{1:t}) p(\mathbf{x}_{0:t-1} | \mathbf{u}_{1:t-1}, \mathbf{z}_{1:t-1}). \tag{2.26}$$

Algorithm 2: Particle filter

```
Input: $\mathcal{X}_{t-1}, \mathbf{u}_t, \mathbf{z}_t$
Output: $\mathcal{X}_t$
1 $\mathcal{X}_t = \mathcal{X}'_t = \emptyset$
2 for $n \leftarrow 1$ to $N$ do
3     sample $\mathbf{x}_t^{[n]} \sim p(\mathbf{x}_t | \mathbf{u}_t, \mathbf{x}_{t-1}^{[n]})$
4     $w_t^{[n]} = p(\mathbf{z}_t | \mathbf{x}_t^{[n]})$
5     $\mathcal{X}'_t \leftarrow \mathcal{X}'_t \cup \langle \mathbf{x}_t^{[n]}, w_t^{[n]} \rangle$
6 for $n \leftarrow 1$ to $N$ do
7     draw $\mathbf{x}_t^{[i]}$ with probability $\propto w_t^{[i]}$ from the set $\mathcal{X}'_t$
8     $\mathcal{X}_t \leftarrow \mathcal{X}_t \cup \langle \mathbf{x}_t^{[i]}, 1/N \rangle$
9 return $\mathcal{X}_t$
```

We can then derive the weight for each particle using Eq. 2.25 and Eq. 2.26 as follows:

$$\begin{aligned} w_t^{[n]} &= \frac{p(\mathbf{x}_{0:t}^{[n]} | \mathbf{u}_{1:t}, \mathbf{z}_{1:t})}{q(\mathbf{x}_{0:t}^{[n]} | \mathbf{u}_{1:t}, \mathbf{z}_{1:t})} \\ &= \frac{\eta\, p(\mathbf{z}_t | \mathbf{x}_t^{[n]})\, p(\mathbf{x}_t^{[n]} | \mathbf{x}_{t-1}^{[n]}, \mathbf{u}_t)}{p(\mathbf{x}_t^{[n]} | \mathbf{x}_{t-1}^{[n]}, \mathbf{u}_t)} \, \frac{p(\mathbf{x}_{0:t-1}^{[n]} | \mathbf{u}_{1:t-1}, \mathbf{z}_{1:t-1})}{p(\mathbf{x}_{0:t-1}^{[n]} | \mathbf{u}_{1:t-1}, \mathbf{z}_{1:t-1})} \\ &\propto p(\mathbf{z}_t | \mathbf{x}_t^{[n]}). \end{aligned} \tag{2.27}$$

As a result, the weight of each particle is proportional to the probability of the measurement given the current state.

The entire particle filter algorithm is given in Algorithm 2. It takes the current particle set $\mathcal{X}_{t-1}$ as well as the current measurement $\mathbf{z}_t$ and control $\mathbf{u}_t$ as input. In line 3, the algorithm propagates the state $\mathbf{x}_t^{[n]}$ of the n-th particle by sampling from the proposal distribution. For robot localization, we sample from the robot motion model given the control command $\mathbf{u}_t$. In line 4, we compute the weight of each particle. We neglect the normalizer η in the weight computation as we resample the particles proportional to their weights. Thus, after computing the weight for each particle, we again normalize all weights to sum to 1. The resampling step (line 6 to line 8) generates the new particle set $\mathcal{X}_t$ by drawing particles with replacement from the set $\mathcal{X}'_t$ according to their weight. This step concludes the algorithm.

As the posterior distribution is represented by particles, the accuracy of the filter increases with the number of particles used. However, as one can see in Algorithm 2, the computational complexity of the filter directly depends on the number of particles. If we assume a fixed particle density for representing the posterior, and increase the dimensions of the state space, the number of particles increases exponentially with the dimensionality of the state space. The complexity of the particle filter algorithm also depends on the implementation of the resampling strategy. In our work, we use *low variance sampling*, which requires $\mathcal{O}(N)$ time for sampling N particles [145, p. 109–110].

We use the particle filter described here for range-based localization of a vacuum cleaning robot in Chapter 6. In Chapter 7, we extend the particle filter to estimate the current state of the dirt in the environment in addition to the pose of the mobile robot.

2.3. Graph-Based Simultaneous Localization and Mapping

In this section, we describe the problem of estimating a model of the environment and the state of a dynamical system at the same time. In the context of mapping with a mobile robot, this is known as simultaneous localization and mapping (SLAM). Specifically, we provide the probabilistic formulation of SLAM that was introduced by Lu and Milios [98] and show how this problem can be modeled as a least-squares optimization problem. Thereby, we largely follow the derivation provided in Thrun et al. [145, p. 353–354].

For the derivation, we extend the example of a mobile robot moving through an environment we used in the previous sections. We aim at estimating the map $\mathbf{m}$ that consists of a set of n discrete landmark positions $\mathbf{m_1} \ldots \mathbf{m_n}$ that the robot can observe. The robot is steered by control commands $\mathbf{u}_t$ and makes an observation $\mathbf{z}_t$ to obtain a noisy measurement of a subset of landmarks at each time t. We additionally assume that the data association is given, i.e., that we know which measurement corresponds to which landmark.

The goal of SLAM is to estimate the posterior distribution of a state sequence and the map

$$p(\mathbf{x}_{0:t}, \mathbf{m} | \mathbf{u}_{1:t}, \mathbf{z}_{1:t}), \tag{2.28}$$

where $\mathbf{x}$ is the state variable, $\mathbf{m}$ is the map, $\mathbf{u}_{1:t}$ are the controls, and $\mathbf{z}_{1:t}$ are the measurements as in the previous section. In the context of SLAM, this posterior is also called full SLAM posterior as it estimates the full trajectory and not only the current state. The corresponding graphical model we consider is shown in Figure 2.2. The shaded areas again mark the unknown variables, the states of the system and the map to be estimated.

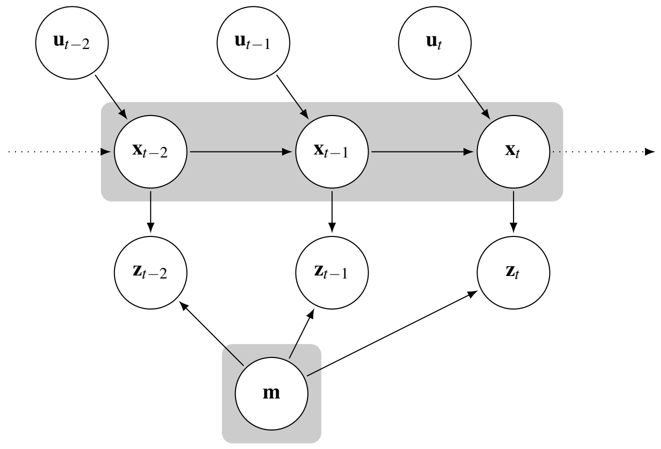


Figure 2.2.: Graphical model of the dynamical system describing the evolution of the state variable $\mathbf{x}$. The nodes in the shaded area are the hidden nodes to be estimated. The other nodes mark variables that can be observed, i.e., the stochastic controls $\mathbf{u}$ and measurements $\mathbf{z}$. Compared to the Bayes' filter, the map $\mathbf{m}$ has to be estimated in addition to estimating the state of the system.

We factor the SLAM posterior to derive a recursive definition similar to the Bayes' filter and the particle filter in Section 2.2.1 and Section 2.2.2 as follows:

$$\begin{aligned}
&p(\mathbf{x}_{0:t}, \mathbf{m} | \mathbf{u}_{1:t}, \mathbf{z}_{1:t}) \\
&= \eta\, p(\mathbf{z}_t | \mathbf{x}_t, \mathbf{m})\, p(\mathbf{x}_{0:t}, \mathbf{m} | \mathbf{z}_{1:t-1}, \mathbf{u}_{1:t}) \\
&= \eta\, p(\mathbf{z}_t | \mathbf{x}_t, \mathbf{m})\, p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{u}_t)\, p(\mathbf{x}_{0:t-1}, \mathbf{m} | \mathbf{z}_{1:t-1}, \mathbf{u}_{1:t-1}). \qquad (2.29)
\end{aligned}$$

In the first step, we apply Bayes' rule on the posterior and observe the Markov assumption, that the measurement $\mathbf{z}_t$ is independent of all other variables given the map $\mathbf{m}$ and the current state $\mathbf{x}_t$. In the second step, we decompose $\mathbf{x}_{0:t}$ into $\mathbf{x}_t$ and $\mathbf{x}_{0:t-1}$ and again apply the Markov assumption.

In the following, we assume that we do not have prior knowledge about the map $\mathbf{m}$. By induction over t, we then can derive the following expression from Eq. 2.29 [145, p. 354]:

$$p(\mathbf{x}_{0:t}, \mathbf{m} | \mathbf{u}_{1:t}, \mathbf{z}_{1:t}) = \eta\, p(\mathbf{x}_0) \prod_t p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{u}_t)\, p(\mathbf{z}_t | \mathbf{x}_t, \mathbf{m}), \qquad (2.30)$$

where $p(\mathbf{x}_0)$ describes the initial state in the global coordinate system. For SLAM, the initial robot pose $\mathbf{x}_0$ is often initialized to the zero, but it could also be a pose in an already available partial map. The factors $p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{u}_t)$ and $p(\mathbf{z}_t | \mathbf{x}_t, \mathbf{m})$ denote the probabilistic state transition and measurement model as used in the previous sections. We assume that they are Gaussians.

Let $h(\mathbf{x}_t, \mathbf{m})$ be a deterministic function that computes the measurement of the robot as expected from the current robot pose. Accordingly, let $g(\mathbf{u}_t, \mathbf{x}_{t-1})$ be a deterministic function that computes the new pose of the robot given the pose at time $t-1$ and the current control command $\mathbf{u}_t$. Using the definition of the Gaussian distribution (see Eq. 2.14), we have

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t) = \eta \exp\left(-\frac{1}{2}(\mathbf{x}_t - g(\mathbf{u}_t, \mathbf{x}_{t-1}))^\top R_t^{-1}(\mathbf{x}_t - g(\mathbf{u}_t, \mathbf{x}_{t-1}))\right) \tag{2.31}$$

$$p(\mathbf{z}_t|\mathbf{x}_t, \mathbf{m}) = \eta \exp\left(-\frac{1}{2}(\mathbf{z}_t - h(\mathbf{x}_t, \mathbf{m}))^\top Q_t^{-1}(\mathbf{z}_t - h(\mathbf{x}_t, \mathbf{m}))\right) \tag{2.32}$$

$$p(\mathbf{x}_0) = \eta \exp\left(-\frac{1}{2}\mathbf{x}_0^\top G^{-1}\mathbf{x}_0\right), \tag{2.33}$$

where R_t, Q_t, and G are the corresponding covariance matrices.

In contrast to Bayesian filtering techniques that estimate the full posterior distribution (see Section 2.2.1), graph-based SLAM aims for a point estimate, the maximum of the posterior distribution. We can obtain the maximum-a-posterior (MAP) estimate that incorporates prior information as

$$\mathbf{x}_{0:t}^* = \operatorname*{arg\,max}_{\mathbf{x}}\, p(\mathbf{x}_{0:t}, \mathbf{m}|\mathbf{u}_{1:t}, \mathbf{z}_{1:t}). \tag{2.34}$$

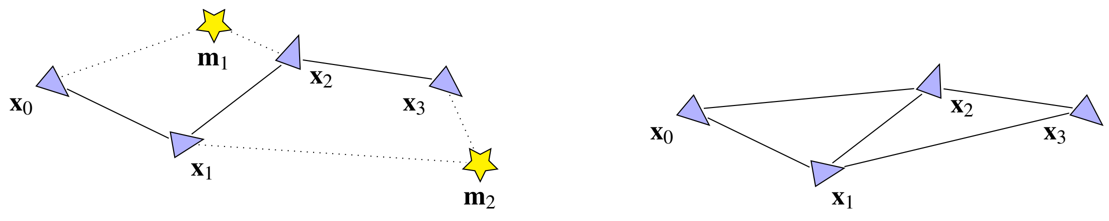
As we aim for computing the argument that maximizes the posterior distribution and not the exact likelihood, we can rewrite the equation to minimize the negative log-likelihood:

$$\mathbf{x}_{0:t}^* = \operatorname*{arg\,min}_{\mathbf{x}}(-\log p(\mathbf{x}_{0:t}, \mathbf{m}|\mathbf{u}_{1:t}, \mathbf{z}_{1:t})). \tag{2.35}$$

Taking the log of Eq. 2.30 and considering the transition and measurement model with Gaussian noise leads to

$$\begin{aligned}
&\log p(\mathbf{x}_{0:t}, \mathbf{m}|\mathbf{u}_{0:t}, \mathbf{z}_{0:t}) \\
&\qquad = \log \eta + \log p(\mathbf{x}_0) + \sum_t \left(\log p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t) + \log p(\mathbf{z}_t|\mathbf{x}_t, \mathbf{m})\right) \\
&\qquad = c + \frac{1}{2}\Big(\mathbf{x}_0^\top G^{-1}\mathbf{x}_0 + \sum_t \big((\mathbf{x}_t - g(\mathbf{u}_t, \mathbf{x}_{t-1}))^\top R_t^{-1}(\mathbf{x}_t - g(\mathbf{u}_t, \mathbf{x}_{t-1}) \\
&\qquad + (\mathbf{z}_t - h(\mathbf{x}_t, \mathbf{m})^\top Q_t^{-1}(\mathbf{z}_t - h(\mathbf{x}_t, \mathbf{m})\Big),
\end{aligned} \tag{2.36}$$

where c is a constant that accounts for all normalizer terms. In general, the problem of minimizing Eq. 2.36 is called least-squares optimization. If the functions $g(\cdot)$ and



(a) General Graph **(b)** Pose Graph

Figure 2.3.: Graphs used for representing the SLAM problem. (a) A general graph representation. The robot poses are marked by triangles while the features in the map, also called landmarks, are depicted as stars. Constraints between states of respective robot poses are denoted as solid lines. Constraints between poses and landmarks are marked by dotted lines. (b) The corresponding pose graph obtained by marginalizing out the landmarks.

$h(\cdot)$ are linear transformations and the noise of both models is Gaussian distributed, the estimation problem can be solved in closed form and is called linear least-squares. In robot mapping, however, we are often confronted with solving a nonlinear version of the least-squares problem due to the related rotational measurements of landmarks or robot poses, for example.

The least-squares problem stated in Eq. 2.36 can also be viewed and represented as a graph optimization problem. The corresponding graph contains two types of nodes, the robot poses as well as the poses of the landmarks. An example of such a graph is shown in Figure 2.3a, which visualizes the robot poses as triangles and the landmarks as stars. The edges reflect the two types of constraints introduced above. The solid edges correspond to constraints obtained from the robot motion and relate poses while the dotted edges relate robot poses to observations. Each observation of a landmark, introduces a constraint into the graph.

For real-world mapping problems this introduces a very large number of constraints, which makes the overall problem hard to solve. Several techniques have been developed to reduce the graph size and thus the number of constraints. Thrun and Montemerlo [143], for example, showed how variable elimination techniques can be used to remove constraints that connect robot poses to observations. They reduce the general graph to a pose graph, a graph that only connects robot poses and thus reduces the number of variables of the optimization problem. Figure 2.3b shows an example of a pose graph .

In contrast to generating constraints to measurements and then marginalizing them out as in Thrun and Montemerlo [143], one can also directly compute constraints between poses. In the context of SLAM with a mobile robot, scan-matching is often used to generate these constraints [15, 98, 114, 132].

Graph-based SLAM can therefore be seen as a two-step process. In the first step, the graph is constructed of robot poses that are related by measurements of the robot movement or the landmarks in the environment. In the second step, the graph is optimized by framing the optimization problem as a nonlinear least-squares problem. The software components realizing these two steps are often referred to as *front-end* and *back-end*.

In the following, we use a simplified notation for stating the nonlinear least-squares estimation problem represented by a pose graph. Let $\mathcal{C}$ be a set of tuples where each tuple $\langle i, j\rangle$ denotes that node i is connected to node j by an edge in the pose graph. Furthermore, let $\mathbf{x}_i$ and $\mathbf{x}_j$ be two poses in the pose graph corresponding to the nodes i and j. They are connected by the constraint $\mathbf{y}_{\langle i,j\rangle}$, i.e., the pose transformation between the poses as estimated by the front-end. We denote the inverse covariance matrix of the constraint $\mathbf{y}_{\langle i,j\rangle}$ as $\Omega_{\langle i,j\rangle}$, which is also called an information matrix. In addition, we define the vector valued function $\mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{y}_{\langle i,j\rangle})$ that computes the difference between the predicted pose transformation $\mathbf{y}_{\langle i,j\rangle}$ and the pose transformation given by the real poses $\mathbf{x}_i$ and $\mathbf{x}_j$. If this function evaluates to zero the estimated constraint matches exactly the difference between the poses it connects. By combining Eq. 2.35 and Eq. 2.36, we can state the pose graph optimization problem as

$$\mathbf{x}_{0:t}^{*} = \operatorname*{arg\,min}_{\mathbf{x}} \sum_{\langle i,j\rangle \in \mathcal{C}} \mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{y}_{\langle i,j\rangle})^{\top} \Omega_{\langle i,j\rangle} \mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{y}_{\langle i,j\rangle}). \tag{2.37}$$

This expression replaces the inverse covariance matrices in Eq. 2.36 by the corresponding information matrix and the evaluation of the differences by the error function defined above. Due to applying the arg min operator, the additive constant and the multiplicative prefactor present in Eq. 2.36 can also be omitted.

In the last years, there have been many developments that substantially improve the graph-based method of Lu and Milios [98]. These techniques render graph-based optimization techniques efficient. In a number of early approaches Gauss-Seidel relaxation for solving linear systems of equations has been proposed for graph optimization [32]. Further insights and the exploitation of the structure of the SLAM problem led to the approach of Olson et al. [114] that suggested using stochastic gradient descent. Grisetti et al. [55] extended this approach in their TORO framework to 3D surfaces. Several methods for solving nonlinear least-squares optimization given an initial guess [26, 57, 77, 78, 84] have also been proposed. In general, they iteratively linearize and optimize the problem at hand. In addition, Konolige et al. [84] analyze and exploit the sparseness of the SLAM problem. Furthermore, Grisetti et al. [57] propose a hierarchical approach of working on manifolds called HOG-Man [57]. Kümmerle et al. [88] propose a general graph optimization framework called $\mathbf{g^2o}$ for minimizing nonlinear error functions. This system is easily adaptable and extendable and available as open source software [88]. In this thesis, we use this framework for solving nonlinear graph optimization problems. Recently released was another general solver for nonlinear graph optimization problems called Ceres [3]. These two frameworks can be seen as the state of the art for solving large-scale SLAM problems.

In Chapter 3, we estimate the trajectory of an RGB-D camera using the framework introduced in this section. We build up a pose graph introducing constraints from matching visual features and use the **g²o** framework [88] to solve the corresponding nonlinear least-squares problem.

2.4. The Traveling Salesman Problem

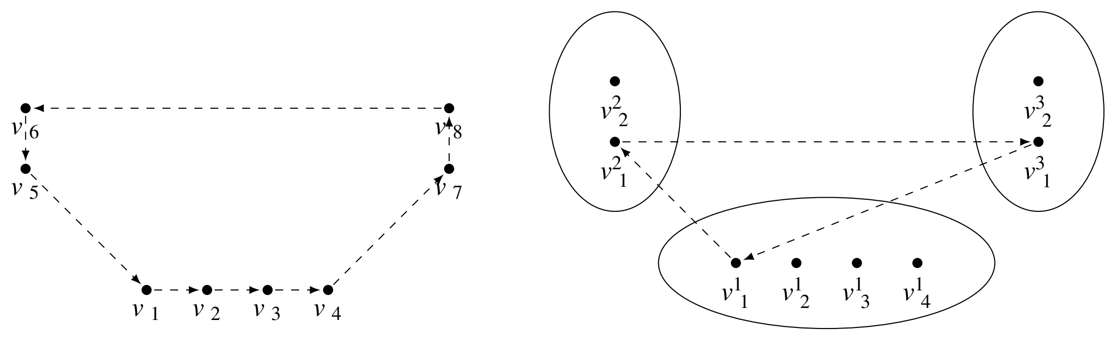
The traveling salesman problem (TSP) is one of the most famous problems in computer science and mathematics. In the classical definition of the traveling salesman problem, a salesman searches for the shortest tour that visits a given number of cities exactly once. In this thesis, we use the graph theory definition of the TSP problem. Given a Graph $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges and $c(i, j)$ is the cost function associated to each edge $(i, j) \in \mathcal{E}$, find the minimum cost tour in the graph that visits each node exactly once. This tour is also called a Hamiltonian cycle. The TSP problem, with general cost functions on the edges, is one of the classical 21 problems shown to be NP-complete by Karp [80]. Figure 2.4a shows an example instance of such a problem.

In the following, we use the notion of a metric. A metric on a set $\mathcal{X}$ is a mapping $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which satisfies the following four properties:

$$d(i,j) > 0 \; \forall i,j \in \mathcal{X} : i \neq j \tag{2.38}$$
$$d(i,j) = 0 \iff i = j \tag{2.39}$$
$$d(i,j) = d(j,i) \qquad \text{(symmetry)} \tag{2.40}$$
$$d(i,j) \leq d(i,k) + d(k,j) \qquad \text{(triangle inequality)} \tag{2.41}$$

A TSP is called symmetric if and only if, for the cost function $c(i, j)$ assigned to the edges in the graph, the first three properties (Eq. 2.38, Eq. 2.39, Eq. 2.40) including the symmetry condition hold. It is said to be metric, if, in addition, the triangle inequality (Eq. 2.41) holds for the cost function of all edges. In the Euclidean TSP the cost function assigned to the graph edges is the Euclidean distance, and the vertices are points in a d-dimensional space.

Many instances of the TSP problem have been shown to be NP-complete. For the Euclidean TSP Papadimitriou [116] showed that this problem is NP-complete. Previously, Garey et al. [50] showed that the TSP problem is NP-complete even for points in the plane



(a) TSP **(b)** GTSP

Figure 2.4.: Visualization of traveling salesman problems applied in this thesis. The nodes in this figure denote points on the 2D plane. The costs between them are given as the Euclidean distance. (a) An instance of the traveling salesman problem. For a given set of nodes (black dots) in a fully connected graph, we seek the shortest tour through all nodes. (b) An instance of the generalized TSP. We are given a set of nodes as well as three clusters defined on the set of nodes. In this case the clusters do not intersect. The objective is to find the minimum cost tour visiting each cluster and exactly one node per cluster. The optimal solution to both problems is denoted by the dashed paths.

when a rectilinear (Manhattan) distance measure is used. However, there are a few special cases for which heuristics with provable guarantees were developed. For the metric version of the TSP, for example, the Christofides algorithm achieves costs of at most 1.5 times the cost of the optimal solution [19]. Furthermore, several improvement heuristics have been developed. They start with some initial tour and then try to improve upon it. One of the most effective heuristics is Lin-Kernighan [95], which works by switching entire sub-tours to find a shorter tour. State-of-the-art TSP solvers such as the Lin-Kernighan heuristic solver (LKH) developed by Helsgaun [61] can use construction heuristics such as the Christofides algorithm to devise an initial tour and then improve it using the Lin-Kernighan heuristic. It is therefore beneficial to model the problem at hand as a metric TSP. This ensures that the Christofides heuristic can be used, and its guarantee on the solution is reached. In addition, it also improves the runtime of the LKH solver [61].

In Chapter 6 and Chapter 7, we frame the problem of planning the cleaning path of a vacuum robot that visits specific parts in the environment as a metric TSP. Thereby we use the LKH solver to query for a solution.

The Generalized Traveling Salesman Problem The generalized traveling salesman problem (GTSP) was first introduced by Henry-Labordere [63]. In the GTSP a set of clusters is defined over the set of nodes. The goal is to find a *g-tour*, a minimum cost tour

Algorithm 3: Greedy algorithm for set cover

```
Input: Collection of subsets $\mathcal{S} = \{\mathcal{S}_1, \cdots, \mathcal{S}_m\}$,
Universe $\mathcal{U}$ with $n$ elements and $\bigcup_{\mathcal{S}_i \in \mathcal{S}} \mathcal{S}_i = \mathcal{U}$
Output: Collection of selected sets $\mathcal{C}$
1 $\mathcal{R} = \mathcal{U}$
2 $\mathcal{C} = \emptyset$
3 while $\mathcal{R} \neq \emptyset$ do
4     $i = \arg\max_j |\mathcal{R} \cap \mathcal{S}_j|$
5     $\mathcal{R} = \mathcal{R} \setminus \mathcal{S}_i$
6     $\mathcal{C} = \mathcal{C} \cup \{\mathcal{S}_i\}$
7 return $\mathcal{C}$
```

which visits at least one node from each cluster [8]. In Chapter 4, we frame the problem of covering an arbitrary 3D surface with minimal cost using a manipulation robot as a GTSP. More specifically, we use the notion of the E-GTSP where the goal is to find a minimal *g-tour* such that each cluster is visited exactly once. Figure 2.4b shows one example of a GTSP where the nodes are visualized by black dots and the three cluster defined over the nodes as ellipses. The dashed path denotes the optimal solution.

Although there exist a large variety of problems that could be modeled with a GTSP, it is nevertheless not often used in practice. This is due to the complexity of solving it, respectively solving for a solution that is even near optimal. On the other hand, there has been a lot of interest in the TSP problem and heuristics dedicated to specific problems [58]. This research has resulted in TSP solvers that can handle instances with a significant number of nodes and edges. The availability of such solvers led researchers to consider transforming the GTSP into a TSP problem and then using a dedicated solver. For this, different techniques and transformations have been proposed (e.g., [8, 29, 112]).

In this thesis, we solve the GTSP by making use of a method introduced by Behzad and Modarres [8]. They show how a GTSP can be converted to a single equivalent directed TSP problem such that the number of nodes in the graph and thus the complexity does not increase. For querying for a solution, a TSP solver can then be applied.

In Chapter 4, we model the problem of planning the end effector trajectory of a manipulation robot for covering an arbitrary 3D object as a GTSP and use the method of Behzad and Modarres [8] for the transformation into a TSP problem.

2.5. The Minimum Set Cover Problem

The set cover problem is one of the 21 original problems that Karp [80] showed to be NP-complete in 1972 [80]. It was in fact shown to be NP-hard [85].

We can state the set cover problem as follows. Given a set $\mathcal{U}$ called universe of n elements and a collection of subsets $\mathcal{S} = \{\mathcal{S}_1, \cdots, \mathcal{S}_m\}$ of $\mathcal{U}$, find the smallest collection of nonempty subsets $\mathcal{C}$ whose union covers all elements of $\mathcal{U}$.

For solving the set cover problem Johnson [75] showed that a greedy heuristic achieves an approximation guarantee of $H(n)$ times the optimal cost, where $H(n) = \sum_{j=1}^{n} 1/j$ is the n-th harmonic number whose value lies in the range between $\ln(n)$ and $\ln(n)+1$. Chvatal [20] showed that this upper bound extends to the weighted version of set cover. The greedy algorithm is detailed in Algorithm 3. Intuitively, it always selects the set that contributes the most towards the solution, i.e., at each step it chooses the set that holds the most uncovered elements (see line 4). Slavík [135] analyzed the lower order terms of the approximation ratio of the greedy algorithm and showed that is has an approximation guarantee of $(1-o(1))\ln n$[41].

Further research was conducted to find lower bounds for approximating the set cover problem and to analyze how far away the performance of the greedy algorithm is from the best achievable approximation ratio. Lund and Yannakakis [100] showed that for all polynomial time algorithms there exists a set cover problem that cannot be approximated better than to the factor of $\frac{\ln n}{2\ln 2}$ of the optimal solution, unless NP has quasi-polynomial time algorithms. This result was improved by Feige [41] to $(1-o(1))\ln n$. Up to low-order terms this matches the approximation guarantee obtained by the greedy algorithm [41]. Therefore, the greedy algorithm results in an approximation guarantee that is essentially the best one can hope for.

In Chapter 4, we frame the problem of constructing an object model suitable for coverage planning as a set cover problem and apply the greedy algorithm to minimize the number of surface patches.

Chapter 3

Three-Dimensional Mapping with an RGB-D Camera

In this chapter, we consider the problem of 3D mapping and object reconstruction with a single RGB-D camera. In many robotic applications such as collision avoidance, localization, and path planning, 3D maps are needed. In particular, they can be used for the reconstruction of the 3D surfaces that are a prerequisite for our approach to coverage path planning (see Chapter 4). We therefore present an approach to simultaneous localization and mapping (SLAM). Based on visual features and the corresponding depth information, we estimate the rigid transformation between camera poses. In addition, we integrate odometry information obtained from a wheeled robot into our system. The estimated camera transformations form the constraints in a pose graph representation. For optimizing this graph, we apply a nonlinear least squares solver. The result of our approach is a globally consistent estimate of the trajectory of the RGB-D camera. Using this estimate, we can obtain different 3D representations of the environment. We evaluate our approach on a publicly available benchmark dataset that includes different scenes with varying lighting conditions and speeds of the moving camera. These experiments show that our approach runs online and can cope with challenging scenarios and, in many cases, results in a highly accurate reconstruction of the camera trajectory. In addition, we show that using available robot odometry information leads to a high robustness even in scenarios with only a few visual features.

Many applications in robotics such as path planning, localization, object reconstruction, and manipulation require dense and accurate 3D models of the environment. A manipulation robot, for example, needs to be able to perceive its 3D environment and create a model

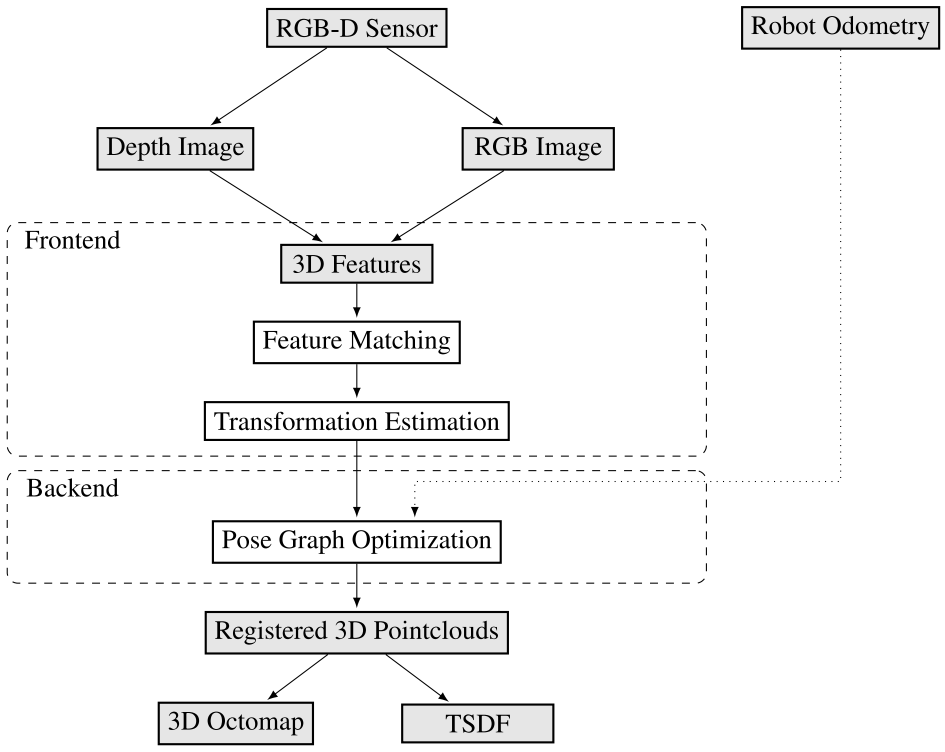


Figure 3.1.: General structure of our SLAM approach.

thereof to plan collision free paths for its manipulator. Household robots such as robotic vacuum cleaners also need to create and update models of their dynamically changing environments. Traditionally, laser scanners have been used for mapping 3D environments. However, for low cost embedded robot systems such as floor cleaning robots, they are often too heavy and expensive. An alternative would be to utilize camera systems, which are typically small and inexpensive. However, on the one hand it is usually hard to obtain 3D information from monocular cameras especially on surfaces with little or no texture. On the other hand, stereo camera systems often only provide sparse depth information. Recently, RGB-D cameras, such as the Microsoft Kinect, have become largely available. At a very low cost they provide color information as well as registered dense depth information at a high frame rate.

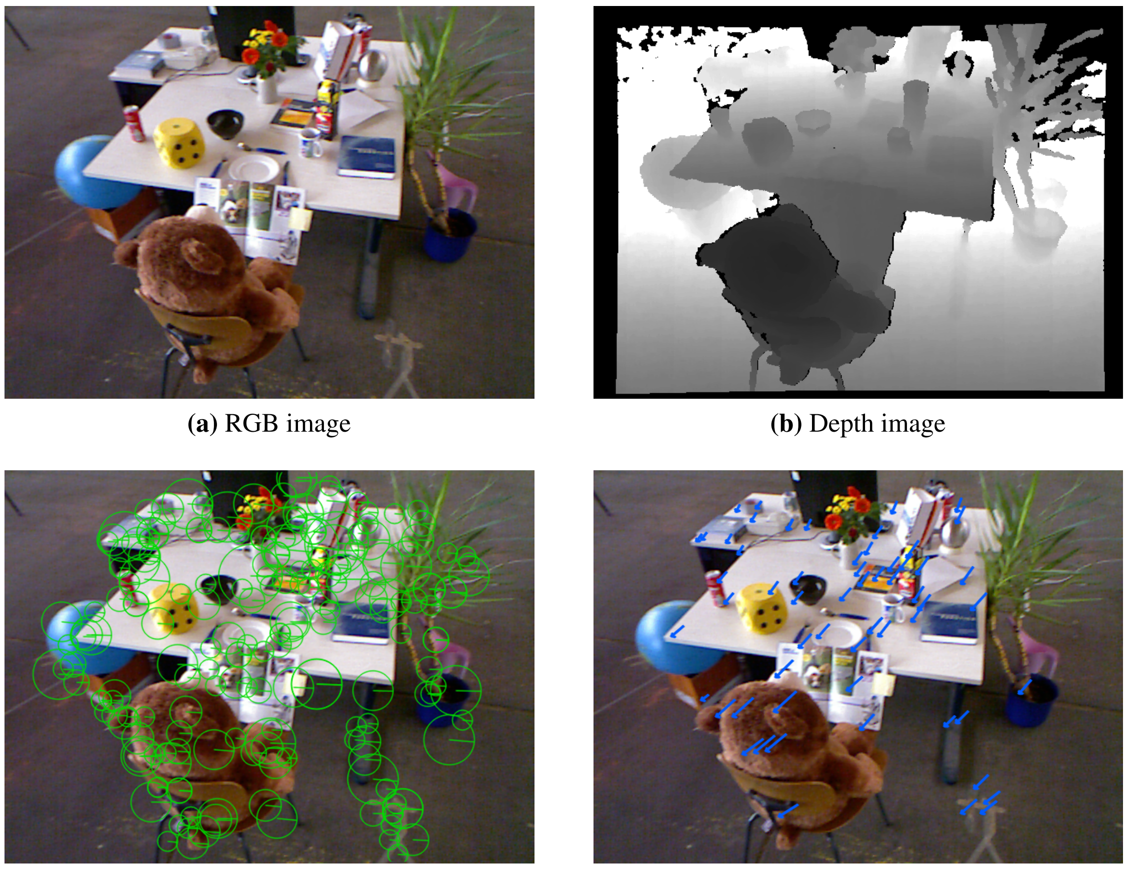
In this chapter, we propose an approach to 3D mapping with a single RGB-D camera. First, we extract visual features from each color image as well as the 3D information corresponding to the features from the depth image. These features are then matched against those from previous frames. Given the resulting 3D point correspondences, we estimate the rigid body transformation between the camera poses of the corresponding frames. If the camera is attached to a mobile robot, we use the odometry information in addition to estimate the camera motion. As a final step, we construct a pose graph consisting of camera poses and all obtained relative transformations. We optimize this graph using the nonlinear least-squares framework $\mathbf{g^2o}$ to obtain a globally consistent estimate. The resulting trajectory allows for computing different 3D representations of the environment.

The result of our approach is a SLAM system for RGB-D cameras, which can be used to acquire precise environment models. Furthermore, it is quickly adaptable, e.g., to different types of visual features, and available as open source software [33]. This allows the user to adjust the system to their needs in terms of calculation time or accuracy. We performed extensive experiments on a publicly available benchmark dataset to evaluate our approach [142]. Thereby, we compared common general purpose feature detectors and descriptors such as SIFT, SURF, and ORB in terms of accuracy and runtime. The experiments show that our approach estimates the camera trajectory with high accuracy in a variety of different and challenging settings. In addition to using camera information only, we also evaluated our approach on datasets in which the camera is attached to a mobile robot and odometry information is available. Our experiments show that including the odometry of the robot can often increase the accuracy of our system, especially in challenging scenarios where only few features can be extracted per frame.

3.1. An Approach to Three-Dimensional Mapping

In this section, we describe the general approach of our SLAM system, which is also visualized in Figure 3.1. The objective of our approach is to estimate the trajectory of camera poses $\mathbf{x}_{0:t}$ over time where a camera pose $\mathbf{x} \in SE(3) \subset \mathbb{R}^{4\times4}$ contains the three dimensional position of the camera and its orientation. As input, our system uses an RGB-D camera, which provides RGB images $M^C \in \mathbb{R}^{m,n}$ as well as depth images $M^D \in \mathbb{R}^{m,n}$. The color and depth image are registered to each other, i.e., the depth measurement of pixel $M^D_{i,j}$ corresponds to the pixel $M^C_{i,j}$ in the color image. Figure 3.2a and Figure 3.2b show an example for a color image and the corresponding depth image. For performing robot SLAM, odometry information can be used as additional input, which we describe in more detail in Section 3.1.1.5.

In general, the mapping system consists of three steps given the input data (see Figure 3.1). In the SLAM frontend, we estimate the camera motion between image frames given the matched visual features between two RGB images and the corresponding depth information. If available, we additionally estimate the camera motion using the robot odometry. The initially estimated poses and constraints between them form a pose graph (see Section 2.3). In the backend, we compute a globally consistent estimate of the camera trajectory by optimizing the nonlinear least-squares problem induced by the pose graph.



(a) RGB image **(b)** Depth image

(c) Extracted keypoints **(d)** Sparse visual flow

Figure 3.2.: Visualization of the processing steps using the *FR2 Desk* sequence of the RGB-D benchmark as an example. (a) + (b) The color image and the depth image provided by the RGB-D camera. (c) Visual features that are extracted from the color image and matched against previous frames. (d) The resulting sparse visual flow of the features in the image.

Figure 3.3 shows an example of an estimated trajectory as well as the corresponding ground truth trajectory for the scene shown in Figure 3.2. As a final step, given the estimated camera trajectory and the camera information, we can compute different types of environment representations. These representations include a point cloud representation, a octree based voxel grid, and a smooth mesh representation called truncated signed distance function (TSDF). They are visualized in Figure 3.4. In the following sections, we provide a detailed explanation of the three main components of our approach.

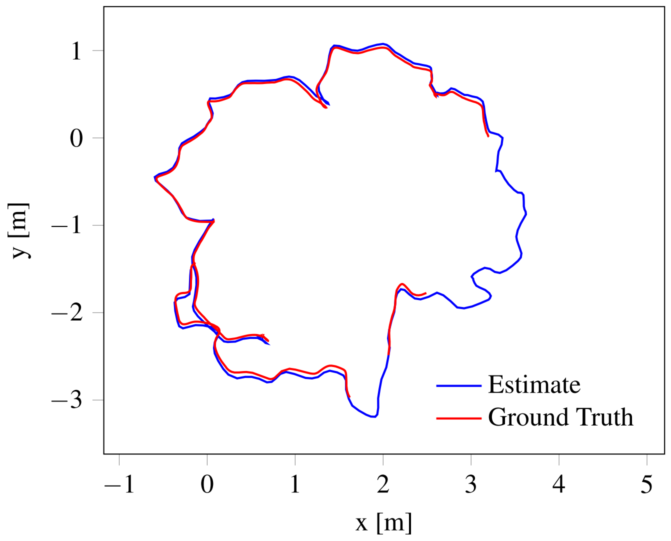


Figure 3.3.: Top view of the ground truth trajectory (red) as well as the trajectory estimated by our approach (blue) for the *FR2 Desk* sequence. The ground truth is not available over the entire trajectory for this dataset.

3.1.1. Frontend for Motion Estimation from Sensor Data

As described in Section 2.3, the objective of the SLAM frontend is to create a pose graph, a graph of camera poses that are connected by constraints. In our approach, each constraint $\mathbf{y}_{\langle i,j \rangle}$ is an estimate of the rigid body transformation between two camera poses $\mathbf{x}_i$ and $\mathbf{x}_j$. The information matrix $\Omega_{\langle i,j \rangle}$ denotes the estimated uncertainty of the constraint. Given the two color images M^{C1} and M^{C2} and their corresponding depth images M^{D1} and M^{D2} as input, we perform the following steps for estimating the camera motion between these two image frames:

1. Compute image features from both color images and extract the 3D position in the camera coordinate frame from the depth image.

2. Match the image features from both color images.

3. Perform random sample consensus (RANSAC) [42] to estimate the relative camera motion given the 3D position of the matched features and identify outliers.

4. Insert the found relative motion as constraint into the pose graph.

Figure 3.2c and Figure 3.2d show an example of the extracted keypoints and the resulting sparse visual flow. For building the pose graph, we first estimate the visual odometry of the camera by matching consecutive frames. If robot odometry information is available, we insert an additional constraint between camera poses. In a second step, we find loop closures by matching image frames over larger parts of the trajectory. The following sections describe the steps for estimating the camera motion and our strategy for finding matching candidates for loop closures in detail.

3.1.1.1. Matching Visual Features

Given two RGB images M^{D1} and M^{D2}, as the first step, we use a feature detector to detect n visual features in each image. For each detected feature we extract an m-dimensional descriptor $\mathbf{f} \in \mathbb{R}^m$. Our approach generalizes over the type of image features. The choice of the type of visual feature largely depends on the application and computational expense that is acceptable. It can, however, significantly influence the overall performance of the SLAM system. In the experiments, we evaluate the commonly used and robust features SURF [7] and SIFT [97]. In addition, we also compare these to the ORB [125] descriptor, which aims at being computationally more efficient than SIFT. We evaluate the accuracy and computational efficiency of these feature types in several different settings in which we expect our approach to be applied.

After extracting a number of feature descriptors from the two images to be compared, we have to determine the data association between the image features. For matching two descriptors, we first compute their distance in feature space. As a distance measure, we use the Euclidean distance for the SURF and SIFT descriptors. In the case of the ORB descriptor, we use the Hamming distance, as proposed by Rublee et al. [125]. As observed by Lowe [97], using only the distance for determining associations between descriptors is not a good criterion. Instead, they suggest using a threshold on the ratio of the distance of the descriptor to its nearest neighbor and its second nearest neighbor. They assume that, if we compare the descriptors obtained from the two RGB images M^{C1} and M^{C2}, a descriptor extracted from M^{C1} should match to exactly one descriptor in M^{C2}. The distance to the second nearest neighbor in descriptor space should therefore be much larger than to the first nearest neighbor. We match two descriptors based on a threshold on this distance. The value of this threshold largely influences the ratio of false positive matches. During the estimation of the motion between camera poses, false positive matches are therefore likely and need to be adequately handled.

3.1.1.2. Computing the Rigid Body Transformation

In the previous step, we have obtained a number of corresponding image features from two camera images. Therefore, we are now given two sets of matched feature descriptors $\mathcal{F}_a = \{\mathbf{f}_a^i\}_1^m$ and $\mathcal{F}_b = \{\mathbf{f}_b^i\}_1^m$ from image M^{C1} and M^{C2}. These sets are paired such that feature $\mathbf{f}_a^i$ matches to feature $\mathbf{f}_b^i$. For computing the rigid body transformation between the camera poses from which the images have been obtained, we determine their position with respect to their local camera viewpoints. This position is computed by reading off the depth value and viewing angle from the respective pixel in the depth image that corresponds to the location in the color image. We save all 3D locations into two paired sets of feature positions $\mathcal{A} = \{\mathbf{a}_1, \cdots, \mathbf{a}_n\}$ and $\mathcal{B} = \{\mathbf{b}_1, \cdots, \mathbf{b}_n\}$ where $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$. Given these two point sets, we are looking for the similarity transformation between them, which corresponds to solving the following minimum least-squares problem:

$$(R, \mathbf{t}, c)^* = \underset{(R,\mathbf{t},c)}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{b}_i - (cR\mathbf{a}_i + \mathbf{t})\|^2, \tag{3.1}$$

where R is the rotation matrix, t is the translation vector, and c is a scaling factor, which is assumed to be one in our case.

To solve this least-squares optimization problem in closed-form, we use the solution provided by Umeyama [150] that extends the work of Arun et al. [4] to give a more robust solution. Let $\Sigma \in \mathbb{R}^{3\times3}$ be the covariance matrix of both point sets, i.e., $\Sigma = \frac{1}{n}\sum_1^n(\mathbf{b}_i - \mu_{\mathcal{B}})(\mathbf{a}_i - \mu_{\mathcal{A}})^\top$, where $\mu_{\mathcal{A}}$ and $\mu_{\mathcal{B}}$ denote the mean of the point sets $\mathcal{A}$ and $\mathcal{B}$. Additionally, let $UDV^\top$ be the singular value decomposition of Σ. Umeyama [150] shows that the optimal transformation is given by

$$R = USV^\top \tag{3.2}$$

$$t = \mu_{\mathcal{B}} - cR\mu_{\mathcal{A}} \tag{3.3}$$

$$c = \frac{1}{\sigma_{\mathcal{A}}^2} \operatorname{tr}(DS), \tag{3.4}$$

where $\sigma_{\mathcal{A}}^2$ and $\sigma_{\mathcal{B}}^2$ are the variances of the point sets $\mathcal{A}$ and $\mathcal{B}$, and S is chosen as

$$S = \begin{cases} \operatorname{diag}(1, 1, \cdots, 1, -1) & \text{if } \det(U)\det(V) = -1. \end{cases} \tag{3.5}$$

In this estimation process, the data association of the point sets $\mathcal{A}$ and $\mathcal{B}$ is assumed to be given and correct. Thus, the result heavily relies on the correct data association.

3.1.1.3. RANSAC for Outlier Detection

To robustly determine the rigid transformation despite false positive descriptor matches, we apply Random Sample Consensus (RANSAC), which was introduced by Fischler and Bolles [42]. RANSAC is a method for robustly estimating model parameters given noisy observations. It assumes that there exist a number of inliers, noisy observations that support the model, and outliers that do not support the model. To fit the model to the data, the algorithm proceeds iteratively. In our case, we determine a valid motion transformation by iteratively performing the following steps:

1. Randomly select four matches. Using their local 3D position, we compute the rigid transformation as described in the previous section.

2. Apply the resulting transformation to all matches and compute the Mahalanobis distance of the 3D position of the matches to the estimated model.

3. Based on an ε-threshold on this distance, we extract the inliers and recompute the transformation estimate.

We terminate the algorithm after a fixed number of iterations and adopt the model with the most inliers. In many environments this approach results in a reliable motion estimate. In environments with highly repetitive structures such as patterned wallpaper or the repeated occurrence of objects like chairs, for example, false matchings could occur. To partially solve this problem, we require a minimum number of inliers for accepting the found transformation as valid. Intuitively, we expect the number of consistent matches from repetitive structures to be substantially smaller than those generated by the camera motion. Depending on the environment, the minimum number of inliers should be as high as possible to ensure a minimum number of wrong associations.

The motion estimate resulting from the RANSAC algorithm forms the mean of a new constraint of a pose graph (see Section 2.3). The information matrix which is the inverse of the covariance matrix, specifies the uncertainty of the constraint. Assuming isotropic noise, we estimate the diagonal entries of the covariance matrix as the mean squared error of the inlier matches to the estimated mean motion. As we show in the experimental section, this approach to motion estimation results in reliable estimates in a large number of settings and environments.

3.1.1.4. Building the Pose Graph

In the previous sections, we showed how to compute an estimate of the camera motion from two RGB-D camera frames. To obtain a globally consistent estimate, we build up a pose graph as described in Section 2.3, which constitutes a nonlinear least-squares problem. For each camera frame received, we insert a node into the graph. We then estimate visual odometry from matching the current frame at time t to the previous one at time $t-1$. If no valid transformation can be obtained, we assume the identity transformation with high covariance as motion estimate to keep the graph connected. Matching only subsequent frames could result in substantial drift, accumulating the error of the individual motion estimates. This error is due to several influences including false positive feature matchings, noise in the distance measurements, errors in the registration of the color and depth image, and others. For reducing drift, we therefore aim at introducing additional constraints into the pose graph that connect nodes other than the direct successor and predecessor. These constraints are also called loop closures. The matching of a frame to a much earlier frame can constrain the drift and thus the error of the estimated camera trajectory. A general approach to determine all possible constraints would be to compare the current frame to all previous ones. The comparison to all previous frames, however, is not computationally feasible.

For two frames to correspond, a substantial overlap between image views is required. The greater the viewpoint difference is, the fewer the number of keypoints becomes that can be robustly matched, even if the same structure is visible in both frames. As a result, only a few previous image frames will match the current one. We propose an efficient heuristic for selecting matching candidate image frames for finding loop closures. The heuristic proceeds in three steps:

1. Select n previous frames for comparison as they are likely to match the to current one. We apply this step to find small loop closures for locally correcting the drift of single image frames.

2. Search the geodesic graph neighborhood of the current node to find large loop closures. To implement this selection schema, we compute a minimum spanning tree of limited depth with the predecessor of the current frame as the root. We then draw k frames from the spanning tree. Thereby we bias the selection to earlier frames to create loop closures over a longer distance. As a result, the search is guided towards previously matching frames. This is especially useful for exploiting already found large loop closures.

3. Randomly sample l frames from all previous ones. This uninformed selection step aims at finding constraints to much earlier frames to achieve a greatly reduced drift.

In the experiments, we show that this heuristic for frame selection works effectively in the moderately sized indoor environments that we consider for our approach.

3.1.1.5. Integration of Odometry Information

In general, our approach only requires the output data of an RGB-D camera to reconstruct the camera trajectory. In addition, a minimum number of distinct visual features with corresponding depth information must exist in the environment such that valid correspondences between image frames can be found. In some environments, however, this might not be the case. Long and uniformly colored hallways, for example, may lack many visual features. Furthermore, as the maximum range of the depth camera is limited, the depth information for an extracted feature may not be available. Another example are large halls such as warehouses. They usually have a very ambiguous structure of isles of racks or shelfs that could lead to false positive feature matchings. These issues could result in the introduction of invalid or false positive camera motion estimates.

In the case of robot SLAM, when the camera is attached to a mobile robot, the odometry information provided can help to overcome these challenges. We thereby assume that the pose of the RGB-D camera on the robot has been calibrated beforehand. To introduce the odometry information into the pose graph, we proceed as follows. If a new image frame has been received and thus a new node been inserted into the pose graph, we additionally estimate the relative displacement of the camera poses given the robot odometry. We then insert the odometry information as an additional edge between camera poses. As described in Section 3.1.1.4, to keep the graph connected, we inserted an identity transform between frames for which no valid motion estimate could be obtained by matching visual features. When using odometry information, the graph will always be connected, which renders the insertion of an identity transform unnecessary. To specify the covariance of the odometry constraints, we calibrate the robot odometry beforehand by applying the odometry motion model [145, pp. 132]. Assuming isotropic noise, we then estimate the diagonal of the covariance matrix.

3.1.2. Backend for Pose Graph Optimization

In the previous section, we showed how camera motion estimates can be computed from pairwise matches of image features. In addition, we showed how motion estimates from robot odometry can be integrated into the SLAM system. Using these constraints, we build up a pose graph resulting in an initial estimate of the camera trajectory. Due to noise in the measurements or systematic errors in general, this initial guess of a trajectory is not globally consistent. Loop closures, constraints that are obtained from revisiting the same parts of the environment, can substantially reduce local errors and drift. In Section 3.1.1.4, we proposed a heuristic for finding these loop closures and introduced the respective constraints into the graph. As we showed in Section 2.3, optimizing the resulting pose graph corresponds to solving a nonlinear least-squares problem. To minimize this nonlinear squared error function, we use the **g^{2}o** framework by Kümmerle et al. [88], a general framework for minimizing nonlinear least-squares error functions. It implements several minimization algorithms, such as Gauss-Newton and Levenberg-Marquardt. Another common practice that we apply in our approach is to use the Levenberg-Marquardt algorithm [104] for nonlinear least squares minimization. It is essentially a combination of the iterative minimization approaches gradient descent and the Gauss-Newton algorithm. While Gauss-

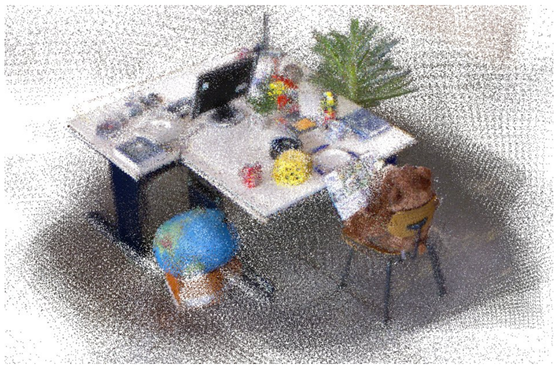
Newton is known to converge quickly in close vicinity to a local minimum, it can also fail in particular when the current estimate is far from the minimum and so the linearization leads to a poor approximation of the nonlinear cost function. In addition, one step of Gauss-Newton must not always reduce the value of the function minimized. Through the combination with steepest gradient descent, Levenberg-Marquardt enforces a smaller function value in each iteration. The optimization framework **g^{2}o** also offers the usage of different kernels as error measures. The experiments reported in this thesis, were obtained using dynamic covariance scaling (DCS), a method for optimizing pose graphs in the presence of outliers [1]. Compared to the standard squared distance error function, dynamic covariance scaling reduces the weight of constraints with large errors by gradually scaling the information matrix. The analysis provided by Agarwal et al. [2] suggests that in case of poor initial configurations, dynamic covariance scaling has a positive effect on the optimizer.

3.1.3. Generating an Environment Representation

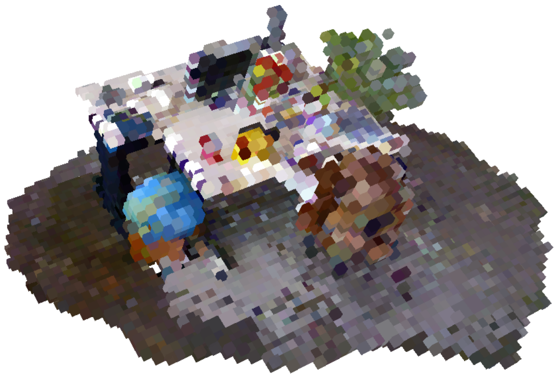
Having obtained a globally consistent estimate of the camera trajectory, the final step is to construct an environment representation. Depending on the application that the resulting model is used for, different types of environment representations are possible. In this section, we give a short overview of these techniques.

One common and very general representation is the point cloud. A point cloud is essentially a set of 3D points. In our case, each point additionally holds color information. Given the estimated trajectory of the camera, we extract the global pose of each camera frame from the optimized pose graph. Together with the color and depth image taken from this pose, we compute a colored point cloud. Figure 3.4a shows an example of such a point cloud resulting from our approach. A point cloud representation, however, has several drawbacks, especially for applications like collision free navigation or localization of a robot in an environment. If checking for collisions between the robot and the environment, for example, the robot model has to be checked against every single point in its vicinity. This can be computationally expensive, especially in areas that are densely represented. In addition, point clouds are not memory efficient and are prone to sensor noise, as every single point is rendered and stored.

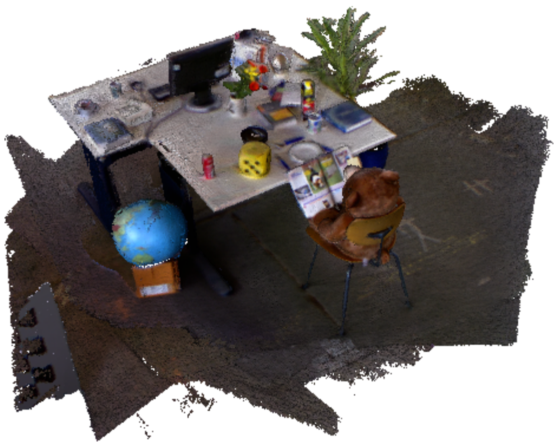
For localizing a robot in the environment or collision checking, often a probabilistic map representation is used. A widely adopted 3D map representation is a probabilistic 3D grid based on an octree data structure. It has several advantages for robot navigation. First, an occupancy probability can be estimated for each cell of the probabilistic 3D grid. Second, free as well as unknown areas are explicitly represented. Finally, using the octree as the basic internal structure, the environment can be represented in multiple resolutions.



(a) Point cloud



(b) OctoMap



(c) Truncated signed distance function (TSDF)

Figure 3.4.: Environment representations that can be computed given the estimated camera trajectory and the data of the RGB-D camera. (a) A point cloud representation directly computed from the depth and color image.(b) Visualization of an OctoMap, a probabilistic 3D grid representation. (c) A smooth mesh representation, which is often used for applications like 3D printing.

Thus, a small number of larger voxels can represent large areas of free or occupied space, which can significantly decrease the memory required for storing the map. An octree is constructed from point clouds that contain the view point of the sensor from which they were obtained. When inserting a point into the octree, a ray casting is performed that starts at the sensor view point. Based on the number of points that end inside a voxel and the number of rays passing, a probabilistic model allows for estimating and updating the occupancy probability of each voxel. Hornung et al. [69] implemented such a data structure in their OctoMap framework and showed that it can be efficiently created, accessed, stored in memory, and used to model large environments. In addition, it can be used to obtain different 2D grid map representations, e.g., by extracting a “slice” at a given height or by “downprojecting” the entire map onto a certain height level. Figure 3.4b shows an example of an OctoMap reconstruction.

Apart from robot localization, where a probabilistic environment representation is often used, other applications such as reconstructing a model for 3D printing, for example, require a smooth surface without holes. In these cases, a grid or point cloud representation is not suitable. Instead, to obtain a smooth surface, we use a meshing technique called signed distance function [23]. Most implementations of signed distance functions require a point cloud that also contains the surface normal for each point as input. They then compute a 3D grid representation of the environment. Each grid cell stores the signed distance to the closest obstacle where the sign is determined by the surface normal. The surface is then estimated as the location of the sign change and extracted using a meshing algorithm such as marching cubes [96]. For efficiency, the maximum distance to the surface is truncated. This technique is therefore called Truncated Signed Distance Function (TSDF). Figure 3.4c shows an example of a TSDF reconstruction that is computed from the point cloud shown in Figure 3.4a. We used the CPU-TSDF [106] library, an implementation by Miller [106], to reconstruct the surface in this figure. It is based on techniques of Curless and Levoy [23] to compute an approximated signed distance function.

Our system allows generating a point cloud representation or an OctoMap directly. For computing the TSDF, the point clouds are exported with estimated surface normals, which can then directly be used as input in CPU-TSDF.

3.2. Evaluation

The presented approach aims at reconstructing the camera trajectory as accurately and robustly as possible. The general system is independent of the type of visual feature used. Furthermore, a number of parameters can be specified that influence the overall performance, e.g., the visual feature used or the parameters for the loop closure search. In addition, there are several external properties and parameters such as the environment in which the system is deployed, the speed of the moving camera, or the lighting conditions. With the evaluation, we aim at assessing the overall performance of the proposed system and answering, in detail, the following questions. What is the best feature to use in terms of accuracy and computation time? How robust is the system under changing environments and at different speeds of the moving camera? Does the integration of the robot odometry substantially increase the accuracy of the system and, if so, in which situations? To answer these questions, we performed extensive experiments on benchmark datasets, which cover a variety of different situations.

In the next section, we describe the RGB-D benchmark and error measure used to evaluate our approach. Thereafter, we present the experimental setup. We then evaluate our approach on handheld datasets, using correspondences from visual features only. Finally, we evaluate the robot SLAM scenario, integrating the odometry measurements into the pose graph in addition to correspondences computed from visual features.

3.2.1. Benchmark Dataset

For evaluating our approach, we rely on a publicly available RGB-D benchmark dataset [142]. The dataset was recorded using the Microsoft Kinect as well as an Asus Xtion Pro RGB-D camera. It contains many challenging scenarios ranging from feature-rich environments to environments that only possess a few visual features. The sequences in the *Handheld* category against which we test our approach, contain slow as well as fast movements and changing lighting conditions. Fast movements can result in substantial motion blur and can lead to additional errors when registering the depth in the color image. Additional challenges include blocking of the field of view for a few seconds while the camera is moving (*FR2 Kidnap* sequence) or pointing the camera onto a floor with possibly ambiguous features (*FR1 Floor* sequence). We also test our approach against the sequences of the *Robot SLAM* category. In these sequences a Kinect is mounted to a Pioneer 3 robot. In addition to the camera data, robot odometry is available and the location of the camera on the robot has been calibrated. The sequences were recorded in a larger industrial building with only a few salient features in the environment. Due to the large size of the hall and the considerably short maximum range of the RGB-D sensor, depth data is not available for a large number of detected features. To pose a further challenge, the robot not only traverses entirely flat areas, but its motion is also greatly disturbed.

Although we aim at accurately reconstructing a map of the environment, the error of the resulting map is generally hard to assess. However, the overall map reconstruction error, is closely related to the accuracy of the trajectory reconstruction. To evaluate the trajectory reconstruction error, the RGB-D benchmark provides the ground truth camera trajectory. The ground truth of all datasets was recorded by a motion capture system from MotionAnalysis [108]. The system tracks the 3D position of passive reflecting markers with high accuracy. Together with the datasets, the benchmark provides tools for evaluating different error measures in comparison to the ground truth. These are the *relative pose error* (RPE) proposed by Burgard et al. [14] and the *absolute trajectory error* (ATE) proposed by Sturm et al. [142]. Over a fixed time interval, the RPE evaluates the local error of the underlying system. It evaluates the rotational and translational error separately and is often used for drift analysis. The ATE additionally assesses the overall consistency of the estimated trajectory and the ground truth trajectory by measuring the absolute distances between both trajectories. In addition, this allows for easy visual inspection when comparing the estimated trajectory to the ground truth. Specifically, let T be the rigid body transformation that maps the ground truth trajectory $Q_{1:n}$ onto the estimated trajectory $P_{1:n}$ using least squares. As the correspondences are known from timestamps in this case, T can be computed by the method of Horn [68] or Umeyama [150]. The offset at time i is then given as

$$\mathbf{F_i} := Q_i^{-1} T P_i. \tag{3.6}$$

For the entire trajectory, Sturm et al. [142] then compute the error as

$$\mathrm{RMSE}(\mathbf{F_{1:n}}) := \Big(\frac{1}{n}\sum_{i=1}^{n} \|\mathrm{trans}\,\mathbf{F_i}\|^2\Big)^{1/2}. \tag{3.7}$$

As mentioned above, this expression only considers the translational part of the error denoted as $\mathrm{trans}\,\mathbf{F_i}$. However, as noted by Sturm et al. [142] the rotational error also manifests itself in the translation and is highly correlated to the RPE. In their experiments, the relative order of the compared approaches in terms of their error was most often the same compared to the RPE. In our experiments, we use the ATE to evaluate the accuracy of our approach.

3.2.2. Experimental Setup

Before evaluating our approach in the following sections, we detail the experimental setup. The presented system can be applied in an online or offline fashion. However, to assess the maximum accuracy of the system, all presented results are computed offline, i.e., we only triggered the optimizer back-end once, after the entire graph had been constructed. The offline computation also allows for comparing the accuracy of the different visual features without having to compromise, for example, the number of features selected due to timing constraints. For the evaluation of the computation times, we used an Intel Core i7 CPU with 3.40 GHz, and a nVidia GeForce GTX 560 graphics card. In preliminary experiments, we determined the parameters for selecting image frames as matching candidates during the loop closure search (see Section 3.1.1.4). For the experiments presented, we set the number of predecessor frames to compare to $n = 8$, the number of frames selected from the minimum spanning tree of the node to $k = 2$ and the number of frames randomly selected for comparison to $l = 3$. For feature detection and implementation of the descriptors, we rely on the OpenCV library [10] for the SURF and ORB features. For implementing the SIFT detector and descriptor, we applied SIFT-GPU, a parallel GPU implementation of SIFT [155]. To determine the nearest neighbors between visual feature descriptors, we found that using the Fast Approximate Nearest Neighbors (FLANN) [109] substantially decreases the computation time compared to the brute force approach while maintaining high accuracy [35]. In all presented experiments, we therefore used FLANN to determine the nearest neighbors. To achieve a motion transformation estimate, we performed a maximum of 200 RANSAC iterations. To ensure that all valid features can be extracted, we set the maximum number of features to a high value. Typical numbers of extracted features per image frame range from 10 for environments with few salient features to 800 for feature-rich environments. In general, we found that these parameter values work reliably for a number of different environmental settings and scenarios that our approach aims at, e.g., indoor environments with medium sized trajectories. In case of other applications such as online mapping with low computational power, parameters such as the number of extracted features and the number of RANSAC iterations can be reduced.

3.2.3. Handheld SLAM

In this section, we evaluate the accuracy and runtime of our approach in the handheld SLAM scenario. We thereby only rely on the data provided by the camera, the RGB and depth image. Our goal is to quantify the expected error of our system in different scenarios. In addition, we evaluate the accuracy and computational cost of our approach for different types of visual features.

We evaluated our approach on sequences from the handheld category of the RGB-D benchmark. Table 3.1 shows a detailed specification of the datasets used for evaluation including their length, duration, as well as their average translational and rotational velocity. As can be seen, the average velocities vary on a large scale, ranging from 6.34 deg/s to 41.6 deg/s for the rotational velocity and from 0.16 m/s to 0.43 m/s for the translational velocity. We omitted two long sequences of the handheld category. Ground truth information was only available for very small parts of their trajectory.

The most influential parameter on the performance of the proposed system is the feature detector and descriptor used. As already mentioned above, there are several combinations

Sequence	Length [m]	Duration [s]	Avg. Angular Velocity [deg/s]	Avg. Transl. Velocity [m/s]	Frames
FR1 360	5.8	28.7	41.6	0.21	745
FR1 Floor	12.6	49.9	15.1	0.26	1214
FR1 Desk	9.3	23.4	23.3	0.41	575
FR1 Desk2	10.2	24.9	29.3	0.43	614
FR1 Room	16.0	48.9	29.9	0.33	1332
FR2 Hemisphere	14.8	91	20.6	0.16	2727
FR2 Kidnap	14.3	48	13.4	0.30	1429
FR2 Desk	18.9	99	6.3	0.19	2964

Table 3.1.: Sequences of the handheld category of the RGB-D benchmark that were used for the evaluation of our approach.

	Frame Rate [hz]	RMSE ATE [m]
SIFT-GPU	3.95	$\mathbf{0.17 \pm 0.22}$
SURF	3.53	0.23 ± 0.23
ORB	3.98	0.34 ± 0.41

Table 3.2.: Evaluation of the ATE and runtime depending on the visual feature used. The results are averaged over all trajectories listed in Table 3.1. Figure 3.5 visualizes the individual errors for each sequence.

of visual feature detectors and descriptors that are commonly used in the literature. In our experiments, we evaluate SIFT-GPU and SURF as well as the ORB feature descriptor. Due to the random selection of features during the motion transformation estimation as well as the random selection of frames during loop closure search, we repeated each experiment 100 times.

Table 3.2 shows the resulting root mean square ATE as well the achieved frame rate of the experiment averaged over all tested sequences. As can be seen, using the SIFT-GPU detector and descriptor results in the smallest error, while using ORB results in the largest one. Paired Wilcoxon tests [154] showed that the error differences between the features tested are significant at a 5% level. Figure 3.5 shows the resulting error for each feature type per sequence tested. For most sequences SIFT-GPU provided the best performance. Only for two of the sequences, namely *FR2 Desk* and *FR2 360 kidnap*, SURF showed the best result. Using ORB always resulted in the largest error. The sequences used in the

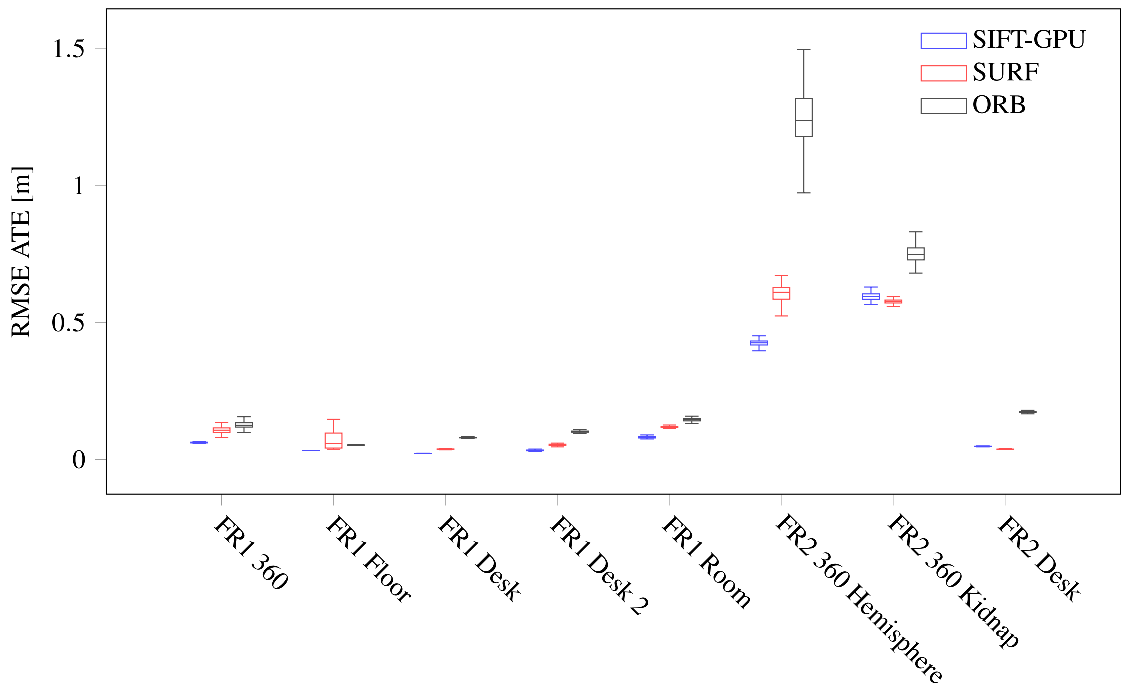


Figure 3.5.: Root mean squared ATE for each sequence and feature type.

handheld category mostly contained feature-rich environments. There are two sequences, however, in which the environment contained comparably few visual features, the *FR2 360 hemisphere* and the *FR2 360 kidnap* sequence. For those sequences, all feature types resulted in a high RMSE ATE compared to the other sequences of more than 0.5 m. Both sequences were recorded in the industrial hall without pointing the camera to a table with objects but to the walls or ceiling with relatively few salient features. Due to the high ceiling in the industrial hall, even for detected features, depth information is either not available or is associated with high noise. In addition, in the *FR2 360 kidnap* sequence, the camera was covered for a few seconds while being moved. In the *FR2 360 hemisphere* sequence, the camera is often pointed toward the ceiling or the ground on which only few features can be detected. As Figure 3.5 shows, using ORB results in a much larger error and variance than the other features in this case.

Considering the frame rate, ORB achieved the best overall result, closely followed by SIFT-GPU (see Table 3.2). Table 3.3 gives a detailed evaluation of the frame rate and runtime using SIFT-GPU. For all sequences, the time needed by the graph optimizer is small as we perform offline optimization, triggering the optimizer only once after the pose graph has been built. The total runtime of the system is therefore dominated by the feature extraction and matching. The computation time spent for matching image frames directly depends on the number of candidate images selected for loop closure detection. For our

Sequence	Total Runtime [s]	**g^2o** Runtime [s]	Frame Rate [hz]	RMSE ATE [m]
FR1 360	139.5	2.3	5.4	0.061 ± 0.002
FR1 Floor	410.4	4.2	3.0	0.032 ± 0.000
FR1 Desk	155.4	4.1	3.8	0.021 ± 0.000
FR1 Desk2	139.4	1.9	4.5	0.033 ± 0.002
FR1 Room	325.1	5.4	4.2	0.081 ± 0.003
FR2 360 Hemisphere	537.1	47.6	5.6	0.435 ± 0.089
FR2 360 Kidnap	314.4	28.2	5.0	0.605 ± 0.110
FR2 Desk	911.4	153.7	3.9	0.047 ± 0.000

Table 3.3.: Detailed evaluation of the runtime of our approach using SIFT-GPU as feature detector and descriptor. The results are averaged over all 100 trials. The timing results have a low variance and are listed for completeness. For this experiment we only optimized once at the end. The calculation of the frame rate therefore does not include the optimizer runtime. The optimizer runtime denotes the time for optimizing the entire pose graph once.

current evaluation setup, the system achieves a rate of 3.0 to 5.6 frames per second. For some sequences of environments with many salient features, e.g., the *FR1 Desk* sequence, the number of previous frames that we match the current image against could be lowered. As a result, the achieved frame rate would increase for these sequences while resulting in a comparable error as shown in Endres et al. [36]. For this evaluation, however, the goal is to demonstrate that the overall system in general is applicable to a variety of situations given a set of standard parameters.

As overall result, we suggest relying on SIFT-GPU, if a parallel GPU-based implementation can be used. In this case, the runtime of SIFT-GPU is comparable to ORB while generally achieving a much lower error. On systems without a GPU, the SURF implementation leads to a stable result for all sequences at a relatively low error. In case of online usage, as well as if only low computing resources are available, using ORB features can be beneficial in environments with many visual features. In the following, we further analyze the robustness of our approach in environments with few salient features as well as for robot SLAM.

3.2.4. Integrating Odometry Information for Robot SLAM

In the previous section, we evaluated our approach in a variety of handheld SLAM scenarios. We now focus on the robot SLAM scenario where the RGB-D camera is attached to a

Sequence	Length [m]	Duration [s]	Avg. Angular Velocity [deg/s]	Avg. Transl. Velocity [m/s]	Frames
Pioneer 360	16.1	737	12.0	0.23	968
Pioneer SLAM	40.4	156	13.4	0.26	2468
Pioneer SLAM 2	21.7	116	12.2	0.19	1798
Pioneer SLAM 3	18.1	112	12.3	0.16	2370

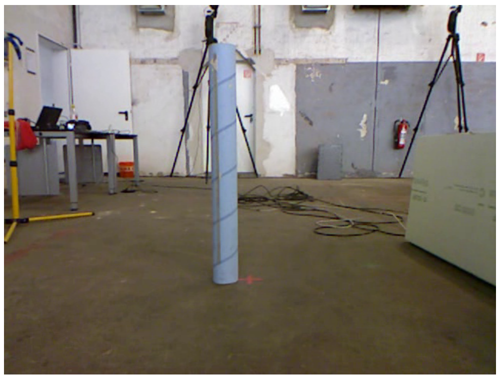
Table 3.4.: Properties of the *Robot SLAM* sequences of the RGB-D benchmark.

mobile base and the odometry information of the robot is available. In general, we assess the performance of our approach in larger environments and environments with few salient features. To evaluate the performance and robustness of our approach, we rely on the *Robot SLAM* sequences of the RGB-D benchmark (see Table 3.4). The benchmark dataset contains five Pioneer robot sequences. In this work, we used the *Pioneer 360 Validation* sequence to obtain an estimate of the uncertainty of the odometry as described in Section 3.1.1.5. We then tested our approach on the remaining four sequences listed in Table 3.4.

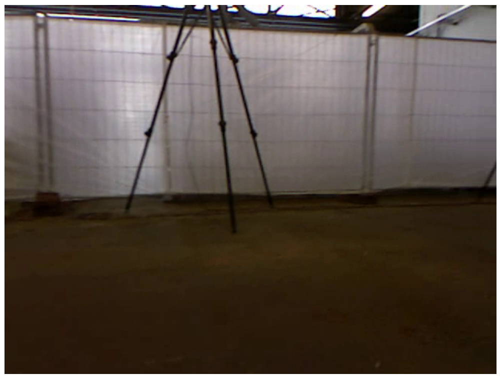
The sequences were recorded in an industrial building with relatively few and sometimes ambiguous features. Figure 3.6 shows some example views from the camera attached to the robot in the industrial hall. Especially in Figure 3.6b, one can see a cylindrical obstacle with a barber pole like pattern. From this pattern, many stable features can be detected. Due to the repetitiveness of the pattern, however, many feature descriptors have a small distance. Matching these descriptors could result in estimating a vertical motion, instead of the real motion performed by the camera, especially if there are only few other features in the environment. Another example is the covered fence shown in Figure 3.6c that also shows a repetitive pattern and represents a similar challenge as the pole for feature matching. As an additional disturbance, the robot moves over cables. Although the robot is largely performing planar movements, the problem at hand is thus not in 2D.

For this evaluation, we rely on SIFT features as they proved to result in the best performance as shown in the previous section. Due to the random elements of our algorithms, we performed 100 runs for each sequence as in the previous section. Table 3.5 shows the result of this experiment using odometry and visual features only as an estimator as well as the result for combining both estimates. As can be seen, the error using visual features only is often higher than that for the handheld sequences tested in the previous section (compare to Figure 3.5). This is due to the environment containing few salient features as described above. Combining odometry and visual features for motion estimation results in a substantially smaller error in three out of four sequences. For the *Pioneer SLAM 2*

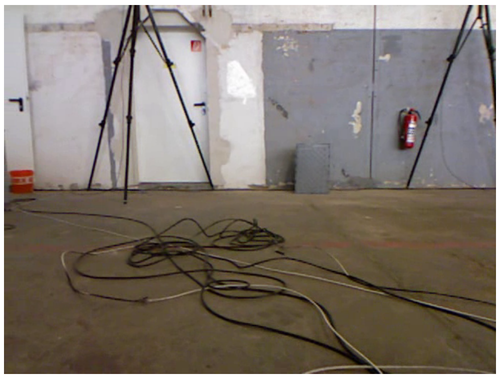
(a)



(b)



(c)



(d)

Figure 3.6.: The Pioneer robot used to record the *Robot SLAM* sequences and some example views of input images of the *Pioneer SLAM 2* sequence. The Pioneer sequences were recorded in a large industrial hall with relatively few features. (a) The robot with an attached Kinect camera as well as reflective markers that are tracked by the motion capture system. (Figure courtesy J. Sturm) (b) + (c) A pole and a wall with repetitive patterns that can lead to false positive feature matchings. (d) Cables on the ground which lead to strong disturbances in the robot odometry, especially its rotation.

sequence, however, the error is only slightly reduced compared to using robot odometry only. Compared to using visual features only, the error even increased. This result is due to the fact that the odometry readings are significantly disturbed in this sequence. This can also be seen in Table 3.5 when comparing the odometry error of the tested sequences. The *Pioneer SLAM 2* sequence results in almost twice the error compared to the other Pioneer sequences, although the distance traveled as well as the rotational and translational speed are comparable (see Table 3.4). The high error is due to slippage of the wheels caused by

Sequence	Visual Features	Odometry	Visual Features + Odometry
Pioneer 360	0.313 ± 0.025	0.15	$\mathbf{0.062 \pm 0.001}$
Pioneer SLAM	0.272 ± 0.010	0.22	$\mathbf{0.094 \pm 0.015}$
Pioneer SLAM 2	$\mathbf{0.306 \pm 0.025}$	0.42	0.385 ± 0.009
Pioneer SLAM 3	0.413 ± 0.045	0.22	$\mathbf{0.111 \pm 0.004}$

Table 3.5.: Evaluation of the RMSE ATE of the Pioneer SLAM sequences. The results are averaged over 100 trials.

driving a significant distance over a set of cables while rotating on them (see Figure 3.6d). The uncertainty about the movement of the robot is underestimated by the previous calibration which was performed on a sequence with only local, comparably small disturbances on the floor of the hall. Furthermore, driving over the cables causes high motion blur in the camera images. Thus only few motion estimates from visual features can be computed in this part of the sequence due to only few available visual features. For these, often no depth information is available, due to the limited range of the sensor and the large size of the hall. The odometry error can therefore not be sufficiently corrected in this sequence.

3.3. Related Work

For computing the relative motion between two observations, graph based SLAM techniques for mobile robots often rely on scan matching, i.e., they apply the iterative closest points (ICP) [44, 150] algorithm on 2D laser scans [56, 115]. Recently, Röwekämper et al. [123] demonstrated that robot localization based on scan matching to reference scans can achieve an accuracy of up to one millimeter. A detailed overview of 2D SLAM techniques for mobile robots can be found in Thrun et al. [145, 146].

In the area of visual SLAM, the motion of the camera, respectively the robot, is estimated by using cameras as sensors. This is also often referred to as *structure from motion*. These SLAM approaches first focused on monocular cameras as sensors [25, 53, 140]. Monocular cameras, however, suffer from the problem that the scale cannot be observed. Therefore, stereo cameras have also often been used in SLAM scenarios [24, 149]. The known extrinsic calibration of the two cameras allows for a fast feature search and matching from which depth information can be computed more accurately.

Vision-based SLAM techniques, in contrast to laser-based systems, often extract visual properties like lines, planes, corners or descriptive features at detected key points. The motion is then estimated from matching these properties [83, 131, 149] or features between different image frames [25, 140]. Due to their distinctness, visual features often ease the process of data association in textured environments. Commonly used feature descriptors include SIFT [97] and SURF [7] as well as ORB features, recently introduced by Rublee et al. [125]. In our approach, we also use visual features to find correspondences between image frames. In experiments, we evaluate the performance of our system using the different feature types.

Camera-based SLAM approaches can perform well for estimating the trajectory of a mobile robot as well as for building a representation, e.g., a map of 3D features, for localization. For robot navigation, however, a sparse representation is often inadequate, as free and occupied space cannot be sufficiently distinguished. In addition, the accuracy of the depth measurements obtained from stereo cameras decreases with distance, due to the camera resolution. Thus, to create dense 3D maps of the environment, camera systems are often combined with laser scanners to provide dense and accurate 3D information for mapping [60]. As laser scanners are often bulky, heavy and expensive, this renders them unusable for some applications, e.g., handheld mapping or SLAM on low cost systems.

Recently, RGB-D cameras like the Microsoft Kinect or the Asus Xtion Pro Live were introduced. They provide a color image as well as a registered depth image of the environment which contains a depth measurement for each pixel in the color image. The depth image is obtained by projecting structured infrared light onto the scenery and perceiving it with an infrared camera. Using structured light renders these RGB-D cameras independent of the texture of the environment. However, it also makes them sensitive to illumination, in this case, by light with a high infrared component. This mostly constrains their usage to indoor environments and renders them inappropriate for use in direct sunlight. For typical indoor environments, however, RGB-D cameras combine the advantages of laser scanners and color cameras. They provide dense depth and the corresponding color information at a high frame rate, a low weight, and a low cost and are thus very suitable for our application.

There is a number of approaches that provide visual odometry for RGB-D cameras based on depth information only. Newcombe et al. [111] introduced KinectFusion, a volumetric voxel based reconstruction method where each voxel contains the truncated signed distance to the surface [23]. They achieve real time performance by directly creating the voxel representation online via high performance graphics card computations. Due to a high expense in memory used for the voxel representation, this system is only suitable for mapping small areas. Upon the development of KinectFusion, a number of systems were developed that use it as a basis [122, 138, 152]. Whelan et al. [153], for example, extended KinectFusion to cover larger areas. They define a fixed size cube volume in which the measurements are represented in the voxel representation and shift the cube along with the camera when it is moving. Those parts moving out of the cube are then meshed. These approaches mainly rely on iterative closest points (ICP) only to determine the frame-to-frame camera motion. Steinbrücker et al. [138] and Whelan et al. [153] use RGB information in addition and estimate visual odometry based on the consistency of the color images. Whelan et al. [153] achieve a comparable accuracy to our approach for the datasets tested. All of these approaches provide frame-to-frame visual odometry, yet they are prone to drift and can therefore result in inconsistencies in the global map. To obtain a metrically correct map, the drift has to be reduced as much as possible.

The first 3D mapping approach using RGB-D cameras and the one most related to ours was published by Henry et al. [62]. They use a combination of GICP [132] on point clouds and feature matching to estimate the relative motion between image frames. In detail, they extract sparse features from consecutive images and use the resulting motion estimate as initial guess for ICP. Henry et al. [62] also extended their algorithm to use ICP only, if few or no feature correspondences can be found. Given the found correspondences between image frames they then build and optimize a pose graph similar to our approach. In contrast to our approach, they generate a surfel representation of the environment. Hu et al. [70] propose another approach that aims at making the SLAM more robust in situations where no depth information is available for the visual features extracted. In this case they apply bundle adjustment based on the visual features only.

To effectively reduce drift, SLAM systems aim at finding loop closures, i.e., correspondences between image frames from different parts of the trajectory. Large scale visual SLAM systems applied in outdoor environments often rely on a place recognition system based on efficient feature extraction and search techniques to identify loop closures [22].

Recently, dense reconstruction methods have been extended from only providing visual odometry to be full SLAM systems. Kerl et al. [82] and Steinbrücker et al. [139], for example, proposed interesting approaches that minimize the photometric and geometric error over all pixels. They perform a metrical nearest neighbor search around key frames and then apply an entropy-based measure for loop closure detection. Stückler and Behnke [141] proposed an approach based on multi-resolution maps that integrate color and depth information. They rapidly extract maps from RGB-D images and use a variant of ICP to register them. Stückler and Behnke [141] perform a loop closure test based on the likelihood of matching their surfel representation. In our approach, we propose an efficient technique based on the incrementally built pose graph to detect loop closures. Specifically, our system applies a search strategy that combines the selection of previous frames, random search and search in the graph neighborhood. These systems based on dense reconstruction of the environment achieve a performance that is comparable to our approach.

The approach presented in this thesis, was one of the first published SLAM systems for RGB-D cameras. Compared to others, we performed extensive experiments to evaluate our approach on a publicly available benchmark dataset [142]. Furthermore, our SLAM system has been released as open source software to be used as a basis for new developments and for comparison.

3.4. Conclusions

In this chapter, we presented an approach to 3D mapping with an RGB-D camera. Our approach first extracts visual keypoints and features from color images. The registered depth image of the RGB-D camera provides the 3D information for each visual feature. We then estimate the relative motion between two image frames and use RANSAC on the matched features. To correct drift and compute a metrically consistent map, we build a pose graph from the estimated local motions and optimize it using a state-of-the-art nonlinear least-squares framework. As a final step, our system can compute different environment representations, e.g., an efficient 3D occupancy map that can be applied to mobile robot navigation. We performed extensive experiments on publicly available RGB-D benchmark datasets to assess the accuracy and performance of our approach. These datasets include different environments, ranging from small office rooms with many salient features to a larger industrial hall with few salient features. In addition, we evaluated the performance of our approach for the type of visual feature used. As a next step, we extended our system to take the odometry of a mobile robot into account by inserting it into the pose graph as an additional constraint. Our experiments showed that this can often improve the trajectory estimate of the camera especially in environments with few salient features. Finally, we published the presented system as open source software to encourage its usage by the scientific community and to allow for comparison to new approaches.

Chapter 4

Effective Coverage of Three-Dimensional Surfaces

In this chapter, we consider the problem of planning a path for redundant manipulators that covers the surface of a 3D object while minimizing a cost function in the joint space of the robot. Existing coverage solutions mostly focus on Euclidean cost functions and often return suboptimal paths with respect to the joint space. In our approach, we explicitly consider the joint space by treating different inverse kinematics solutions as individual nodes in a graph and model the problem as a generalized traveling salesman problem (GTSP). In addition to this general approach, we also propose an hierarchical approximation that is computationally more efficient. In the previous chapter, we introduced an approach to SLAM with RGB-D cameras and showed how a 3D model of an environment or a specific object can be obtained. Using such a registered point cloud as a prerequisite, we compute an object model that minimizes the number of nodes used in the planning problem. We evaluate our approach using a PR2 robot and different complex objects. The results demonstrate that our method outperforms Euclidean coverage algorithms in terms of manipulation effort and completion time.

In the previous chapter, we considered the creation of 3D maps and object models using the data of an RGB-D camera as input. We showed that from the reconstructed camera trajectory, different consistent environment representations can be acquired, which form the basis of many service tasks performed by domestic robots. In this chapter, we focus on the task of covering an arbitrary 3D surface for which we use a point cloud model of the object as a prerequisite.

Coverage of 3D surfaces is a problem that is increasingly gaining more attention due to its interesting potential applications such as the autonomous cleaning, painting, or

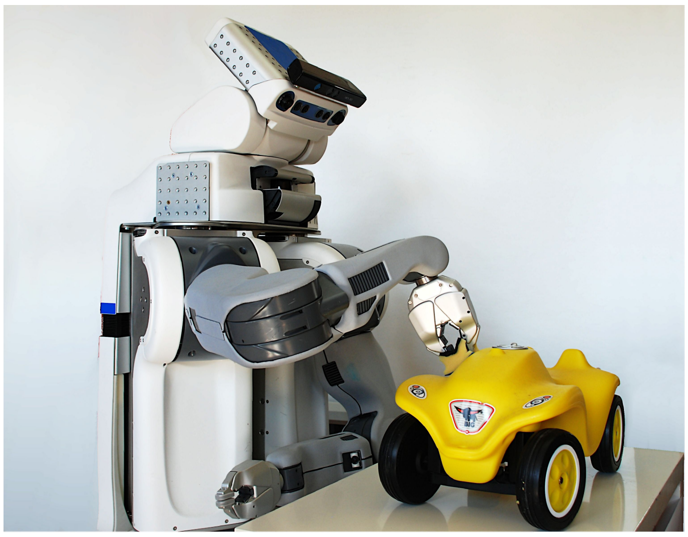


Figure 4.1.: PR2 robot using a sponge to clean a toy car.

scraping of complex 3D objects. All currently available domestic cleaning robots are either floor or window cleaning robots and thus operate on a 2D planar surface. Our approach is motivated by the concept of a service robot dusting or cleaning with a sponge in a household environment. In these environments, typical objects such as furniture and decorations can possess very complex shapes. Figure 4.1 shows an example where the robot is presented the task of cleaning the surface of a toy car. The surface of this car is complex and cannot easily be described as a set of simple geometric forms. Our goal is therefore to devise a novel approach that allows a manipulation robot to clean arbitrary 3D surfaces. Specifically, in this chapter, we consider the problem of coverage path planning for robotic manipulators, where the task space is constrained to lie on the surface and specific costs in joint space need to be minimized. We therefore assume that the orientation of the end effector is orthogonal to the direction of travel, which is the case for a variety of different applications and tools including paintbrushes, sponges, and squeegees.

A popular solution to the problem is to discretize the surface, transform it into a graph and solve the associated traveling salesman problem (TSP). Although this approach is well suited for minimizing the Euclidean path length, it limits the possibility to define and optimize appropriate cost functions in joint space. There are two reasons for this limitation. The first is that the orientation of the end effector is only known or can only be computed after a full path is available. The second reason is that the cost of travel from one node to another one depends on the configuration of the robot in the first node. This configuration is not unique for redundant manipulators and depends on the sequence in which the nodes are visited.

To overcome these limitations, we model the problem as a generalized traveling salesman problem (GTSP), a generalization of the TSP, where a set of clusters is defined over the nodes (see Section 2.4 for details). As a result, our approach generates coverage strategies that are optimized with respect to user-defined cost functions in joint space.

As the computational time of our approach is exponential to the number of nodes in the graph, we additionally propose an efficient hierarchical approximation. This approximation proceeds in two steps. It first optimizes the end effector trajectory, and, in a second step, the joint space trajectory.

We evaluate our approach using real data collected from a PR2 robot. Figure 4.1 shows the typical setup of our experiments, where the robot is cleaning the surface of a toy car using a sponge. The results show that our approach generates paths covering the object while minimizing the target cost. We evaluate our approach using two different cost functions, the completion time and the distance in joint space. According to the results, our method outperforms Euclidean coverage algorithms with respect to both cost functions. Our experiments also show that the hierarchical approximation we propose outperforms Euclidean coverage with respect to two user-defined cost functions in joint space at a significantly lower computation time than the full approach.

4.1. Formulating the Coverage Problem as a Generalized TSP

In this section, we formulate the coverage problem in terms of a GTSP. As already mentioned in Section 2.4, the GTSP is a generalization of the TSP, where a set of clusters is defined over the nodes. Each solution to a GTSP includes at least one node from each cluster (see Figure 2.4b for an example). As in the TSP, the goal is to find a tour of minimum cost. In our case, we are confronted with a special version of the GTSP where the clusters do not intersect and we search for a tour that visits each cluster exactly once.

First, we show how we convert the surface of the object that is given as a point cloud into a set of locally planar patches. Then, we describe how we use this representation to generate a Euclidean graph which encodes locally collision-free traveling paths over the surface. Finally, we use this graph to construct a GTSP that solves the coverage problem while minimizing a user-defined cost function in joint space. Figure 4.2 visualizes these steps of the planning process.

4.1.1. Modeling the Object Surface

Our robot is equipped with a Kinect sensor that generates a point cloud representation of the environment it senses (see Chapter 3). Using the techniques described in the same chapter, we can obtain an object model comprised of several point clouds generated from different points of view. Given a point cloud of the object to be covered, our goal is to first approximate it with a set of planar surface patches. A surface patch represents a part of the surface, i.e., a set of points on the surface, by a 3D position and a surface normal. This set of surface patches comprises the object model for the coverage task and is used as the basis for planning a path over the surface. As we discuss in the next sections, the complexity of the planning problem directly depends on the number of patches. Therefore, we aim at minimizing the number of patches for representing the surface.

In the following, we frame the problem of selecting a minimal set of patches as a minimum set cover problem (see Section 2.5). We first create a set of patches from which we then aim to select a minimal set. We are given a set $\mathcal{U}$ that contains the N points of the point cloud. For each point of the point cloud, we perform the following steps to construct a surface patch $\mathcal{S}_i$:

1. Determine the points that lie within an ε-neighborhood, where the value of ε depends on the size of the tool used to accomplish the task.
2. Use RANSAC [42] to fit a plane to the points selected in the previous step.
3. Accept the plane as a new surface patch if the root mean square error (RMSE) of the points to the fitted plane is below a threshold. The exact value of this threshold depends on the measurement noise of the sensor that was used for recording the point cloud.

All surface patches together form a collection of sets of data points $\mathcal{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_m\}$ with $m \leq N$. Selecting the minimal number of patches for representing the surface that cover all points in the point cloud corresponds to solving the minimum set cover problem. As mentioned in Section 2.5, Algorithm 3 achieves the best polynomial time approximation guarantee for this problem. We apply this greedy algorithm for solving the patch selection problem as follows:

1. Select the patch $\mathcal{S}_i$ that holds the maximum number of points of the point cloud that are not yet covered by a patch. If two or more patches fulfill this property, we randomly select one of these patches.
2. Mark the points represented by the selected patch as covered.
3. Continue until all points are covered by a patch.

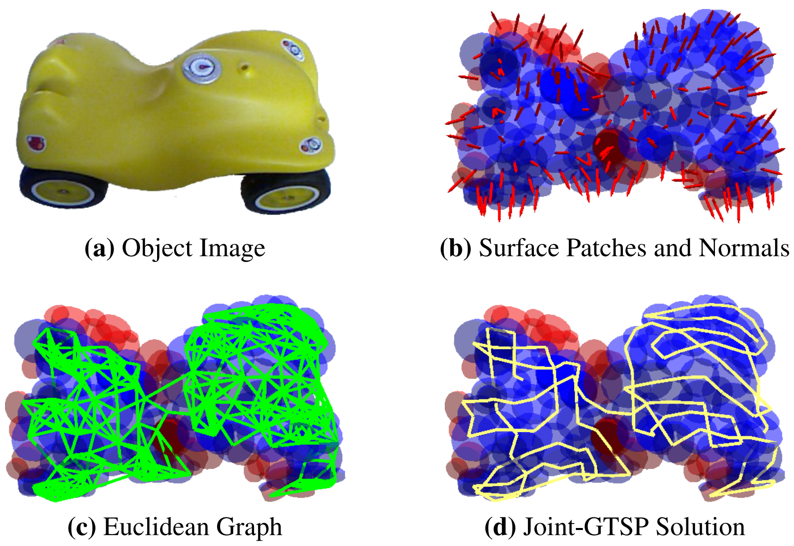

(a) Object Image **(b)** Surface Patches and Normals **(c)** Euclidean Graph **(d)** Joint-GTSP Solution

Figure 4.2.: Illustration of the different steps of the coverage planning process. Given the surface of the toy car (a), we discretize it into a set of surface patches and normals (b). The surface normals as well as patches that are not reachable by the robot are visualized in red. Based on the set of reachable surface patches, we compute a graph of locally collision-free traveling paths of the end effector over the surface (c). The Joint-GTSP aims at finding a path through this graph that minimizes the cost function in joint space (d). The figure shows the solution that minimizes the manipulation effort as defined in Eq. 4.2.

Figure 4.2b shows an example of the resulting surface representation. The blue patches in the figure are reachable while the red patches are not reachable from the current base position of the robot.

4.1.2. Euclidean Graph Construction from Point Clouds

In the previous section, we showed how to model the surface as a set of locally planar patches and their surface normals. We now aim at constructing a neighborhood graph of locally collision-free paths of the end effector on the object surface. Our approach considers the workspace currently reachable by the robot without additional movement of the base. However, in general, the movement of the base could be integrated similarly. In detail, we seek a graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges, with the following properties:

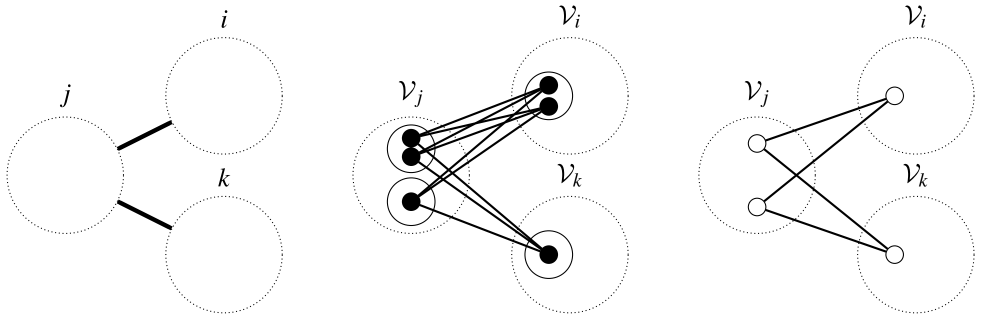
- The nodes correspond to the set of patches reachable from the current robot position.
- The edges represent locally collision-free paths for the end effector.

The construction of the graph proceeds as follows. We check the reachability of each patch from the current base position of the robot. If a patch is reachable, we mark it accordingly and insert the node into the graph. For each remaining node, we select the top k nearest neighbors within a radius r and connect them linearly with an edge. We then simulate an end effector movement along each of the edges and check them for collision. If there is no valid path along an edge, we delete the edge from the graph. As a result, each node in the graph can be reached on a collision-free path along the edges. If the graph construction results in more than one connected component, we apply the optimization approach described below to each of the components separately. Figure 4.2c shows the resulting graph for the patch set in Figure 4.2b. The red patches are not reachable from the current robot position. We frame the problem of covering the entire surface as traversing all nodes of the Euclidean graph, which corresponds to the set of surface patches selected in the previous section. Minimizing the Euclidean distance over the surface thus means finding a minimal path through the Euclidean graph, which we describe in Section 4.3 having implemented it for comparison. Instead of minimizing the Euclidean distance, however, we are more interested in minimizing cost functions in the joint space of the robot.

4.1.3. GTSP for Joint Space Minimization

For minimizing a user-defined cost function in the joint space of the robot, we formulate the problem as a GTSP. In contrast to the Euclidean distance on the surface, minimizing a cost measure in joint space poses the problem that the cost for moving the arm along an edge of the Euclidean graph depend on the previous movement. Thus the cost for moving to a new joint position depends on the joint position reached in the last step. As we show below, the GTSP allows for formalizing this property accordingly.

A GTSP is defined over a set of clusters of nodes (see Section 2.4 for a detailed description). In our case, we seek the shortest tour that visits each cluster exactly once and exactly one node per cluster. We obtain the GTSP by extending the Euclidean graph to account for the possible joint configurations for each pose. Each cluster $\mathcal{V}_i$ of the GTSP corresponds to a node i in the Euclidean graph. For each edge $e_{i,j}$ in the Euclidean graph, we first compute both the start pose i and the end pose j of the end effector given the patch and constraining the end effector orientation to the travel direction along the edge. We then sample a set of inverse kinematics solutions for the start and the end pose of the edge. These inverse kinematics solutions are samples in the null space of the robot and are due to redundancies. In the case of using a non-redundant robot arm, we obtain exactly one inverse kinematics solution for each pose. The solution sets are then inserted in the respective clusters and used in the GTSP. Nodes located within the same cluster are not



(a) TSP **(b)** Joint-GTSP **(c)** Curvature-GTSP

Figure 4.3.: The different graph representations we use for coverage planning. The surface patches correspond to the dotted circles. (a) The Euclidean graph used for computing the TSP solution. The center of the surface patches are points in 3D and form the nodes while the edges correspond to the edges in the Euclidean graph. (b) + (c) The graph representations used by the Joint-GTSP and the Curvature-GTSP. The solid circles correspond to 3D poses of the end effector. In the Joint-GTSP (b), we sample a set of joint space positions, denoted by the small black circles, for each end effector pose. We connect the nodes in the Joint-GTSP and the Curvature-GTSP according to the connectivity of the surface patches in the Euclidean graph (a).

connected. Figure 4.3b illustrates the resulting graph. The nodes corresponding to joint space solutions are marked solid black. The solid circles mark the ends of each edge in the Euclidean graph, while the dotted circles mark the clusters.

The final step is to define the cost functions of the GTSP in terms of joint space configurations. In our work, we are interested in two cost functions, namely the manipulation effort and the time required to complete the task. For simplicity, we assume the manipulator as being controllable in velocity and neglect the dynamics and accelerations. They can, however, be taken into account by appropriately modifying the expressions to be computed. We aim at defining distance measures which are metrics in the joint space, for which we need to consider the kinematic structure of the robot. The PR2 robot we use in this work (see Figure 1.1a and Figure 4.1) has two seven DoF arms. All arm joints of the PR2 move independently of each other and can rotate in both directions, clock and counterclockwise. In the following, we define the distance between two angular positions of a single joint. Let Δq_m be the minimum displacement of the m-th joint between two joint configurations $\theta_1, \theta_2 \in [0, 2\pi)$:

$$\Delta q_m = \min\{|\theta_1 - \theta_2|, 2\pi - |\theta_1 - \theta_2|\}. \tag{4.1}$$

This quantity for the difference between two angles is known to be a metric [91].

Let v_i^k and v_j^l respectively be the nodes k and l of the clusters $\mathcal{V}_i$ and $\mathcal{V}_j$ in the GTSP graph that correspond to two joint configurations. We then define the following two cost functions in the joint space of the robot:

Manipulation Effort Given Δq_m as defined in Eq. 4.1, we define the distance with respect to the manipulation effort as the total amount of displacement of each joint between two configurations:

$$d(v_i^k, v_j^l) = \sum_m \Delta q_m. \tag{4.2}$$

Given that Δq_m is always positive or zero, it is evident that Eq. 4.2 yields a metric.

Task Time With respect to the time required for the task, the distance is the minimal time needed for the movement between the joint configurations assuming a maximum rotational velocity ω_m of the respective joint:

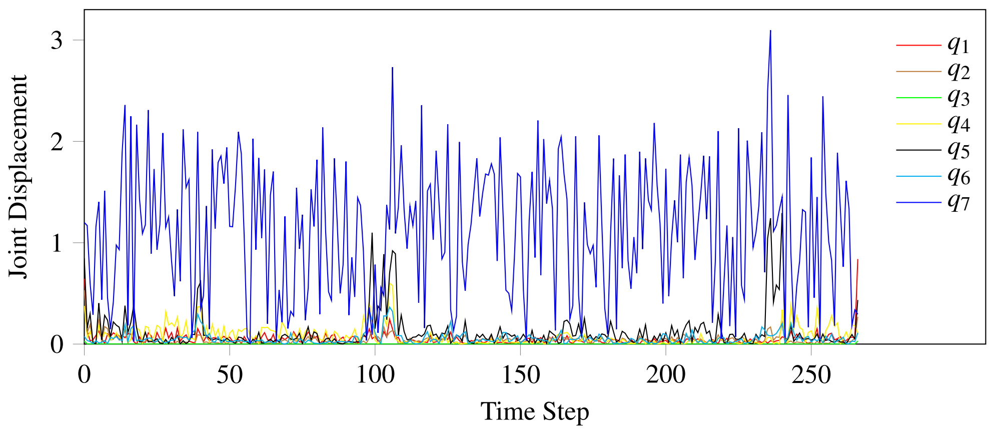
$$d(v_i^k, v_j^l) = \max_m \left(\frac{\Delta q_m}{\omega_m} \right). \tag{4.3}$$

If we require $\omega_m > 0$ and $\Delta q_m \geq 0$, one can see that Eq. 4.3 also yields a distance metric for joint space positions.

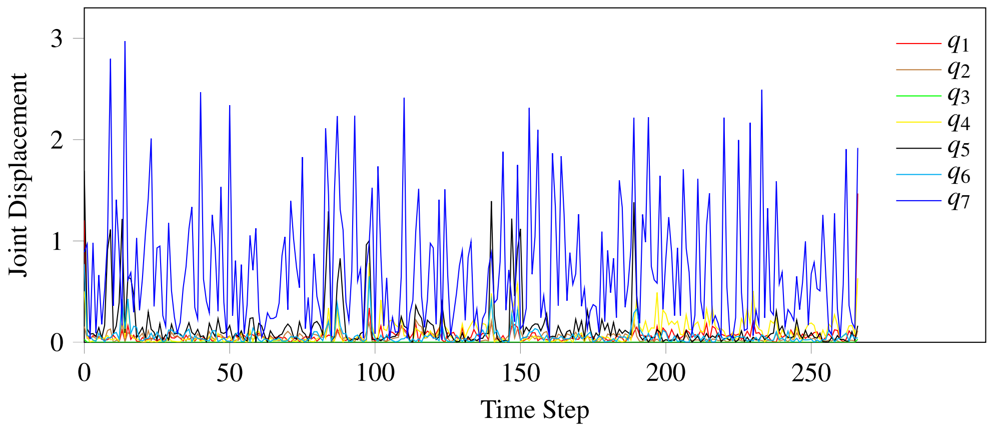
Having defined the distance measures, the edge costs in the GTSP graph are given as:

$$c(v_i^k, v_j^l) = \begin{cases} d(v_i^k, v_j^l), & \text{if } A(i,j) = 1 \text{ and } i \neq j \\ \infty, & \text{otherwise} \end{cases}, \tag{4.4}$$

where A is the adjacency matrix of the Euclidean graph. Two nodes i and j are adjacent in the Euclidean graph, i.e., $A(i, j) = 1$, if they are directly connected by an collision-free path of the end effector. As only the neighboring nodes are connected in the Euclidean graph, we generally cannot ensure that a Hamiltonian cycle can be found. In addition, deciding if a solution exists is known to be NP-complete [49]. However, for our application, we are more interested in covering every part of the surface than visiting each spot exactly once. Therefore, we relax this condition by fully connecting the graph to ensure that we can always find a solution. To each new edge in the graph, we assign the graph distance between the nodes connected by the edge, which we compute using the Floyd-Warshall all-pairs-shortest-path algorithm [45]. If such an edge were selected, the end effector of the robot would travel along the path over the surface for which this distance was computed. Having constructed this graph, we solve the GTSP by transforming it into a



(a) TSP



(b) Joint-GTSP

Figure 4.4.: Displacement of the single joints over a trajectories that cover the surface of a chair using different TSP solutions. Joint q_7 corresponds to the end effector. We computed solutions using the TSP (a) and the Joint-GTSP (b). As one can see, using the Joint-GTSP reduces the effort of joint q_7.

TSP (see Section 2.4) using the method of Behzad and Modarres [8]. They showed how to change the graph of a metric GTSP and how to compute the cost of the edges such that the resulting graph corresponds to a TSP and the number of nodes does not increase. As a result, we can once again apply a TSP solver on the transformed graph to obtain an effective coverage path with respect to the cost function selected. In the following, we refer to this GTSP approach for computing a coverage path as Joint-GTSP.

4.2. Hierarchical Approximation for Efficient Planning

Unfortunately, solving the GTSP described in the previous section is exponential to the number of nodes that correspond to the number of sampled inverse kinematics solutions. This limits us to using only a few inverse kinematics samples. In this section. we describe a hierarchical approximation of the Joint-GTSP that scales quadratically in the number of inverse kinematics solutions and only scales exponentially in the number of edges in the Euclidean graph.

To find a suitable approximation, we analyzed the cost profile of each joint in both the Joint-GTSP and the TSP solution that we implemented for comparison. The TSP solution minimizes the Euclidean distance and was obtained by using the Euclidean graph to find a solution, which we describe in Section 4.3. Figure 4.4 shows an example of the displacement of the single joints of the robot arm over trajectories computed by the Joint-GTSP and the TSP. As one can see, most of the gain of the GTSP solution results from the cost reduction of the joint corresponding to the end effector orientation. Minimizing the effort of this joint in turn means minimizing the curvature of the path in Euclidean space. Using this insight, we decided to decouple the full minimization problem by first optimizing for the end effector orientation and distance and then optimizing the remaining joints. More formally, we first generate a simplified GTSP problem that minimizes a weighted cost function on the Euclidean distance and the curvature of the path on the manifold (Curvature-GTSP). As in the general case (see Section 4.1.3), each cluster $\mathcal{V}_i$ of the Curvature-GTSP corresponds to node i in the Euclidean graph. We then consider only each start and end point of an edge in the Euclidean graph as a node in the Curvature-GTSP by constraining the orientation of the effector to point towards the direction of travel. Thus, the number of nodes in a cluster $\mathcal{V}_i$ only corresponds to the number of edges of node i in the Euclidean graph and is independent of the number of inverse kinematics samples.

Each node v_i^k in the graph defines its own local coordinate system, where the x-axis is oriented with respect to the direction of travel and the z-axis with respect to the surface normal of the patch. Let a node be represented as $\begin{bmatrix} R & \mathbf{x} \\ 0 & 1 \end{bmatrix} \in SE(3) \subset \mathbb{R}^{4\times 4}$ in the global frame of reference, where $R \in SO(3)$ is the rotation matrix and $\mathbf{x} \in \mathbb{R}^3$ is the translational part of the homogeneous matrix. For the distance of two rotations $R_1, R_2 \in SO(3)$, we apply the metric proposed by Park [117]:

$$\Phi(R_1, R_2) = \|log(R_1 R_2^\top)\|. \tag{4.5}$$

Intuitively, this metric measures the amount of rotation required to align R_1 and R_2. It results in distance values in the range $[0, \pi)$ [72]. The result of $log(R_1, R_2)$ is a skew-

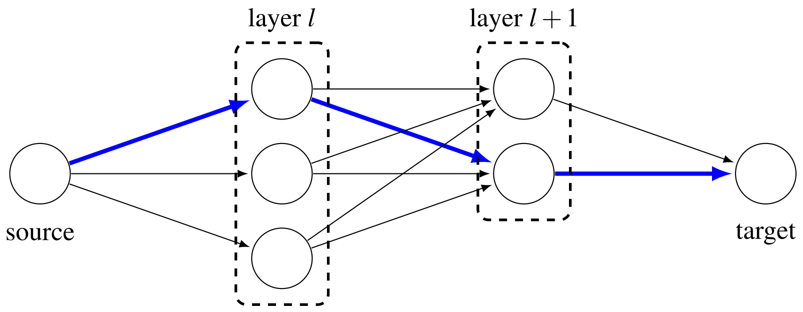


Figure 4.5.: Directed source to target graph built for a trajectory of end effector poses. Each node in the graph corresponds to a joint configuration and each layer to one end effector pose. The sequence of layers is given by the previously computed end effector trajectory. All nodes in one layer are connected forward to all nodes in the next layer. We apply the Dijkstra algorithm to solve for a solution, which is marked as the thick (blue) path.

symmetric matrix. The expression $\|log(R_1,R_2)\|$ thus results in the magnitude of the rotation angle. The distance between two nodes k and l of the clusters $\mathcal{V}_i$ and $\mathcal{V}_j$ is then given as:

$$d(v_i^k, v_j^l) = (1-\beta)\left\|x_i^k - x_j^l\right\| + \beta\,\Phi(R_i^k, R_j^l),\ \beta \in (0,1) \tag{4.6}$$

where β is a parameter weighting between the Euclidean distance and the curvature of the path. We require $\beta \in (0,1)$ to ensure that the convex combination of the Euclidean distance and the rotational distance metric in Eq. 4.6 remains a metric. Setting $\beta = 0$ or $\beta = 1$ would result in a pseudo-metric, i.e., there could exist nodes i, j with $i \neq j$ for which $d(i,j) = 0$.

The cost assigned to edges in the Curvature-GTSP graph are then computed in the same way as in the general setting described in Section 4.1.3. Figure 4.3c shows an example of such a graph. The dotted circles denote the nodes in the Euclidean graph and the clusters in the GTSP. The smaller circles denote the end points of each edge in the Euclidean graph and form the new set of nodes in the Curvature-GTSP.

As the second step, we solve the Curvature-GTSP using the same process of reduction to a TSP as we do for the general case described in the previous section. For further optimization of the path in joint space, we construct a directed source to target graph of joint space positions (see Figure 4.5). The graph is constructed in the following way. From the TSP solution we extract the sequence of 6 DoF end effector positions of the tour. For each position, we compute a set of inverse kinematics solutions that forms the nodes of an intermediate layer l of the graph. Thus, each layer corresponds to one end effector pose from the computed path and each node to one inverse kinematics solution. We select the current joint position of the manipulator as the source and connect it to the first layer. The first layer is determined as the position of the TSP solution closest to the current end effector position. All nodes in layer l are connected forward to all nodes in layer $l+1$. The target node of the graph is an artificial sink since we do not require an exact final joint position. Thus the weight of the edges that point from the last layer to the sink are all set to zero. The weight of each edge in this graph is equivalent to the cost functions computed in Eq. 4.2 and Eq. 4.3 between the joint configurations of the respective nodes in the Joint-GTSP. We then solve for a solution using the Dijkstra algorithm to compute the shortest path through the graph (see Figure 4.5).

4.3. TSP for Euclidean Coverage

For comparison to our approach, we compute a coverage path over the entire surface that minimizes the Euclidean distance for comparison to our approaches proposed in the previous sections. We frame the problem as a Euclidean traveling salesman problem as follows. To each edge $e_{i,j}$ in the Euclidean graph, we assign the Euclidean distance between the corresponding patches i and j as a cost. We then fully connect the graph as in the general case described in Section 4.1.3. A TSP solver is then directly applied on the resulting graph representation that is visualized in Figure 4.3a. Figure 4.6b shows an example result of such a path on a chair surface. After the tour construction, we compute the end effector orientation and optimize the joint space in the same way as for the Curvature-GTSP described above.

4.4. Evaluation

We validated our approach using real data recorded with the PR2 mobile manipulation robot. We recorded the data from two different objects, a chair and a toy car, using a Kinect sensor. A single view of each object was sufficient in this case. However, our approach also extends to multiple views that are fused, for example, using our approach presented in Chapter 3 or the method of Ruhnke et al. [126]. We chose the toy car and the chair for their differently structured surfaces. The toy car is a complex non-convex object whereas the chair is largely planar. For the construction of the Euclidean graph, we chose the number of nearest neighbors k to be 8 and set the search distance to 10 cm to limit the graph complexity. These settings resulted in a sufficiently dense graph. The β parameter in the cost function of the Curvature-GTSP (see Eq. 4.4) was set to 0.9 in order to favor

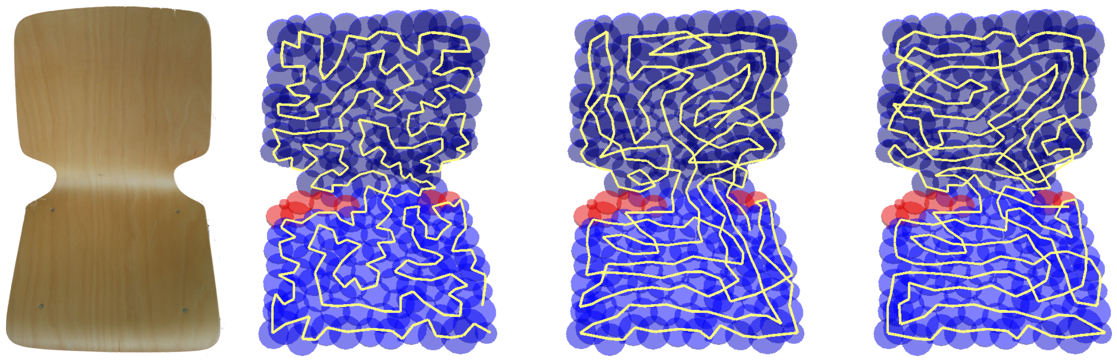


Figure 4.6.: Sample coverage paths for the chair experiment. (a) The surface of the chair that has to be covered. (b) + (c) + (d) The end effector trajectories through the reachable surface patches (blue) resulting from the different traveling salesman approaches. The red surface patches are not reachable from the current robot position.

solutions with smoother curvature changes. The exact setting, however, was not crucial for the experiments.

We evaluated three approaches, the Joint-GTSP, the Curvature-GTSP, and the TSP, with a different number of inverse kinematics samples. For the cases with a small number of samples < 10, we manually sampled the joint around the center of the configuration space. The 100 samples were obtained by sampling uniformly. We also simulated a non-redundant manipulator by fixing one of the joints. This is equivalent to the one sample case described below. We obtained the samples by fixing one joint and then using OpenRAVE [28] to calculate the inverse kinematics solution. The TSP has been solved using the LKH solver, a state-of-the-art TSP solver based on the Lin-Kernighan heuristic [61]. Due to the random element in the LKH solver, we repeated the experiments 10 times.

To illustrate the differences of the resulting Cartesian paths on the surfaces of the objects, a sample solution for both objects can be found in Figure 4.6 and Figure 4.7. Table 4.1 and Table 4.2 show the results of our experiments for the chair and for the toy car. For both objects and minimization strategies, i.e., effort and time (see Eq. 4.2 and Eq. 4.3), we compute the total Euclidean distance of the end effector on the surface and the calculation time. Depending on the minimization strategy, we also compute the total effort or the total time for task completion. In the calculation of the task completion time, we assume a maximum velocity of $\omega = 2$ for each joint and a bang-bang velocity profile that is equivalent to impulsive accelerations. The calculation time specifies the time needed for solving for a coverage path given the Euclidean graph.

	Task Time		
	TSP	Curvature-GTSP	Joint-GTSP
	1 Sample		
Eucl. Distance [m]	9.54 ± 0.00	12.03 ± 0.23	12.26 ± 0.18
Task Time [s]	153.46 ± 1.55	97.76 ± 2.17	**94.58 ± 2.15**
Calc. Time [s]	3.84 ± 0.36	114.40 ± 9.20	118.03 ± 6.68
	3 Samples		
Eucl. Distance [m]	9.54 ± 0.00	12.04 ± 0.12	13.03 ± 0.26
Task Time [s]	138.34 ± 1.47	87.15 ± 1.48	**84.53 ± 2.64**
Calc. Time [s]	3.73 ± 0.15	129.60 ± 7.76	1649.11 ± 150.90
	100 Samples		
Eucl. Distance [m]	9.54 ± 0.00	12.09 ± 0.16	—
Task Time [s]	101.60 ± 1.53	**69.91 ± 1.47**	—
Calc. Time [s]	23.92 ± 0.69	148.42 ± 8.64	—

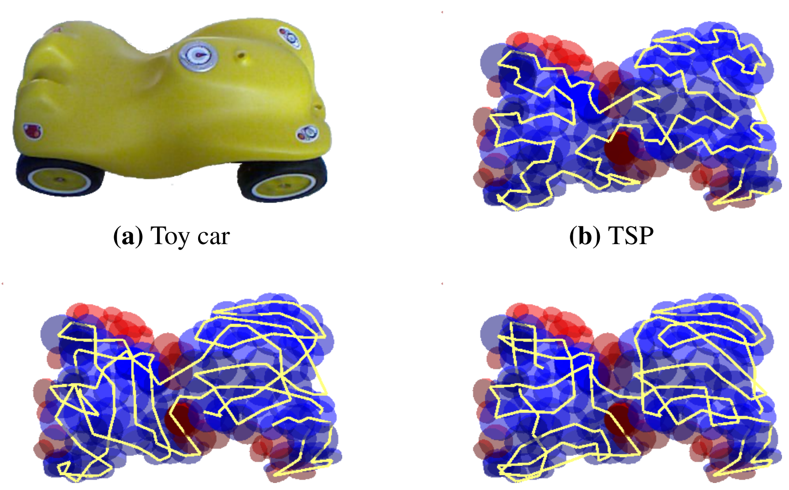
	Manipulation Effort		
	TSP	Curvature-GTSP	Joint-GTSP
	1 Sample		
Eucl. Distance [m]	9.54 ± 0.00	12.11 ± 0.25	12.09 ± 0.21
Manip. Effort	335.88 ± 3.27	228.70 ± 5.67	**215.65 ± 3.37**
Calc. Time [s]	4.12 ± 0.48	133.27 ± 8.07	131.06 ± 7.83
	3 Samples		
Eucl. Distance [m]	9.54 ± 0.00	12.19 ± 0.19	13.10 ± 0.35
Manip. Effort	305.39 ± 1.96	204.59 ± 3.28	**183.73 ± 4.02**
Calc. Time [s]	4.19 ± 0.31	137.88 ± 8.50	1808.60 ± 182.45
	100 Samples		
Eucl. Dist. [m]	9.54 ± 0.00	12.18 ± 0.30	—
Manip. Effort	206.81 ± 1.44	**134.49 ± 5.66**	—
Calc. Time [s]	12.01 ± 0.41	152.72 ± 14.40	—

Table 4.1.: Results for the chair experiment, optimizing for task time (top) and effort (bottom). The Joint-GTSP shows the best overall performance for those cases where a solution could be obtained. The Curvature-GTSP outperforms the TSP with a greatly reduced runtime compared to the Joint-GTSP.

	Task Time		
	TSP	Curvature-GTSP	Joint-GTSP
		1 Sample	
Eucl. Distance [m]	5.89 ± 0.00	7.61 ± 0.19	7.48 ± 0.15
Task Time [s]	85.00 ± 0.36	72.08 ± 1.65	**63.31 ± 1.59**
Calc. Time [s]	1.06 ± 0.25	30.01 ± 1.89	31.07 ± 3.38
		9 Samples	
Eucl. Distance [m]	5.89 ± 0.00	7.88 ± 0.22	8.49 ± 0.29
Task Time [s]	72.25 ± 0.56	61.60 ± 1.93	**60.85 ± 1.65**
Calc. Time [s]	1.15 ± 0.09	30.84 ± 1.86	1549.44 ± 183.41
		100 Samples	
Eucl. Distance [m]	5.89 ± 0.00	7.60 ± 0.15	—
Task Time [s]	68.14 ± 0.32	**54.86 ± 1.34**	—
Calc. Time [s]	4.29 ± 0.29	35.25 ± 3.43	—

	Manipulation Effort		
	TSP	Curvature-GTSP	Joint-GTSP
		1 Sample	
Eucl. Distance [m]	5.89 ± 0.00	7.49 ± 0.12	7.53 ± 0.16
Manip. Effort	205.33 ± 3.49	176.95 ± 6.95	**153.07 ± 3.57**
Calc. Time [s]	1.11 ± 0.24	32.16 ± 2.51	26.58 ± 0.81
		9 Samples	
Eucl. Distance [m]	5.89 ± 0.00	7.73 ± 0.14	8.97 ± 0.30
Manip. Effort	171.67 ± 2.22	143.83 ± 4.15	**137.50 ± 7.78**
Calc. Time [s]	1.17 ± 0.13	31.98 ± 2.10	2255.48 ± 213.98
		100 Samples	
Eucl. Distance [m]	5.89 ± 0.00	7.61 ± 0.16	—
Manip. Effort	159.70 ± 2.67	**131.88 ± 5.86**	—
Calc. Time [s]	2.47 ± 0.36	33.50 ± 1.94	—

Table 4.2.: Results of the toy car experiment. As for the chair experiments, we optimized for the task time (top) and for the effort (bottom). The result is similar to the chair experiment (compare to Table 4.1).



(a) Toy car **(b)** TSP

(c) Curvature-GTSP **(d)** Joint-GTSP

Figure 4.7.: Sample coverage paths for the toy car experiment. (a) The surface as viewed by the robot. (b) + (c) + (d) The end effector trajectories resulting from the different optimization techniques.

Table 4.1 and Table 4.2 show the results for one sample (no redundancy), the maximum number of samples usable for the Joint-GTSP (three for the chair and nine for the toy car), and 100 samples (only for the TSP and the Curvature-GTSP). As can be seen, the TSP results in the shortest Cartesian path but also in the highest effort and execution time. This comes with no surprise as it is not possible to encode these costs in the Euclidean graph. More interestingly, this also shows that the shortest Euclidean path is not always the best one with respect to execution time. The Joint-GTSP results in a significant reduction in terms of effort and time although the length of its Cartesian path increases. For both the one sample and the maximum samples cases, we see that the Joint-GTSP and the Curvature-TSP perform significantly better than the TSP in both experiments (minimum effort and minimum time) while the Joint-GTSP performs slightly better. If we increase the number of samples to 100, we see that the Curvature-GTSP is able to perform better than the Joint-GTSP at the maximum number of usable samples, with no significant overhead on the calculation time. While the presented solution results in a higher calculation time than the TSP, it has not been optimized for calculation time and is meant to show a principled reduction in execution time. In a practical application, the calculation time may also not play a dominant role in cases where identical objects have to be cleaned, e.g., cleaning all chairs in a restaurant. In those cases, one could store the computed path and reuse it if the same chair is recognized again using the methods by Steder et al. [137], for example.

4.5. Related Work

One of the earliest navigation systems for autonomous floor cleaning robots targeted for cleaning chain stores, is the SINAS system [92], which was developed as early as 1996. It is one example of a system that aims at executing a specific cleaning task by covering a known surface, otherwise referred to as *coverage path planning*. Choset [17] and LaValle [91] provide a detailed discussion and overview of this area.

Most of the approaches for coverage planning assume that the environment is known and seek the shortest path that traverses each location once. Practical solutions for 2D surfaces typically rely on heuristics to reduce the problem size, or they utilize specific structures of environments. Gabriely and Rimon [48], for example, decompose the surface into a grid and suggest different coverage strategies based on spanning trees. Other approaches use a decomposition of non-overlapping cells of different shapes. Latombe [90] use a trapezoidal decomposition. Oh et al. [113] propose a triangular cell map representation and suggest that it results in a shorter, more flexible path for a cleaning robot than a rectangular decomposition. Another approach is the Boustrophedon cellular decomposition [18], which divides the free space into cells that can be covered with vertical back and forth motions. Huang [71] uses this decomposition and compute a coverage path that minimizes the number of turns of the robot. Mannadiar and Rekleitis [103] propose a graph structure based on the Boustrophedon cellular decomposition and show that a complete minimal path through this graph can be computed in polynomial time. All of these approaches, however, assume that the robot is moving in a 2D plane. They also do not consider costs in the configuration space of the robot.

Recently, coverage algorithms have also been extended to non-planar surfaces. Xu et al. [156], for example, extend the work of Mannadiar and Rekleitis [103] to the field of aerial coverage of terrain with unmanned aerial vehicles (UAVs). Cheng et al. [16] focus on 3D urban coverage with UAVs. They approximate urban structures with geometric 2.5D features and plan a coverage path that considers the motion constraints of a fixed-wing plane. Coverage has also been addressed in the field of spray painting automotive parts [5, 6]. Atkar et al. [6] show how simple automotive parts like convex sheets can be covered such that the resultant paint deposition on the target surface achieves acceptable uniformity. In these applications, however, the robot does not directly operate on the surface. In addition, these approaches do not address the problem of minimizing costs in the configuration space of the robot.

In the field of inspecting 3D marine structures with sonar sensors, the work of Englot and Hover [38] is related to our approach. At first, they sample a set of feasible robot configurations from which the entire 3D structure can be observed. Similar to our approach, they then also solve a set covering problem. In a final step, they frame the path planning problem as a TSP. In contrast to our planning problem, Englot and Hover require several observations of the same part of the surface. In addition, they do not consider redundancies in the configuration space of the robot.

Breitenmoser et al. [12] extend 2D coverage for mobile robots to 3D surfaces by using Voronoi tessellations to map the surface to a 2D plane. Although closely related to our work, they only consider coverage in terms of minimizing the Euclidean distance.

In the area of manipulation planning for a robotic arm, the work of Saha et al. [128] is related to our approach. Given a number of sets of robot configurations, they aim at visiting one configuration of each set. The distance between the robot configurations of different sets is typically large in their application. Saha et al. therefore apply additional sampling based planning techniques to calculate valid collision-free paths. Due to the computation time, they aim at minimizing the number of point-to-point paths that have to be computed. In contrast to our approach, they therefore solve for a tour by means of computing a group Steiner tree and traversing its edges, which leads to revisiting those sets of robot configurations that correspond to the inner nodes of the tree.

The ideas presented in this chapter are also related to approaches in the area of planning paths for nonholonomic vehicles with curvature constraints (Dubins' vehicles). In this area, the path planning problem is also often framed as a TSP problem [93, 101, 130]. It is then called Dubins' TSP (DTSP) and was introduced by Savla et al. [130]. Le Ny et al. [93] show that the DTSP for planar environments without obstacles is NP-hard. They also propose different approaches for solving this problem, one of which frames the path planning problem similar to our *Curvature-GTSP* approach. Le Ny et al. [93] discretize the possible heading directions of a vehicle moving in a plane and construct a GTSP. To solve the GTSP, they use the same transformation to a TSP by Behzad and Modarres [8] as our approach and then apply a TSP solver. However, our approach differs in the used cost functions, the method used for constructing the graph, and the graph connectivity. Due to the complexity of the surfaces that we consider, we cannot assume that each surface patch can be reached on a straight line path from any other surface patch, for example.

In the area of robotic cleaning, Leidner et al. [94] recently proposed an approach to whole-body mobile manipulation for cleaning a surface. They use a high-level planner to parameterize mobile manipulation tasks on a symbolic and geometric level and then select an appropriate controller for the cleaning task. However, they do not explicitly consider cost functions in the joint space of the robot but plan a Cartesian path on the surface and then use, for example, an impedance controller to control the applied force. Our approach can also be extended to controlling the force applied to the surface. After solving for a solution, the use of a hybrid position/force controller [21, 43] would enable the robot to follow a trajectory in joint space while controlling the force applied to the surface.

In this chapter, we proposed a novel solution to the problem of covering 3D surfaces with a redundant manipulator. In contrast to the majority of previous work, our method goes further to address the problem of minimizing user-defined cost functions in joint space. In the area of biomedical imaging, Simaiaki et al. [134] apply our approach on a KUKA LWR arm to minimize the scanning time of probe-based laser endomicroscopy. They reconstruct the 3D surface of tissue and use our approach to cover it in order to create highly magnified visualization at the cellular level.

4.6. Conclusions

In this chapter, we presented a novel approach to minimizing cost functions in joint space for coverage path planning problems on 3D surfaces. Existing coverage algorithms mostly focus on robots moving on a planar surface, minimizing Euclidean properties on the plane. We showed that when considering manipulators, this property no longer holds, and costs in joint space need to be explicitly considered. Additionally, we showed how to express these costs by modeling the problem in terms of a generalized traveling salesman problem and presented a general framework for joint space optimization of arbitrary cost functions. Furthermore, we proposed an efficient approximation of the general approach that scales quadratically with the number of inverse kinematics samples. We evaluated the approach using real data collected from a PR2 robot. Results obtained with real-world objects show that we are able to obtain paths that cover the object while minimizing a user-defined cost function in joint space. They furthermore show that our general approach, as well as the hierarchical approximation, outperform Euclidean coverage algorithms in terms of manipulation effort and completion time. Interestingly, the experiments also show that the shortest path in Euclidean distance is not always the best one with respect to execution time.

The coverage approach presented in this chapter aims at visiting each location in the environment once. Some locations in the environment, however, may not have to be cleaned while others may need repeated cleaning. In the next chapter, we present a method that enables a manipulation robot to observe the outcome of its actions. As a result, we generate a trajectory that maximizes the effect of cleaning.

Chapter 5

Estimating the Effect of Cleaning Actions

When cleaning an environment, visiting every spot, the goal of our approach in the previous chapter, is often not required. Some parts of the environment might not need to be cleaned while others may need repeated cleaning actions. Knowledge about the dirt in the environment can therefore enable cleaning robots to generate cleaning paths that focus on the dirty areas of an environment and are thus more efficient. In this chapter, we present an unsupervised approach that enables a robot to learn how to effectively clean unknown surfaces by observing the outcomes of its actions. We model the cleaning task as a Markov decision process (MDP) where the state transition model is unknown and needs to be estimated. By observing the outcomes of its actions, the robot is able to learn this transition model. Based on this model, the robot can identify those parts of the surface that need to be cleaned, generate its actions accordingly, and verify its success. In experiments carried out with a mobile manipulation robot, we demonstrate that our approach leads to an effective cleaning strategy.

The approach developed in the previous chapter enables a manipulation robot to cover arbitrary 3D surfaces while minimizing a cost function in the joint space of the robot. For some tasks, such as dusting, covering each spot once can lead to a satisfactory result. In other cases, cleaning a table for example, some regions may not be dirty and thus do not require cleaning while other spots might even require repeated cleaning. These challenges can be overcome, if the robot can identify those parts of the surface that need cleaning.

In this chapter, we propose a vision-based approach that employs unsupervised learning and allows a manipulation robot to clean efficiently. The key idea is to identify the dirty

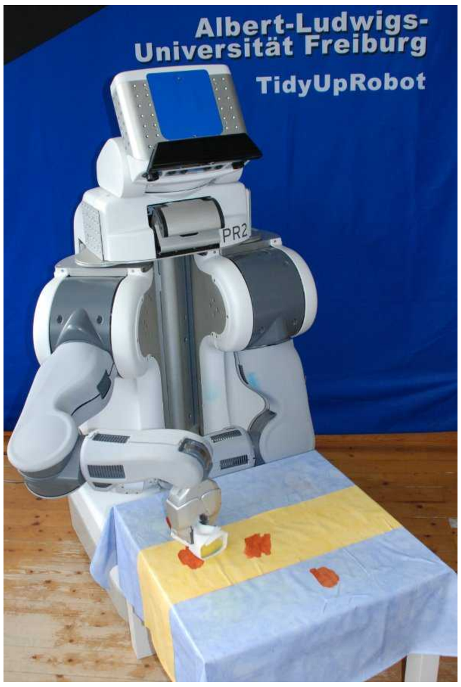


Figure 5.1.: PR2 robot using a sponge to remove dirt (seen in brown) from a table. Instead of cleaning the whole table, the robot observes the outcome of its actions, learns which features correspond to dirt, and cleans only the dirty regions.

parts of a surface as those that can be changed by a cleaning action such as wiping with a sponge. The robot learns about these parts by taking sensor data into account to observe the outcome of its actions.

We phrase this cleaning task as a Markov decision process. To be fully defined, a MDP requires an observation model that maps the input images to discrete states and a transition model that describes the effects of the robot's actions on the state variable. In contrast to existing approaches, we require neither the observation nor the transition model to be specified beforehand, i.e., the robot does not need to know which parts are dirty and which are clean. Rather, we enable the robot to learn these models autonomously by experimentation.

In our specific scenario, we consider the task of cleaning the surface of a table with a manipulation robot as shown in Figure 5.1. Using our approach the robot proceeds as follows. It divides the surface into a grid of cells and discretizes the color of each cell of the grid into a small set of discrete color classes. The state of all cells together with the position of the tool forms the state space of our approach. Initially the robot performs cleaning actions to learn the transition function between the color classes. It cleans cells which contain a large amount of a single color class. It repeatedly observes the outcome of such actions and updates its estimate of the transition function until it has converged. Afterwards, the robot selects actions according to the expected future reward. We define the reward function such that the robot favors actions that correspond to cleaning, i.e., actions that lead to changes in the class assignment of surface cells. The robot computes a cleaning trajectory exploiting its knowledge about the transition model and the state and computes an estimate for the new state in each step. It then executes the trajectory until it reaches the estimated desired table state. Subsequently, the robot scans the table again to determine the outcome of its previous cleaning actions. If needed it repeats the cleaning process.

Figure 5.1 depicts the experimental setup. The robot uses a sponge covered with a wiping towel that it holds in its end effector and that it can move across the table. By moving its end effector over the table, performing a wiping motion, and watching for changes in the table state, the robot learns which class labels indicate dirt that can be removed. By exploiting this knowledge, the robot seeks for the path that maximizes the effect of its cleaning actions. In experiments carried out on a real manipulation robot, we demonstrate that our approach allows a manipulation robot to clean a table efficiently and to verify its success.

5.1. Estimating the Transition Function

To solve the task of cleaning the surface of a table and to estimate which class labels correspond to dirt, we first formulate the problem as a MDP. Second, we describe how the current state of the table can be observed and knowledge about the class labels can be obtained. Third, we show how this information can be used to select the next best action and to generate efficient cleaning paths.

5.1.1. Problem Formulation

A MDP is typically defined as a tuple $(\mathcal{S}, \mathcal{A}, P(s' \mid s, a), R(s, a, s'))$, where $\mathcal{S}$ refers to the set of states and $\mathcal{A}$ to the set of actions. Whereas $P(s' \mid s, a)$ is the transition function for getting to state s' when taking action a in state s, $R(s, a, s')$ is the reward function that assigns a reward to taking action a in state s and ending up in state s'. In our cleaning task, we discretize the workspace into a set $\mathcal{C} = \{c_1, \ldots, c_M\}$ of M individual cells. We define a state $s \in \mathcal{S}$ as $s = \{s_1, \ldots, s_M, \mathbf{x}\}$, where $s_1, \ldots, s_M \in \{1, \ldots, K\}$ encode the state of the individual cells $c_1, \ldots, c_M$ of the workspace. Furthermore, let $\mathbf{x}$ represent the position of the robot's tool on the workspace and $\mathbf{x}_1, \ldots, \mathbf{x}_M \in \mathbb{R}^3$ denote the location of the cells. The robot's tool can be located in one of the cells resulting in M different locations. As a consequence there are $|\mathcal{S}| = MK^M$ different states, where K refers to the number of possible classes a cell can be in and M to the number of cells. The set of actions $\mathcal{A}$ is defined as moving the tool to one of the cells of the surface and cleaning it. We therefore consider $|\mathcal{A}| = M$ different actions, which the robot has to choose from in each time step.

The transition function $P(s' \mid s, a)$ consists of two parts: the movement of the tool and the change in the workspace. We assume the movement of the tool to be deterministic, always reaching the desired position in the workspace and only cleaning the desired cell. However, the outcome of the cleaning action on the state of the current cell is non-deterministic and initially unknown to the robot.

Let $P(s'_a \neq j \mid s_a = j, a)$ denote the probability that action a carried out in cell c_a assigned to color class j leads to a change of this color class, i.e., that cleaning is successful. Taking the above assumptions about the task into account, we then define the transition model as follows:

$$P(s' \mid s, a) = \begin{cases} P(s'_a \mid s_a, a) & \text{if } \forall i \neq a : s'_i = s_i \\ 0 & \textit{otherwise}, \end{cases} \tag{5.1}$$

where the current action a cleans cell c_a. In the expression above, if we take action a, the new position of the tool after the cleaning action is the location of cell c_a, i.e., $\mathbf{x}' = \mathbf{x}_a$. We assume that only cell c_a is changed by the cleaning action a and that the state of the other cells c_i remains unchanged. The transition probability for the entire state $P(s' \mid s, a)$ thus equals the transition probability $P(s'_a \mid s_a, a)$ of cell c_a.

We define the reward function $R(s, a, s')$ such that the robot gets a reward for every change it accomplishes in the table state, i.e., we assume that a change in the currently cleaned cell is an indicator of successful cleaning

$$R_{\text{clean}}(s, a, s') = \begin{cases} 1 & \text{if } s_a \neq s'_a \\ 0 & \textit{otherwise}. \end{cases} \tag{5.2}$$

This means that the reward obtained for cleaning a cell only depends on the change of the corresponding cell state s_a to s'_a as a result of this action. Further, we penalize the distance the robot has to travel from its current position $\mathbf{x}$ to position $\mathbf{x}_a$ for reaching cell c_a

$$R_{\text{travel}}(s, a, s') = -\left|\mathbf{x} - \mathbf{x}_a\right|. \tag{5.3}$$

If we combine these two components, we obtain

$$R(s, a, s') = \alpha R_{\text{clean}}(s, a, s') + (1 - \alpha) R_{\text{travel}}(s, a, s'), \tag{5.4}$$

where α is a weighting factor that trades off the cleaning result and the traveled distance. However, the robot cannot know whether cleaning a particular cell will lead to a change, as the successor state s', or more specifically, its a-th component s'_a corresponding to the state of cell c_a, is unknown. Therefore, the robot can only compute the expected reward of an action a using its current estimate of the transition function and the noisy observation of the cell state s_a, that is

$$\begin{aligned}\mathbb{E}[R(s,a,s')] =& \alpha \sum_{j=1}^{K} P(s'_a \neq j \mid s_a = j, a) P(s_a = j) \\ & - (1-\alpha)\left|\mathbf{x} - \mathbf{x}_a\right|,\end{aligned} \tag{5.5}$$

where $P(s_a = j)$ corresponds to the probability that cell c_a is assigned to color class j.

Given the reward function, the robot can now compute the expected future reward, also called value of a state s, as

$$V(s) = \max_a \left\{ \sum_{s'} P(s' \mid s,a) \left(\mathbb{E}[R(s,a,s')] + \gamma V(s') \right) \right\}, \tag{5.6}$$

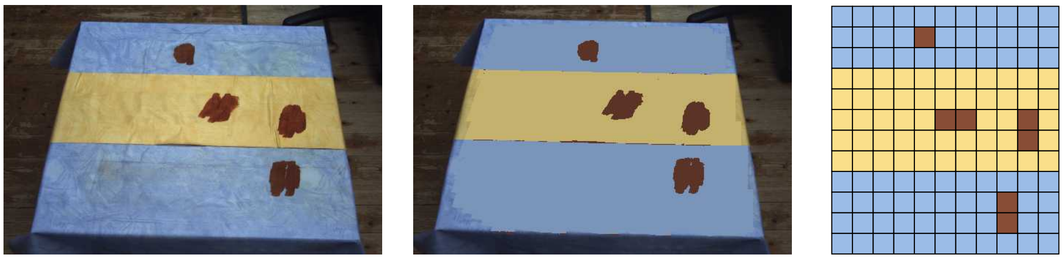
where γ is a discount factor. One way for computing $V(s)$ is to apply value iteration. Given $V(s)$, the optimal policy and thus the best action that the robot can choose in state s can then be calculated according to

$$\pi(s) = \arg\max_a \left\{ \sum_{s'} P(s' \mid s,a) \left(\mathbb{E}[R(s,a,s')] + \gamma V(s') \right) \right\}. \tag{5.7}$$

In each time step, the robot therefore computes the value function $V(s)$ and executes the optimal action given the resulting optimal policy $\pi(s)$.

5.1.2. Perception

In order to determine the workspace in our approach, the robot requires a color image of the scene as well as the depth information corresponding to the image pixels. As in the previous chapters, we use an RGB-D camera attached to the manipulation robot to obtain this information and compute an RGB-D point cloud. In particular, we carry out the following steps to determine the state of the table. We first detect the table by extracting the most dominant plane in front of the robot that has a normal vector approximately similar to the ground plane. After determining the points of the point cloud that are inliers of this plane, we compute their convex hull, which we consider as the surface to be cleaned. We then divide the surface into a regular grid, choosing the size of the cells such that a single



(a) Original image **(b)** Discretized image **(c)** Table state

Figure 5.2.: Example of the table state extraction. (a) The original input image. The depth information provided by the RGB-D camera allows us to identify the table surface. The part of the image that corresponds to the surface is then clustered into a set of color classes (b). In a final step, the surface is discretized into a regular grid and the current table state is extracted (c). The cells in the figure are colored according to the predominant color class occurring in this cell.

cell can be covered entirely by the robot's tool. In our case, the individual cells have a side length of 4 cm. Given this grid, we assign the pixels of the color image to the individual cells. Finally, we discretize the colors of the grid cells into a set of distinct color classes.

5.1.3. Automatic Color Clustering

We apply K-means clustering [102] to learn a suitable mapping from RGB colors to cell states. This allows the robot to autonomously bootstrap an observation model that is suitable for its current environment. To achieve this, we use the expectation maximization (EM) algorithm as implemented in OpenCV [11]. In preliminary experiments, we found that the *Lab* color space yields the best results for our application. The number of color classes could be selected automatically, e.g., by applying an extended version of K-means clustering [119]. In our current implementation, however, we set it manually (see Section 5.2). Figure 5.2 shows an example of the result of color clustering and table state discretization. Whereas the input image is shown in Figure 5.2a left, the result after table detection, segmentation, clustering, and color discretization is depicted in Figure 5.2b. As each cell consists of multiple pixels that may belong to different classes, we compute the class distribution for each cell, i.e., the probability $P(s_i = j)$ that the cell c_i belongs to class j. Figure 5.2c depicts the discretization as well as the dominating color class per cell. An example of the class distribution of one specific cell, before and after cleaning, is shown in Figure 5.3. As one can see, the brown components (class 1) were removed by cleaning this cell. Most of these pixels transitioned to the yellow class (class 2) and some to the blue class (class 3).

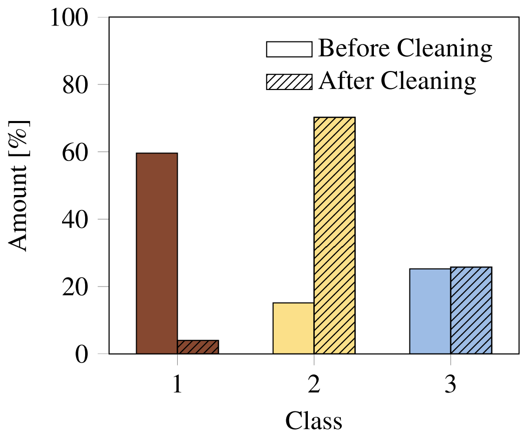


Figure 5.3.: Class distribution of a cell before and after a cleaning action. The dirt corresponding to class 1 (brown) is reduced and mostly changes to class 2 (yellow).

5.1.4. Learning the Transition Matrix

Our main goal is to learn the transition model $P(s' \mid s, a)$. As the immediate reward for cleaning only depends on the state change of individual cells (see Eq. 5.2), it suffices to learn the transition function for cells. Concretely, we estimate $P(s'_a \mid s_a, a)$ by means of a counting matrix $T \in \mathbb{N}^{K \times K}$, which reflects how often the robot has observed a (pixel-wise) transition between color classes. The elements t_{ij} of T thus correspond to the absolute frequency that a pixel of class i has changed to class j through cleaning. For each pixel located in the cell that the robot acted upon and observed, we increase the count t_{ij} by 1. From this counting matrix, we compute an estimate $\hat{P}$ of the transition model $P(s'_a \mid s_a, a)$ as follows:

$$\hat{P}(s'_a = j \mid s_a = i, a) = \frac{t_{ij}}{\sum_{j'=1}^{K} t_{ij'}}. \tag{5.8}$$

We initialize each element of the counting matrix to a small value to express our prior belief that transitions between all color classes are equally likely.

In the following we let $\hat{P}_{i,j} = \hat{P}(s'_a = j \mid s_a = i, a)$. To determine when the transition model has converged we compute the variance of the estimator $\hat{Var}(\hat{P}_{i,j})$, which is given as

$$\hat{Var}(\hat{P}_{i,j}) = \frac{1}{\sum_{j'=1}^{K} t_{ij'} - 1} \hat{P}_{i,j}(1 - \hat{P}_{i,j}). \tag{5.9}$$

We define a threshold criterion based on this estimated variance:

$$\max_{i,j} \hat{Var}(\hat{P}_{i,j}) < \beta, \tag{5.10}$$

and thus stop learning when all entries of the probabilistic transition matrix have converged.

5.1.5. Determining When to Stop Cleaning

Given the current estimate of the transition function, we can compute the expected number of cells that the robot can clean as

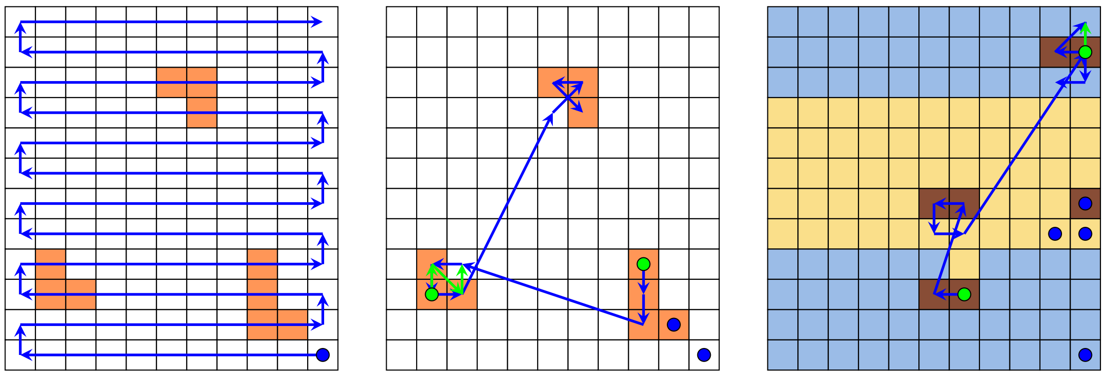
$$\delta = \sum_{a=1}^{M} \sum_{j=1}^{K} \hat{P}(s_a \neq j \mid s_a = j, a). \tag{5.11}$$

A possible criterion for prompting the robot to stop cleaning the table is to check whether δ drops below some threshold, for example, $\delta < 1$ would let the robot stop cleaning if it expects that, in sum, less than one uncleaned cell remains. In practical experiments, we found that a threshold on the expected change of δ after the next cleaning step also yields a stable criterion. The robot therefore stops cleaning when the expected change it can induce by its cleaning actions is estimably small.

5.2. Evaluation

All experiments were carried out on the PR2 mobile manipulation robot from Willow Garage. We set up a table and positioned the PR2 such that the entire table could be reached with one arm (see Figure 5.1). As a tool, we used a sponge covered with a cleaning towel. The goal of our experiments is to demonstrate that our approach allows the robot robustly to learn the state transition model, which in turn allows for increasing the speed and efficiency of table cleaning. We evaluate our approach in two different environments, using tables with different dirt and cloth colors.

In our experiments, we set the threshold for determining if the learning of the transition matrix has converged to $\beta = 0.01$ (see Eq. 5.10). Furthermore, as we are mostly interested in estimating the effect of cleaning by learning the transition model, we reduced the policy search problem to a greedy search, which can be solved online. In preliminary experiments, we found that this simplification does not significantly affect the performance of our system. This is mainly due to the small environment where small movements of the robot arm suffice to reach each part. However, if the travel time between different locations on the table played a more significant role, the order of execution would become more



(a) Sweeping pattern **(b)** Our strategy (Experiment 1) **(c)** Our strategy (Experiment 2)

Figure 5.4.: Sample results of the experiments. (a) The sweeping path we compare our approach to that covers all cells. (b) + (c) The path followed by our approach in one trial of the two experiments we performed. The cells marked by blue dots denote cells that were cleaned until the transition model converged. The other dots mark the beginning points of the trajectories. Upon traversing the first trajectory (shown in blue), the robot observed the state again and applied additional cleaning motions (shown in green).

important and thus more steps would need to be planned ahead. We set the threshold used for deciding when to stop cleaning to $\delta = 0.03$ (see Eq. 5.11). We found this value through manual visual inspection of the cleaning result after several trials.

Using our approach, the robot proceeds as follows. In the first step, the robot learns the transition model by observing the outcomes of its actions. It selects those cells for learning that hold the maximum amount of image pixels belonging to each color class. After cleaning a cell, the robot observes again and updates the probabilistic transition matrix. After the convergence of the transition model, the robot stops learning and exploits this knowledge by planning a trajectory that provides a sufficiently clean table in expectation. This is done by taking into account the current estimates of the environment and the transition model. The robot then executes the planned trajectory, observes the state again and verifies its cleaning result. If required, it plans an additional cleaning trajectory.

Figure 5.4 shows the application of our approach to the two different environments we tested and also visualizes the sweeping strategy that we compare our approach to in Figure 5.4a. In the first environment, the table surface is white. We added orange paint that the robot could remove using the sponge and set the number of color classes to $K = 2$ according to the number of distinct features present in this environment. Figure 5.4b shows a single run of our cleaning approach in the first environment. The blue dots mark the cells used for learning the transition model. Due to the large number of observed class changes,

		🟧	⬜
class 1	🟧	0.07	0.93
class 2	⬜	0.01	0.99

(a) Experiment 1

		🟫	🟨	🟦
class 1	🟫	0.04	0.96	0.0
class 2	🟨	0.03	0.97	0.0
class 3	🟦	0.0	0.0	1.0

(b) Experiment 2

Figure 5.5.: Learned transition matrices. The classes corresponding to each row/column are depicted with their mean color. Rows: Class label before cleaning. Columns: Class label after cleaning.

the learning converged very quickly. As the figure shows, only two steps were necessary to learn the transition matrix. Figure 5.4b shows the actually executed trajectory. As we assume that the expected class change is initially the same for all cells, the difference in reward for cleaning different cells only results from the distance to be traveled. Thus, the path starts at the table cell closest to the initial gripper position, which is located in the right bottom corner. While the robot starts to execute this path, it re-observes the table state and updates the transition matrix. In particular, it detects that it can clean orange cells. As a consequence of this update, it re-plans the cleaning trajectory.

Figure 5.5a shows the state transition model corresponding to this run. The values on the diagonal indicate the probability that a cell of this color is not affected by cleaning. In this experiment, the robot estimated the probability of not being able to clean the orange (class 1) and white (class 2) cells to 0.07 and 0.99, respectively. In contrast, the robot learned that orange cells (class 1) transition with a high probability of 0.93 to white after being cleaned. The value is not exactly one as a result of noise in the observations of the robot. This experiment shows that the robot correctly recognized that the orange color class is cleanable and the cells corresponding to the white table are immutable. Figure 5.4b depicts one of the cleaning trajectories followed by the robot in blue. After following the trajectory, the robot observed the table again. As the desired dirt ratio was not yet reached, it executed another trajectory (green) and thus additional cleaning steps.

To demonstrate the robustness and stability of our approach and to show its efficiency in terms of cleaning speed, we repeated this experiment ten times. Each time the robot reached the desired dirt ratio. After every run, we set the table back to its initial state. In this experiment, we compare the execution speed of our approach to the systematic cleaning of the entire table. To generate a path for the entire table, we manually devised a sweeping pattern which is shown in Figure 5.4a. Table 5.1 shows the result of our experiment. The execution time subsumes all steps needed to fulfill the task, including

	Learning Time [s]	Execution Time [s]	Cleaning Steps
Our Approach	22.8 ± 1.24	93.1 ± 25.7	16.5 ± 7.6
Sweeping Approach	-	393.2 ± 4.3	132

Table 5.1.: Results of the first experiment. The learning time denotes the entire time needed until the transition model converged. The execution time subsumes all steps, including the learning time, the cleaning itself, and the verification of the result. The number of cleaning steps denotes the number of cells needing cleaning.

the time needed for learning, cleaning, and observing the result. A t-test showed that the number of cleaning steps as well as the overall execution time of our approach are significantly lower compared to wiping the entire table in this scenario. Furthermore, although the robot has to remove its arm entirely from the scene to observe the state of the table and to learn the transition model, the learning time is not the dominating factor. Of course, these results depend on the state of the table, especially on the amount and distribution of the dirt as well as the number of color classes. In many day-to-day situations, however, it is not necessary to clean the entire surface every time, for example, when removing a stain from a table or a counter top. Rather, one can often focus on the dirty parts of the environment. In our approach, the ability to verify the cleaning result and to focus on areas that need additional cleaning effort is a substantial advantage over systematic or randomized cleaning strategies.

In addition to the first experiment, we applied our approach to a different environment. For the second experiment, we covered the table with the blue and yellow table cloth as shown in Figure 5.1 and again used paint as dirt. For this experiment, we set the number of classes to $K = 3$. Figure 5.4c shows the trajectory followed in one of the runs in this environment. The resulting behavior is similar to the first experiment. The robot correctly recognizes that the brown color class is cleanable and thus specifically approaches those cells (see Figure 5.4c). The learned transition model is depicted in Figure 5.5b. There is a high probability for class 2 and class 3 that they are not affected by cleaning. In contrast, class 1 has a low probability of remaining brown. Compared to the first experiment, there is a small probability that class 2 changes to class 1 that corresponds to dirt. This result is attributed to an increasing noise in the clustering at the edges between the yellow and the blue part of the surface and is also due to the table cloth which was slightly structured. During the learning process, it may also happen that not all dirt stays on the sponge but that a small part gets distributed to a clean cell. Due to the slightly increased noise and the increase in color classes, the number of cells observed until the convergence of the transition model also increased slightly to four. Figure 5.4c shows the positions used for learning the transition model in blue.

5.3. Related Work

Robotic cleaning is often framed as a coverage path planning task, i.e., planning a path for a cleaning robot that covers each part of a known surface (see Section 4.5 for a detailed description). In contrast to these approaches, we seek for the path that maximizes the cleaning performance and specifically covers those parts of the surface that the robot can actually clean. Covering specific parts of an area has also been addressed in multi-robot exploration and routing tasks [89, 159]. Zlot et al. [159] generate possible exploration goals and frame the problem of visiting all goals as a TSP. They approximate the TSP solution by choosing the closest unexplored node as the next goal. In principle, this strategy could also be applied to find optimal cleaning routes, i.e., the shortest trajectory that visits all dirty cells. However, in our problem setting, the robot cannot know which cells are dirty before it starts cleaning. Therefore, we frame the cleaning problem in this chapter more generally as a Markov decision process. The robot then acquires a probabilistic belief of the dirtiness of each cell during the execution of its task. Furthermore, the robot might need to clean particular cells more than once. In our approach, we realize this through observation and re-planning.

In the area of surface cleaning with a manipulation robot, Urbanek et al. [151] have developed a system that allows a robot to learn a low-level motor controller for wiping motions. They propose that the robot can learn the parameters of an oscillator from human demonstrations, and they demonstrate that their system can learn and reproduce different wiping styles, e.g., zigzag or roundish movements, from teacher demonstrations. Eppner et al. [39] present an approach that enables a robot to infer generalized task descriptions based on imitation learning including cleaning tasks. However, since the robot imitates the task at a trajectory level, it cannot substantially modify the path during reproduction and the resulting motion is independent of the current dirt distribution. In contrast to that, our approach observes the state, reacts to it, and allows the robot to recognize that the surface has been successfully cleaned.

Describing effects of actions on the world state has seen recent interest in the area of affordance learning. Sahai et al. [129] use a robot to estimate which tools are suitable for writing on different materials. Metta and Fitzpatrick [105] propose an approach that allows a robot to learn a categorization for objects as rollable and unrollable. Griffith et al. [54] also categorize objects based on movement patterns resulting from interaction with the object and have created a model to apply the learned categories to new objects. Do et al. [30]

have taught robots to learn associations between object properties (softness and height) and action parameters for the wiping task. As a result, the robots can make predictions about the effects of using novel objects such as a tool for wiping.

In the area of mobile robot navigation, Frank et al. [47] describe the effects of a robot interacting with deformable objects. They show that navigation tasks, such as passing through a curtain, can be performed more efficiently when one considers the cost of deformation. Ziebart et al. [158] also model path planning as a MDP where they assume that the reward function is a linear combination of the features of the environment. Using inverse reinforcement learning, their goal is to infer the reward function that underlies the human demonstrations. Based on the maximum entropy model, they estimate the weights corresponding to selected features from observed trajectories. This enables them to predict, for example, the chosen trajectories of pedestrians or taxi drivers. In contrast to learning the reward function, our goal is for the robot to learn the effects of its actions, i.e., the transition model of the MDP.

5.4. Conclusions

In this chapter, we presented an unsupervised approach that enables a manipulation robot to learn how to clean surfaces. Our approach phrases the cleaning task as a MDP where the state transition model is initially unknown. The robot learns the transition model by observing the outcomes of its actions. The robot then exploits its knowledge and plans a trajectory After the convergence of the transition model, the robot exploits this knowledge by planning a trajectory that results a sufficiently clean table. After the execution of the trajectory, the robot observes again and, if required, plans additional cleaning actions. In practical experiments carried out with a real robot, we demonstrated that our approach effectively enables a service robot to learn a cleaning task by learning how to recognize dirt and by focusing its actions on the dirty regions. As a result, our approach can determine areas that need additional cleaning and can verify the cleaning result. In addition, we demonstrated that depending on the environment, the resulting policy can result in a shorter cleaning time compared to a systematic cleaning strategy.

In the next chapter, we transfer the idea of observing the outcome of cleaning actions to a vacuum cleaning robot. Compared to the approach presented in this chapter, the robot cannot directly observe the dirt in the environment but can only observe the dirt it has cleaned. By learning the dynamics of the generation of dirt, we can predict the state of the environment at the time of cleaning. Based on this prediction, we propose efficient cleaning policies for vacuum cleaning robots.

Chapter 6

Estimating the Distribution of Dirt in the Environment

The ability to estimate where the dirt is in an environment makes it possible to compute efficient paths for robotic cleaners. In this chapter, we present a novel approach for modeling and estimating the dynamics of the evolution of dirt in an environment. While our vision-based method for identifying dirty parts in the environment (presented in Chapter 5) assumed that the entire workspace can be observed, this is often not the case. Floor cleaning robots, for example, can only observe a small part of a potentially large workspace. To clean efficiently, they therefore need to be able to make predictions about the current dirt distribution. Our model uses cell-wise Poisson processes on a regular grid to estimate the distribution of dirt in the environment. It allows for an effective estimation of the dynamics of the evolution of dirt and also allows for making predictions about the absolute dirt. We furthermore propose two efficient cleaning policies that are based on the estimated dirt distributions and can easily be adapted to the different needs of users. We demonstrate the effectiveness of our approach both through extensive experiments carried out with a modified iRobot Roomba vacuum cleaning robot and in simulation.

In Chapter 5, we presented a vision-based approach that enables a manipulation robot to learn to clean surfaces. By observing the results of its wiping actions, the robot learned a model for predicting the effects of its cleaning actions on the environment. The robot exploited this knowledge by planning an effective cleaning trajectory that focuses on the dirty parts. The approach we present in this chapter follows a similar idea that knowledge about the distribution of dirt in the environment can be used to plan more efficient cleaning paths. In contrast to the previous chapters, this chapter focuses on providing efficient

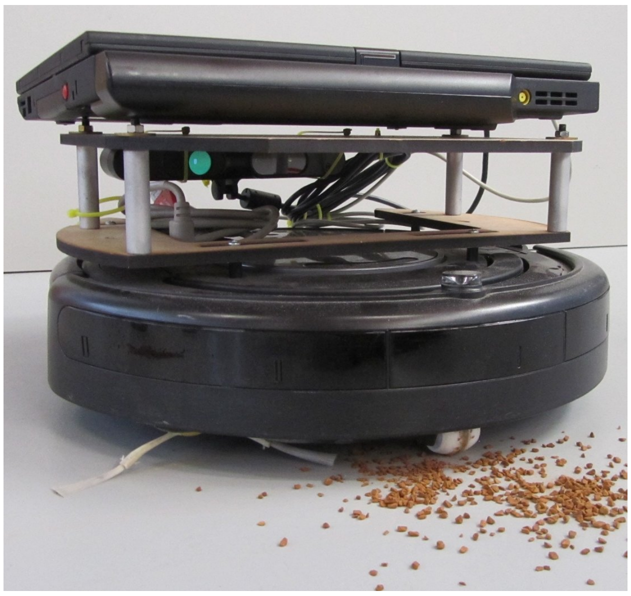


Figure 6.1.: Roomba 560 robot used in the experiments. The vacuum cleaning unit of the robot is equipped with a dirt detection sensor. The mounted Asus Xtion Pro Live depth camera is used for localization.

cleaning strategies for floor cleaning robots. The very first floor cleaning robots, such as the iRobot Roomba, did not systematically clean the environment but rather performed predefined motion patterns mixed with random movements [133]. Accordingly, such systems in general cannot guarantee the success of the cleaning process, especially when their operation time is limited. Recently, vacuum cleaning robots have been developed that are able to systematically clean an environment, e.g., the Neato XV, the Samsung Navibot, and the Evolution Robotics Mint. These robots apply simultaneous localization and mapping (SLAM) procedures to estimate the pose of the vehicle, and this enables them to clean the floor in a more systematic fashion. However, as already argued in Chapter 1, some parts of the environment, such as the entrance area of a home or the kitchen, might become dirty more frequently than other areas such as the living room.

In this chapter, we consider the problem of estimating the dirt distribution on the floor of an environment and of planning efficient cleaning paths given this distribution. In the previous chapter, the robot could observe the entire workspace at once. Compared to this approach, we face the additional challenge that the floor cleaning robot can only observe its cleaning result in the part of the workspace where it is currently located at. To efficiently clean, the robot therefore requires a prediction about the distribution of dirt in the environment at the beginning of a cleaning cycle, even for those parts that it cannot currently observe.

For estimating the dirt distribution and for computing efficient cleaning policies, we assume a cleaning robot that can localize in the environment given an occupancy map. We additionally assume that the robot is equipped with a sensor that can robustly detect the dirt that it cleans. To model the dirt distribution, we divide the environment into regular grid cells and apply cell-wise Poisson processes for estimating the amount of dirt in every cell. The corresponding dirt values grow over time and are reset by the cleaning operation. We show that this model enables the effective estimation of the dynamics of dirt generation and the prediction of the absolute dirt values.

To exploit the estimated dirt distribution, we develop two efficient cleaning policies. The first policy aims at minimizing the cleaning time while reducing the dirt in the entire environment below a user-defined threshold. The second policy aims at minimizes the maximum dirt in the environment within a given, limited cleaning time.

The approach presented in this chapter allows for modeling and estimating the distribution of dirt as well as its dynamics in a given environment. Furthermore, the estimated dirt distribution can be used to generate efficient cleaning policies for cleaning robots. These policies either result in shorter robot operation times while still restricting the maximum dirt level in the environment, or they can be adjusted by the user to specific preferences regarding cleanliness, cleaning time, and even the importance of different areas in the environment.

6.1. Poisson Processes for Modeling the Distribution of Dirt

To selectively clean an environment, a robot needs to know where the dirt is located. If the robot cannot observe the state of the entire environment, one approach for achieving this, is for the robot to learn how quickly the individual parts of the environment typically get dirty.

Motivated by the well-known occupancy grid map [107], our approach makes use of a regular tessellation of the environment into grid cells. In the occupancy grid mapping approach, it is typically assumed that the environment is static and that therefore the occupancy of a cell does not change over time. This assumption, however, is not reasonable in the context of the dirt distribution since the dirt in a cell grows over time because of polluting events like crumbling or chipping and typically gets reduced to zero through cleaning. To model this behavior, we apply for every cell c a homogeneous Poisson process $N^c(t)$ over time t.

6.1.1. Properties of Homogeneous Poisson Processes

A homogeneous Poisson process $\{N^c(t), t \geq 0\}$ is a continuous-time counting process whose increments are stationary, independent, and Poisson distributed (see Section 2.1 for details about the Poisson distribution). The probability of observing z polluting events during time interval $(s,t]$ is given by

$$P(N^c(t) - N^c(s) = z) = \frac{e^{-\lambda^c(t-s)}(\lambda^c(t-s))^z}{z!}, \tag{6.1}$$

where the parameter λ^c is called intensity. Additionally, at time steps t_v. at which cell c is cleaned, we set its level of dirt to $N^c(t_v) = 0$.

6.1.2. Predicting the Dirt in the Environment

For deciding which cells require cleaning, the cleaning robot needs to be able to make predictions about the amount of dirt in the individual cells of the dirt distribution. For predicting the amount of dirt $N^c(t)$ at time t, we use its expectation. The expected value of the amount of dirt z produced during the interval $(s,t]$ in cell c according to the distribution of Eq. 6.1 is given by

$$\mathbb{E}[N^c(t) - N^c(s)] = \lambda^c(t-s). \tag{6.2}$$

If the latest cleaning operation happened at time s, the dirt level at time $t > s$ can be predicted as

$$\begin{aligned} &\mathbb{E}[N^c(t)] \\ =&\mathbb{E}[N^c(t) - N^c(s)] + \mathbb{E}[N^c(s)] \\ =&\lambda^c(t-s) + 0. \end{aligned} \tag{6.3}$$

6.1.3. Parameter Estimation

Before the robot can use the dirt distribution for efficient cleaning, it needs to estimate its parameters λ^c for every cell c. During operation, a cleaning robot equipped with a dirt detection sensor receives for every grid-cell c a series of dirt readings $z_0^c, \ldots, z_n^c$ at time steps $t_0, \ldots, t_n$. At the moment of sensing, the cleaning unit with the dirt detection sensor coevally cleans the observed cell. Therefore, every reading z_i^c except the first z_0^c is a sample from the Poisson distribution defined in Eq. 6.1. If the robot observes all cells c in unit intervals, i.e., $t_i - t_{i-1} = 1$, the maximum likelihood estimator for λ^c is given by

$$\hat{\lambda}^c_{\text{unit}} = \frac{1}{n} \sum_{i=1}^{n} z_i^c \,. \tag{6.4}$$

In our applications, the observation intervals of the vacuum cleaning robot typically do not have unit length. However, we can still construct a maximum likelihood estimator for λ^c by considering its log-likelihood

$$\begin{aligned} &\log \mathcal{L}(\lambda^c \mid z_1^c, \dots, z_n^c, t_0, \dots, t_n) \\ =& \log \prod_{i=1}^{n} \frac{e^{-\lambda^c (t_i - t_{i-1})} (\lambda^c (t_i - t_{i-1}))^{z_i^c}}{z_i^c!} \\ =& \sum_{i=1}^{n} (z_i^c \log \lambda^c - \lambda^c (t_i - t_{i-1}) + \log \frac{(t_i - t_{i-1})^{z_i^c}}{z_i^c!}) \,. \end{aligned} \tag{6.5}$$

Calculating the derivative of Eq. 6.5 and setting it to zero yields the maximum likelihood estimator

$$\hat{\lambda}^c = \frac{1}{\sum (t_i - t_{i-1})} \sum_{i=1}^{n} z_i^c \tag{6.6}$$

for the non-unit interval case.

In an initial learning phase, the robot cleans the whole environment several times and applies the estimator $\hat{\lambda}^c$. It thus estimates for each cell c of the dirt distribution how quickly it typically gets dirty over time. After that, the learned dirt distribution can be used for efficient cleaning, while its parameters can still be updated according to Eq. 6.6 to refine the estimate. For cells c for which all measurements $z_i^c = 0$, we set λ^c to $\varepsilon > 0$ and thus ensure that all cells are considered in the cleaning procedure.

6.2. Efficient Cleaning Strategies using the Distribution of Dirt

In this section, we show how the learned dirt distribution can be utilized for efficient cleaning. We propose two policies. The first policy aims at minimizing the cleaning time while guaranteeing that after cleaning, the dirtiest cell is less dirty than a user defined threshold. The second one allows a user to define the maximum duration of a cleaning run and aims at minimizing the maximum dirt value during this time. A cleaning policy takes into account the predicted state of the dirt distribution and the user preferences. We define the predicted state of the dirt distribution $\mathbf{d}_t$ at time t as

$$\mathbf{d}(t) := (\mathbb{E}[N^{c_1}(t)], \ldots, \mathbb{E}[N^{c_M}(t)]), \tag{6.7}$$

where M is the maximum number of cells. For modeling user preferences, we include a weight w_c for every cell c, describing how important the cleanliness of this cell is for the user. The default weight is 100%, meaning that the amount of dirt in the cell is considered as predicted. The weight can be increased in areas important for the user and decreased in unimportant areas. Utilizing these definitions, we define a cleaning policy π_t as the set of cells that the robot has to clean during the cleaning run at time t, that is

$$\pi_t(\mathbf{d}(t), w_{c_1}, \ldots, w_{c_M}) := \{c_{i_1}, \ldots, c_{i_n}\}_{(\pi_t)}. \tag{6.8}$$

To execute a policy π_t, we compute a path from the parking position of the robot through all cells comprised in the policy and back to the parking position. We frame this problem as a metric traveling salesman problem (TSP) with the selected cells $\{c_{i_1}, \ldots, c_{i_n}\}_{(\pi_t)}$ as vertices in a fully connected graph. We then apply a state-of-the-art TSP solver [61] to this graph to find the shortest collision-free path through all cells $\{c_{i_1}, \ldots, c_{i_n}\}_{(\pi_t)}$. As edge costs, we apply the Euclidean distances of the shortest collision-free paths between the vertices, thereby ignoring the rotational cost of the robot. These rotational cost could be inserted into the calculation by extending the TSP to a generalized TSP in which a cluster is comprised of several orientations in the same position. This approach was presented as *Curvature-GTSP* in the context of covering 3D surfaces with a manipulation robot in Section 4.2. Assuming that the robot executes the planned path with constant velocity, we can calculate the duration $\tau(\pi_t)$ of the policy execution.

6.2.1. Bounding the Maximum Dirt while Minimizing the Cleaning Time

The first cleaning policy aims at reducing the weighted maximum dirt value in all cells below a user defined threshold $d_{\max}$ with confidence $1 - \delta$, while minimizing the execution time. More formally, at the beginning t of each cleaning cycle, we select the *bounded dirt policy*

$$\pi_t(d_{\max}) = \operatorname*{arg\,min}_{\pi; P_\pi(d_{\max}) < \delta} (\tau(\pi)), \tag{6.9}$$

where

$$P_\pi(d_{\max}) = P(\max_c [w_c N^c(t + \tau(\pi))] > d_{\max} \mid \pi_t = \pi) \tag{6.10}$$

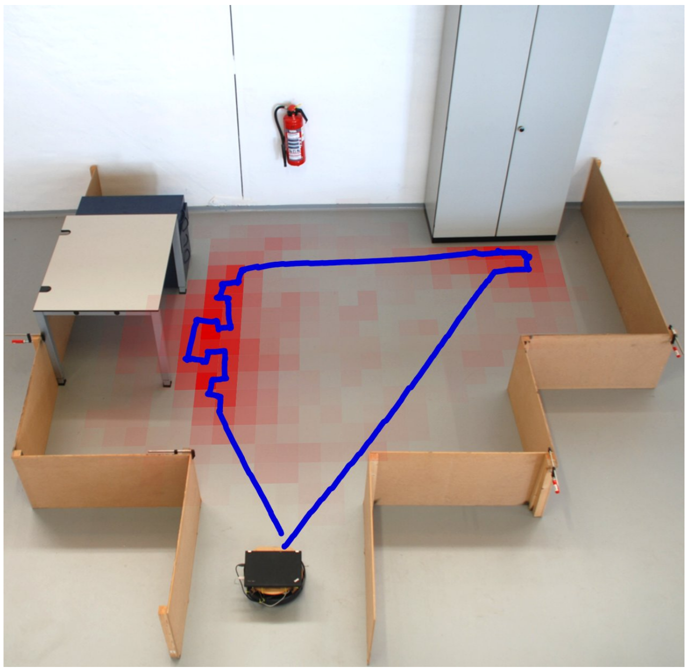


Figure 6.2.: Experimental setup and learned distribution of dirt, which is shown in red. The red color corresponds to the λ_c value, and thus to the estimated frequency of polluting events occurring in each cell. A darker red color means a higher λ_c value. The maximum $\hat{\lambda}_c$ value estimated in the experiment is 10.5. The grid size of the cells used for representing the dirt distribution is 0.15 m. The estimated trajectory of the robot when executing one of the planned paths is shown in blue.

is the probability that after executing policy π at time t the dirtiest cell has a weighted dirt value higher than $d_{\max}$. We can calculate $P_\pi(d_{\max})$ from the estimated λ^c values, the predicted dirt distribution $\mathbf{d}(t)$, and the policy π. To do so, we set $N^c(t) = 0$ for all cells c in π and calculate the $(1-\delta)$-quantiles of the dirt distributions of all cells according to their learned λ^c values. The confidence level $1-\delta$ holds true assuming that the estimated dirt distribution is correct. To calculate the policy, we select all cells whose maximum value of the $(1-\delta)$-quantiles exceeds $d_{\max}$ and calculate a TSP path through these cells as described in the previous section.

6.2.2. Bounding the Cleaning Time while Minimizing the Maximum Dirt

For the second cleaning policy, the user specifies a maximum allowed execution time $\tau_{\max}$ for every cleaning cycle. Given this maximum execution time, we aim at finding the policy that minimizes the weighted maximum dirt value in the cells of the dirt distribution after cleaning. More formally, at the beginning t of each cleaning cycle, we select the *bounded time policy*

$$\pi_t(\tau_{\max}) = \underset{\pi;\tau(\pi)<\tau_{\max}}{\arg\min} \left(\max_c \mathbb{E}[w_c N^c(t+\tau(\pi)) \mid \pi_t = \pi] \right) . \tag{6.11}$$

To calculate this policy, we do not need to consider quantiles, as the cell with the highest expected dirt value is also the cell with the highest $(1-\delta)$-quantile for every $\delta < 0.5$. For this policy, in contrast to the first one, the TSP solver has to be applied iteratively. In every iteration, we add the remaining cell with the highest expected dirt value to the policy and apply the TSP solver. We repeat this process until the maximum allowed execution time $\tau_{\max}$ is reached.

For both cleaning policies the set of cells selected for cleaning changes in consecutive runs of the same policy. The reason for this is that the dirt in all cells grows over time between runs and the dirt value of the cleaned cells gets reset. However, over time all cells get cleaned. If a cell with low dirt production value λ^c does not get selected for cleaning several times, its absolute dirt value $N^c(t)$ keeps growing until its expected dirt value is high enough such that the cleaning policy incorporates this cell into the next cleaning cycle.

6.3. Evaluation

To evaluate the estimation of the distribution of dirt as well as the efficiency of the cleaning policies, we performed extensive experiments both in simulation and with a real robot. In all experiments, we set the confidence level $1-\delta$ for the bounded dirt policy to 95%.

6.3.1. Learning the Distribution of Dirt from Real Data

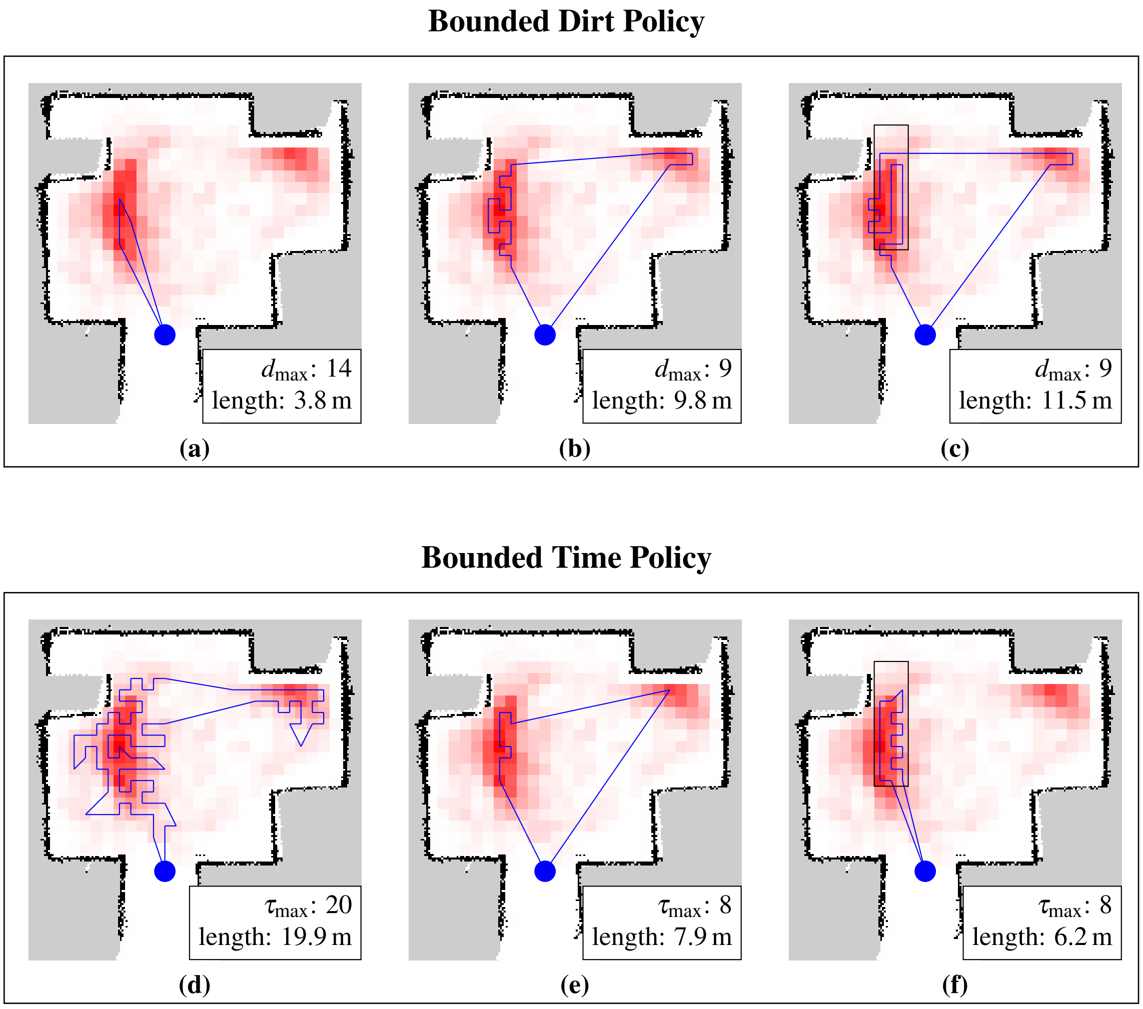
To analyze the learning of the dirt distribution in practice, we equipped an iRobot Roomba 560 vacuum cleaning robot with an Asus Xtion Pro Live Sensor and a notebook with a 2.26 GHz Intel Core2 Duo processor (see Figure 6.1). We found the Roomba robot particularly suited for our problem because it is equipped with a dirt sensor that generates a measurement whenever a dirt particle hits a small metal plate inside the suction unit. In the experiment, we used large-grained flour as dirt. Every dirt measurement corresponded to approximately 0.03 g of flour.

Figure 6.2 shows the environment in which we performed the robot experiments. In order to learn the distribution of dirt, we first built a grid map of the environment. Afterwards, we repeatedly distributed the dirt, mostly in front of the desk and cabinet, and manually steered the robot to clean the entire environment. We repeated this ten times. Thereby we used Monte Carlo localization [145] for robot pose estimation (see Section 2.2.2). In Monte Carlo localization, at every time step t the estimated probability distribution of the pose of the robot is stored as a set of weighted particles $\{\mathbf{x}_t^{[1]}, \ldots, \mathbf{x}_t^{[K]}\}$. As a result, each particle represents a hypothesis about the pose of the robot. After each entire traversal of the environment, we updated the dirt distribution of the environment according to Eq. 6.6 using the dirt measurements received. To account for the localization uncertainty, when we received a dirt measurement z, we considered the particle set only after the resampling step in the particle filter when all particles have the same weight. We divided the dirt measurement evenly over all K particles and integrated, for every particle, the resulting value $\frac{z}{K}$ as dirt measurement for the cell in which the particle was located. This results in the maximum-likelihood dirt distribution given the localization uncertainty. With increasing uncertainty in the localization, the dirt distribution tends towards a uniform distribution, and with a more accurate localization, the dirt distribution becomes more peaked. Thus, the learned dirt distributions depend on the sensor and actuator noise of the robot used. Figure 6.2 shows an example of a dirt distribution learned with the vacuum cleaning robot described above.

6.3.2. Cleaning Policies with User Preferences

Having learned the dirt distribution, we created cleaning paths according to the bounded dirt and the bounded time policies. Figure 6.3 shows two results for each policy with different values for the maximum dirt $d_{\max}$ and the maximum time $\tau_{\max}$ allowed by the user. The cleaning paths were planned for the first cleaning cycle after learning the dirt distribution. For additional cleaning cycles, the paths change as the previously cleaned cells are reset and other cells not cleaned at the first time may be selected.

The time for computing the policies ranged from about one second for the path from Figure 6.3a to about four minutes for the path from Figure 6.3d. The large difference in computation time is due to the fact that the bounded dirt policy (see Figure 6.3a) needs to query the TSP solver only once, while the bounded time policy (see Figure 6.3d) applies the TSP solver iteratively. The computation time, of course, depends on the number of dirty cells in the environment as well as the TSP solver used. As in Chapter 4, we use the LKH solver [61] for querying for a solution and initialize it using the Christofides heuristic (see Section 2.4), which achieves costs of at most 1.5 times the cost of the optimum solution [19]. The LKH solver runs in approximately $O(n^{2.2})$, but a faster one could be applied if less accuracy were sufficient. As can be seen in the figure, if the user specifies a low maximum dirt value, more cells are cleaned by the robot. The same holds



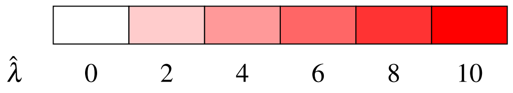


Figure 6.3.: Paths planned with our policies for the first cleaning cycle. The distribution of dirt was learned from real data. The paths in (a) and (b) result from the bounded dirt policy $\pi(d_{\max})$, the ones in (d) and (e) from the bounded time policy $\pi(\tau_{\max})$. The cleaning paths in (c) and (f) for the dirt and bounded time policy result from increasing the weights w_c of the cells in the rectangle to 300%. To allow a comparison with the path length, the unit of the τ values stated in the figure is scaled such that the robot traverses one meter per time step. In (a) and (b), the estimated values $\hat{\lambda}$ for which the dirt stays below a level of 9 and 14 with 95% confidence correspond to 5 and 9, respectively.

for the maximum time allowed. For a higher maximum time value, more cleaning steps are performed and thus the environment is cleaned more thoroughly.

We also made the robot execute the cleaning path from Figure 6.3b. The executed path estimated by the localization system of the robot is shown in blue in Figure 6.2. The path execution was performed by an off-the-shelf navigation system. Evaluating the accuracy of this system and adapting the path execution accordingly could improve the cleaning performance and is addressed in Chapter 7.

Figure 6.3c and Figure 6.3f show the results of increasing the weights of some cells of the dirt distribution. The rectangle in the figure, located in the working area in front of the table, specifies an area where the user increased the weight of the cells to 300%. Figure 6.3c shows the changed cleaning path of the corresponding bounded dirt policy. Due to the increased weights, more cells in the specified area meet the dirt threshold and are thus visited more often than in Figure 6.3b. For the bounded time policy (Figure 6.3f), the cleaning path is changed such that more cells are visited in the specified area. The restricted time, however, does not allow the robot to pass over all previously covered dirty cells, e.g., those in front of the cabinet.

These experiments show that the dirt distribution estimation performs well and that the policies yield reasonable paths. Additionally, they show that the user adapted weights naturally integrate with the proposed policies.

However, the experiment results also raise questions for practical application, e.g., how long does it take until the estimation of the dirt distribution has converged? How efficient are the proposed policies compared to a full coverage of the environment and how does the structure of the environment influence this efficiency? To answer these questions, we performed a number of simulation experiments.

6.3.3. Map Parameter Estimation

To efficiently clean the environment, the robot needs a converged estimate of the parameter λ^c for each cell in the dirt distribution. This raises the question of how many times the entire area has to be covered until the learning of the dirt distribution has converged and the difference is acceptable between the estimated $\hat{\lambda}^c$ for each cell and the real λ^c. For this experiment, we simulated the cleaning of a single cell. We repeatedly sampled polluting events from Poisson distributions given different values of λ, cleaned the cell and updated the estimate $\hat{\lambda}$. Figure 6.4 visualizes the result for λ values of 3, 10, and 20. For all values, the difference between the real and the estimated value decreases significantly after only a few update steps. As an example, a difference of one between the estimated and the real values is noted as a black line in the figure. In the experiment with the real robot, a difference of one corresponds to a difference of 0.03 g of dirt generation per cycle. As one

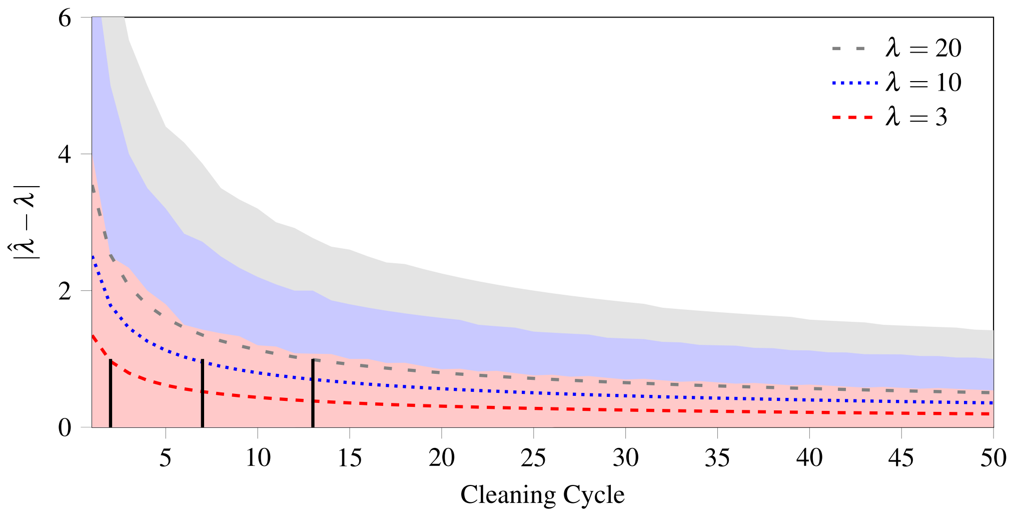


Figure 6.4.: Difference between the estimated dirt value $\hat{\lambda}$ of a cell and the ground truth λ over a number of consecutive cleaning cycles. This figure shows the average deviations as well as the corresponding empirical 95% confidence intervals. The black vertical lines denote the run after which the required difference between the estimated and the ground truth λ value is reached. As one can see, it only requires 14 runs to learn a value of $\lambda = 20$ with the required precision.

can see, the number of cycles needed for reaching this value increases with the value to be estimated. However, even for a relatively large value of 20, only 14 steps are needed until the required distance of the estimated to the real λ is reached.

6.3.4. Evaluation of the Cleaning Policies

To evaluate the cleaning policies and their dependency on the structure of the environment, we simulated different environments with approximately 100 cells. Their size ranges from an environment with 1×100 cells to a quadratic environment with 10×10 cells. We assume that the robot always returns to its start position after it finishes cleaning. Therefore, the 1×100 environment, which corresponds to a long narrow corridor, is the worst case for our approach. In this environment, a single dirty cell can make it necessary to traverse twice the number of cells in the environment. We repeated the experiment $1{,}000$ times for each environment and different maximum dirt levels $d_{\max}$ as well as maximum durations $\tau_{\max}$. For each episode, we sampled a new dirt distribution, i.e., independently for each cell, we sampled a new ground-truth lambda value from an exponential distribution with the mean $\mu_e = 3$. We applied the exponential distribution for sampling due to its similarity

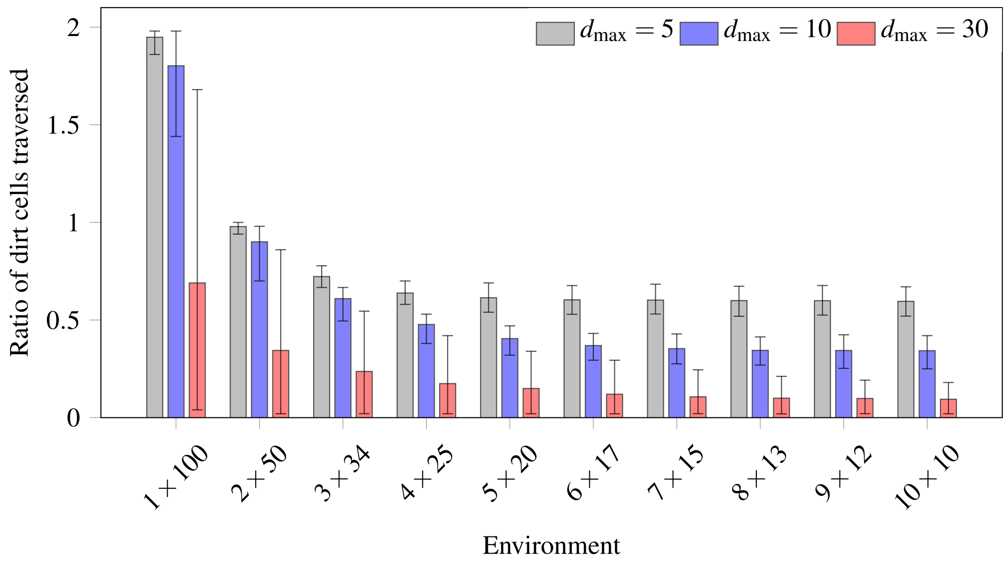


Figure 6.5.: Evaluation of the bounded dirt policy. The figure shows the average ratio of cells traversed to the total number of cells given different values for $d_{\max}$ and differently structured environments. Also shown are the empirical 95% confidence intervals.

to distributions of real environments. In real environments, it is common that a few parts are very dirty more often, e.g., around the breakfast table, while most other areas do not get dirty as regularly. The possible spatial dependency of dirt values in the environment was not considered in the simulation. This is the worst case for our cleaning policies, as spatially correlated dirty cells lead to shorter cleaning paths. Figure 6.5 shows the results for the bounded dirt policy $\pi(d_{\max})$. It plots the number of cells needed to be traversed by our cleaning policy for the different types of environments. This includes cells that were not part of the set selected for cleaning but were only traversed to reach selected cells. Also included are cells covered on the way to and from the start position. That means that cleaning the entire 1×100 cell environment would yield a traversal cost of 200 cells while cleaning the other environments requires traversing every cell in the environment, but at most the number of cells in the environment plus two (for the 7×15 environment). One can see in the figure that for all values of $d_{\max}$, the number of traversed cells decreases as the environment becomes less stretched. A t-test showed that already starting with the 3×34 scenario, our policy traverses significantly fewer cells on a 5% level than the full coverage path, which requires visiting all cells.

To evaluate the bounded time policy $\pi(\tau_{\max})$, we consider the maximum dirt value remaining in all cells after cleaning for a given period of time. For this experiment, we sampled λ^c values and actual dirt values from the Poisson distributions given λ^c for each

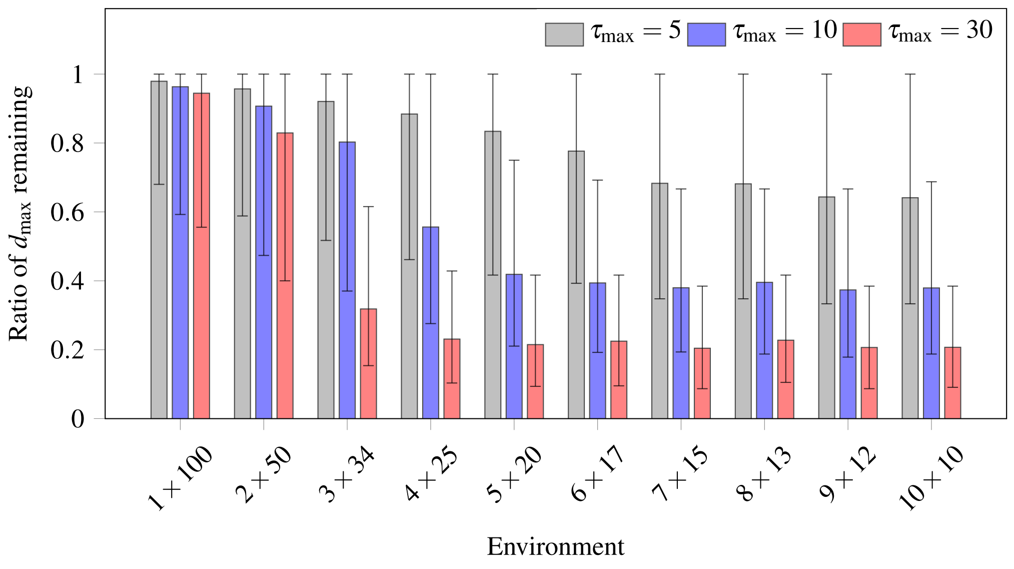


Figure 6.6.: Evaluation of the bounded time policy. The figure shows the average maximum dirt value that remains after cleaning for different values of $d_{\max}$ and differently structured environments. Also shown are the empirical 95% confidence intervals.

cell. We then calculated a cleaning path given the dirt expectation. Figure 6.6 shows the ratio between the maximum dirt value in the environment after cleaning and the maximum dirt value before cleaning. As expected, the more time the robot is allowed to spend for cleaning, the lower the maximum dirt value after its run. More importantly, the efficiency of the bounded time policy does depend on the environment. However, the bounded time policy effectively reduces the maximum dirt value even in demandingly-structured environments, such as the 1×100 cell environment that corresponds to a long corridor.

These experiments show that the efficiency of our cleaning policies depends on the structure of the environment. Nevertheless, even in cases that are far from beneficial for our approach, such as the 3×34 environment, a significant reduction in cleaning time and maximal remaining dirt can be achieved.

6.4. Related Work

The task of cleaning a known surface is often framed as a *coverage path planning* problem, which refers to passing each free location in the environment once, while minimizing some cost measure. In Section 4.5 we provide an overview of these techniques. In contrast to these complete coverage approaches, we aim at covering only a part of the environment while either minimizing the time needed to reach a certain level of cleanliness or minimizing the weighted maximum dirtiness after cleaning. Only without any prior information about the dirt distribution, the problem considered in this chapter corresponds to the complete coverage problem.

Covering only specific parts of an environment has also been addressed in multi-robot navigation [89, 159]. Zlot et al. [159], for example, generate possible exploration goals and, similarly to our approach, frame the problem of visiting all goals as a traveling salesman problem (TSP).

Cleaning applications that aim at covering the entire environment or covering specific parts of it require that the robot can accurately map the environment and localize itself. In the area of floor cleaning, Gutmann et al. [59] propose an algorithm for simultaneously estimating a vector field induced by stationary signal sources in the environment as well as the robot pose in this vector field. Jeong and Lee [74] use a single camera pointing towards the ceiling and apply a landmark-based SLAM algorithm. The approaches of Zhang et al. [157] and Erickson et al. [40] show that SLAM can also be successfully applied using very limited-range sensors like bumpers. In our work we use a low-cost range sensor for mapping and localization.

In the areas of lifelong localization and mapping of changing environments, there are several approaches that also explicitly model dynamics in the environment. Tipaldi et al. [148] use a combination of a Rao-Blackwellized particle filter with a hidden Markov model to localize in dynamically changing environments. Thereby they use a dynamic occupancy grid which models the evolution of the state of each cell over time with a hidden Markov model. Krajník et al. [86] propose the so-called frequency mapping for grid-based maps. They explicitly model the dynamics of single grid cells over time by applying Fourier analysis. This enables them to predict the state of the environment, e.g., whether a door is open or closed, on a particular day at a particular time.

The ideas presented in this chapter are most related to the work of Luber et al. [99] in the area of people tracking. The authors propose spatial affordance maps to represent space-dependent occurrences of people-activity events. Similar to our approach, these maps use cell-wise Poisson processes. Luber et al. use these spatial affordance maps to improve people tracking, while Tipaldi and Arras [147] present a model for coverage applications that minimizes the interference with people. An earlier work by Kruse and Wahl [87] proposes a statistical grid, a grid map in which each cell models the probability of the appearance of a dynamic obstacle with a Poisson process. We apply Poisson processes to model and estimate the dynamics of the evolution of dirt in an environment and utilize this representation to generate efficient cleaning policies.

6.5. Conclusions

In this chapter, we presented a novel approach based on Poisson processes for representing, estimating, and predicting the distribution of dirt in an environment. We described how the robot can learn the parameters of these models and presented two efficient cleaning policies for robotic vacuum cleaners using these models. In extensive simulation experiments, we showed that, compared to the state-of-the-art approach, which is full and systematic coverage, our informed approach can lead to significantly shorter cleaning times. We furthermore demonstrated the practical applicability of our approach in experiments with a real robotic vacuum cleaner.

In this chapter, we assumed that the robot's cleaning unit behaves deterministically and thus always cleans all the dirt it passes over. Additionally, although we considered the localization uncertainty for learning the dirt distribution, we did not consider the localization uncertainty in the path execution. In the next chapter, we extend our approach to a fully probabilistic model that explicitly considers the uncertainty in the path execution as well as the uncertainty of the dirt actuator. This model enables the robot to jointly estimate its trajectory and the current distribution of dirt in the environment during the cleaning process. We exploit this knowledge for adapting the cleaning path online, which results in guaranteed low dirt levels in the entire environment with high confidence.

Chapter 7

High-Confidence Cleaning Guarantees for Low-Cost Robots

The goal of robotic cleaning is to achieve low dirt levels in the whole environment. Low-cost consumer robots, however, are typically prone to high motion and sensor noise. Additionally, their cleaning units do not always remove the dirt entirely. As a result, there is a substantial probability that some parts of the environment are insufficiently cleaned. In this chapter, we extend the approach developed in the previous chapter to consider the uncertainty of the actuator unit as well as the uncertainty of localization during task execution. This results in an approach to robotic cleaning that guarantees that, in the whole environment, the dirt levels after cleaning are reduced below a user-defined threshold with high confidence. We introduce a novel probabilistic model for jointly estimating the past trajectory of the robot and the current dirt distribution in the environment. Based on this estimate, we adapt the future cleaning path during operation such that the robot re-visits areas in which high dirt levels are still likely. We demonstrate the effectiveness of our approach in extensive experiments carried out both in simulation and with a real vacuum cleaning robot.

In the previous chapter, we presented an approach that uses Poisson processes for representing, estimating, and predicting the distribution of dirt in an environment. In addition, we developed two efficient cleaning policies for robotic vacuum cleaners using these models. We assumed that the robot possesses a deterministic actuator unit, i.e., every spot is cleaned if the robot has covered it, and did not consider the noise in the path execution.

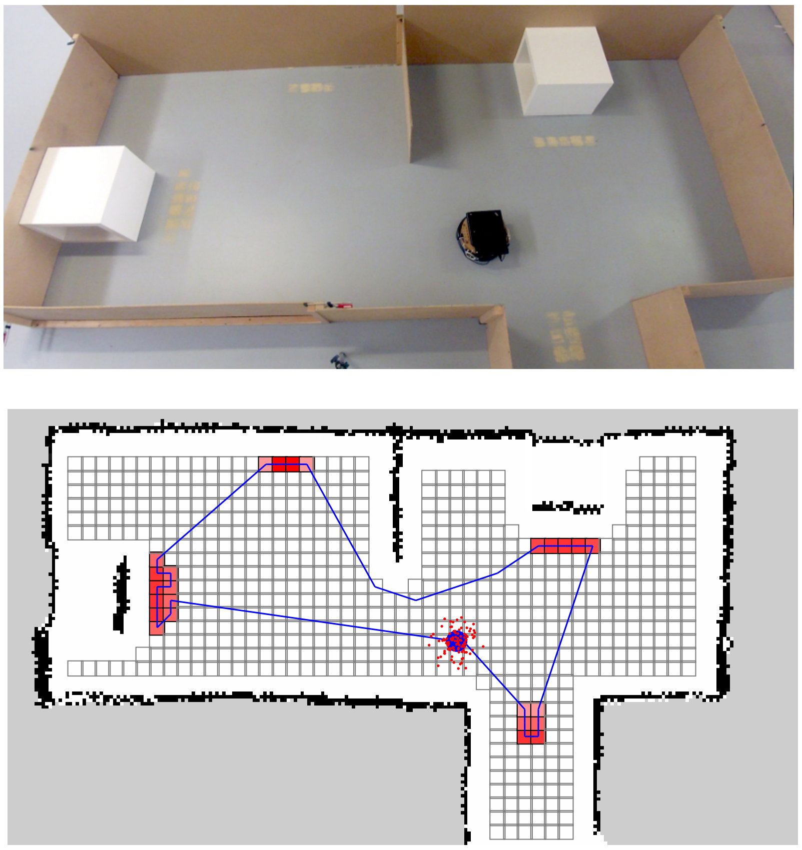


Figure 7.1.: The Roomba robot in the experimental environment (top) and its planned path (blue line) in the map (bottom). The bottom part shows a map of the environment (black obstacles) together with a set of particles (red dots) representing the probability distribution of the pose of the robot. The grid denotes the dirt distribution with dirty (red) and clean cells (white).

However, low-cost consumer robots are particularly subject to high motion and sensor noise. Therefore, they might miss certain areas during the execution of the cleaning path. Additionally, these robots do not always remove the dirt entirely on the first pass. Thus, some parts of the environment may need repeated cleaning. Obviously, if the robot knows which trajectory it took thus far and what the results of its cleaning actions are, it can more easily identify such areas. In this chapter, we propose a probabilistic model for jointly estimating the trajectory of the robot and the current distribution of dirt in the environment. We furthermore show how this model can be used to guarantee low dirt levels. We consider a cleaning robot with a range sensor to perceive its environment and a dirt sensor to measure the amount of dirt currently cleaned. The robot has access to an occupancy map of its environment and a prior estimate of the distribution of dirt in the environment that it obtains as described in Chapter 6.

Based on the prior dirt distribution, we select the parts of the environment that need to be cleaned (see Figure 7.1 for an example) and compute an initial cleaning path. We model the execution of the cleaning path in a probabilistic framework. In addition to the uncertainties of the odometry and the range sensor, we explicitly model the uncertainties of the cleaning unit and the dirt sensor. Using this framework, we jointly estimate the past trajectory of the robot and the current dirt distribution. This estimate enables the robot to identify the parts of the environment that have not yet been sufficiently cleaned. We exploit this knowledge for iteratively adapting the cleaning path during operation.

The approach developed in this chapter guarantees low dirt levels in the whole environment with high confidence. Depending on the actual distribution of dirt, this approach typically needs considerably less time than approaches that always traverse the entire environment. Due to the estimation of the uncertainties in the task execution, our approach even works on low-cost consumer robots with high noise levels. As a side effect, the integration of dirt measurements into the estimation can significantly improve the localization performance of the robot. Extensive experiments, both in simulation and with a real cleaning robot, demonstrate that our approach is also applicable in practice.

7.1. Probabilistic Models for Localization and Cleaning

In the following, we consider a mobile cleaning robot equipped with a cleaning unit (dirt actuator), a sensor inside the cleaning unit that measures the amount of dirt sucked in at the moment (dirt sensor), as well as a sensor for measuring the distances to obstacles in the environment, i.e., a depth camera or a laser scanner. We assume that the robot has a geometric map of its environment (e.g., an occupancy grid map), which it uses to match its distance measurements against. Additionally, the robot has access to an estimate of the dirt distribution in the environment at the beginning of the cleaning cycle. To represent the dirt distribution, we follow our approach as presented in Chapter 6 and use a regular tessellation of the environment into M grid cells. We furthermore assume that, for every grid cell c_i, the amount of dirt in the cell at time t is Poisson distributed with parameter λ_t^i (see Section 2.1 for details about the Poisson distribution). Together, the values λ_t^i for all cells c_i define the dirt distribution $\mathbf{d}_t = (\lambda_t^1, \ldots, \lambda_t^M)$. We also obtain the initial dirt distribution $\mathbf{d}_0$ applying the techniques described in the previous chapter.

We now describe how to jointly estimate the trajectory of the robot and the current dirt distribution from the observations of the robot. Furthermore, we introduce probabilistic models describing the cleaning unit and the dirt sensor of the robot.

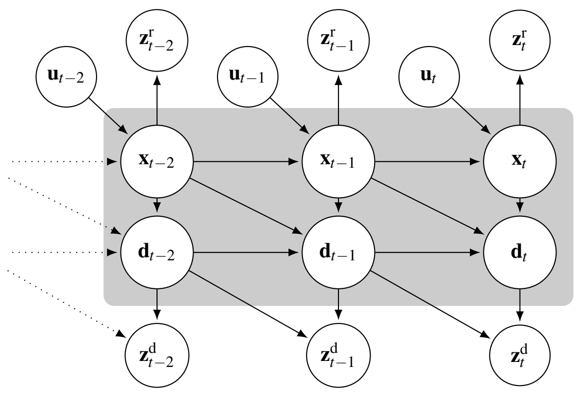

$\mathbf{z}^{\mathrm{r}}_{t-2}$
$\mathbf{z}^{\mathrm{r}}_{t-1}$
$\mathbf{z}^{\mathrm{r}}_{t}$
$\mathbf{u}_{t-2}$
$\mathbf{u}_{t-1}$
$\mathbf{u}_{t}$
$\mathbf{x}_{t-2}$
$\mathbf{x}_{t-1}$
$\mathbf{x}_{t}$
$\mathbf{d}_{t-2}$
$\mathbf{d}_{t-1}$
$\mathbf{d}_{t}$
$\mathbf{z}^{\mathrm{d}}_{t-2}$
$\mathbf{z}^{\mathrm{d}}_{t-1}$
$\mathbf{z}^{\mathrm{d}}_{t}$

Figure 7.2.: Graphical model of the dynamical system describing the evolution of the robot pose $\mathbf{x}_t$ and the dirt distribution $\mathbf{d}_t$. The shaded area marks the hidden variables.

7.1.1. Joint Estimation of the Trajectory and the Dirt Distribution

Our goal is to estimate the joint posterior distribution of the trajectory of the robot and the dirt distribution, i.e.,

$$p(\mathbf{x}_{0:t}, \mathbf{d}_{0:t} \mid \mathbf{u}_{1:t}, \mathbf{z}^{\mathrm{r}}_{1:t}, \mathbf{z}^{\mathrm{d}}_{1:t}), \tag{7.1}$$

where $\mathbf{x}_{0:t}$ and $\mathbf{d}_{0:t}$ are the full histories of the pose of the robot and the dirt distribution starting from the beginning of the current cleaning cycle. Here, $\mathbf{u}_{1:t}$ are the motion controls of the robot, $\mathbf{z}^{\mathrm{r}}_{1:t}$ are the range measurements, and $\mathbf{z}^{\mathrm{d}}_{1:t}$ are the dirt measurements. We assume that while the robot cleans the environment, the values of the dirt distribution are only changed as a result of the the robot's cleaning actions. Thus, the current dirt distribution $\mathbf{d}_t$ depends only on the previous dirt distribution $\mathbf{d}_{t-1}$ and the path taken by the cleaning robot. Similarly, the dirt measurement $\mathbf{z}^{\mathrm{d}}_t$ depends only on the change of the dirt distribution from time $t-1$ to time t. We describe the resulting dynamical system by the graphical model shown in Figure 7.2.

Applying Bayes' rule and the rules of d-separation [9] to the graphical model, we can factor the distribution in Eq. 7.1 similarly to the derivation of the particle filter presented in Section 2.2.2:

$$\begin{aligned} & p(\mathbf{x}_{0:t}, \mathbf{d}_{0:t} \mid \mathbf{u}_{1:t}, \mathbf{z}^{\mathrm{r}}_{1:t}, \mathbf{z}^{\mathrm{d}}_{1:t}) \\ & = \eta \quad \underbrace{p(\mathbf{z}^{\mathrm{r}}_t \mid \mathbf{x}_t)}_{\text{distance sensor model}} \ \underbrace{p(\mathbf{z}^{\mathrm{d}}_t \mid \mathbf{d}_{t-1}, \mathbf{d}_t)}_{\text{dirt sensor model}} \underbrace{p(\mathbf{d}_t \mid \mathbf{d}_{t-1}, \mathbf{x}_{t-1}, \mathbf{x}_t)}_{\text{dirt actuator model}} \\ & \quad \underbrace{p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t)}_{\text{robot motion model}} \underbrace{p(\mathbf{x}_{0:t-1}, \mathbf{d}_{0:t-1} \mid \mathbf{u}_{1:t-1}, \mathbf{z}^{\mathrm{r}}_{1:t-1}, \mathbf{z}^{\mathrm{d}}_{1:t-1})}_{\text{prior distribution}}, \end{aligned} \tag{7.2}$$

where η is a normalization constant. Eq. 7.2 defines a recursive update rule, which allows us to compute the desired distribution at time t from the distribution at time $t-1$, given all sensor and actuator models. For the distance sensor and the robot motion, we employ standard models from the literature [145]. In the following sections, we develop the models for the dirt actuator and the dirt sensor. Before introducing these models, however, we state the derivation of the factored posterior below (see Eq. 7.2). In this derivation, we denote the application of Bayes' rule or the Markov assumption for each step:

$$
\begin{aligned}
&p(\mathbf{x}_{0:t}, \mathbf{d}_{0:t} | \mathbf{u}_{1:t}, \mathbf{z}^{\mathrm{r}}_{1:t}, \mathbf{z}^{\mathrm{d}}_{1:t}) \\
&\overset{\text{Bayes}}{=} \eta\, p(\mathbf{z}^{\mathrm{r}}_t | \mathbf{x}_{0:t}, \mathbf{d}_{0:t}, \mathbf{u}_{1:t}, \mathbf{z}^{\mathrm{r}}_{1:t-1}, \mathbf{z}^{\mathrm{d}}_{1:t})\, p(\mathbf{x}_{0:t}, \mathbf{d}_{0:t} | \mathbf{u}_{1:t}, \mathbf{z}^{\mathrm{r}}_{1:t-1}, \mathbf{z}^{\mathrm{d}}_{1:t}) \\
&\overset{\text{Markov}}{=} \eta\, p(\mathbf{z}^{\mathrm{r}}_t | \mathbf{x}_t)\, p(\mathbf{x}_{0:t}, \mathbf{d}_{0:t} | \mathbf{u}_{1:t}, \mathbf{z}^{\mathrm{r}}_{1:t-1}, \mathbf{z}^{\mathrm{d}}_{1:t}) \\
&\overset{\text{Bayes}}{=} \eta\, p(\mathbf{z}^{\mathrm{r}}_t | \mathbf{x}_t)\, p(\mathbf{z}^{\mathrm{d}}_t | \mathbf{x}_{0:t}, \mathbf{d}_{0:t}, \mathbf{z}^{\mathrm{r}}_{1:t-1}, \mathbf{u}_{1:t}, \mathbf{z}^{\mathrm{d}}_{1:t-1})\, p(\mathbf{x}_{0:t}, \mathbf{d}_{0:t} | \mathbf{u}_{1:t}, \mathbf{z}^{\mathrm{r}}_{1:t-1}, \mathbf{z}^{\mathrm{d}}_{1:t-1}) \\
&\overset{\text{Markov}}{=} \eta\, p(\mathbf{z}^{\mathrm{r}}_t | \mathbf{x}_t)\, p(\mathbf{z}^{\mathrm{d}}_t | \mathbf{d}_t, \mathbf{d}_{t-1})\, p(\mathbf{x}_{0:t}, \mathbf{d}_{0:t} | \mathbf{u}_{1:t}, \mathbf{z}^{\mathrm{r}}_{1:t-1}, \mathbf{z}^{\mathrm{d}}_{1:t-1}) \\
&\overset{\text{Bayes}}{=} \eta\, p(\mathbf{z}^{\mathrm{r}}_t | \mathbf{x}_t)\, p(\mathbf{z}^{\mathrm{d}}_t | \mathbf{d}_t, \mathbf{d}_{t-1})\, p(\mathbf{d}_t | \mathbf{d}_{0:t-1}, \mathbf{x}_{0:t}, \mathbf{z}^{\mathrm{r}}_{1:t-1}, \mathbf{u}_{1:t}, \mathbf{z}^{\mathrm{d}}_{1:t-1}) \\
&\qquad\qquad (\mathbf{x}_{0:t}, \mathbf{d}_{0:t} | \mathbf{u}_{1:t}, \mathbf{z}^{\mathrm{r}}_{1:t-1}, \mathbf{z}^{\mathrm{d}}_{1:t-1}) \\
&\overset{\text{Markov}}{=} \eta\, p(\mathbf{z}^{\mathrm{r}}_t | \mathbf{x}_t)\, p(\mathbf{z}^{\mathrm{d}}_t | \mathbf{d}_t, \mathbf{d}_{t-1})\, p(\mathbf{d}_t | \mathbf{d}_{t-1}, \mathbf{x}_t, \mathbf{x}_{t-1})\, p(\mathbf{x}_{0:t}, \mathbf{d}_{0:t-1} | \mathbf{u}_{1:t}, \mathbf{z}^{\mathrm{r}}_{1:t-1}, \mathbf{z}^{\mathrm{d}}_{1:t-1}) \\
&\overset{\text{Bayes}}{=} \eta\, p(\mathbf{z}^{\mathrm{r}}_t | \mathbf{x}_t)\, p(\mathbf{z}^{\mathrm{d}}_t | \mathbf{d}_t, \mathbf{d}_{t-1})\, p(\mathbf{d}_t | \mathbf{d}_{t-1}, \mathbf{x}_t, \mathbf{x}_{t-1}) \\
&\qquad\qquad p(\mathbf{x}_t | \mathbf{x}_{0:t-1}, \mathbf{d}_{0:t-1}, \mathbf{z}^{\mathrm{r}}_{1:t-1}, \mathbf{u}_{1:t}, \mathbf{z}^{\mathrm{d}}_{1:t-1})\, p(\mathbf{x}_{0:t}, \mathbf{d}_{0:t-1} | \mathbf{u}_{1:t}, \mathbf{z}^{\mathrm{r}}_{1:t-1}, \mathbf{z}^{\mathrm{d}}_{1:t-1}) \\
&\overset{\text{Markov}}{=} \eta\, p(\mathbf{z}^{\mathrm{r}}_t \mid \mathbf{x}_t)\, p(\mathbf{z}^{\mathrm{d}}_t \mid \mathbf{d}_{t-1}, \mathbf{d}_t)\, p(\mathbf{d}_t \mid \mathbf{d}_{t-1}, \mathbf{x}_{t-1}, \mathbf{x}_t)\, p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t) \\
&\qquad\qquad p(\mathbf{x}_{0:t-1}, \mathbf{d}_{0:t-1} \mid \mathbf{u}_{1:t-1}, \mathbf{z}^{\mathrm{r}}_{1:t-1}, \mathbf{z}^{\mathrm{d}}_{1:t-1}).
\end{aligned}
\tag{7.3}
$$

For simplifying the notation, we let the normalizer η represent different quantities during this derivation. The normalizer changes with every application of Bayes' rule.

7.1.2. Dirt Actuator Model

The dirt actuator model describes how the values in the dirt distribution change from one time step to the next, depending on the area covered by the cleaning unit, i.e., the path traveled by the robot.

Every speck of dirt that the cleaning unit passes over is either sucked in or left on the floor. Each of these events can be modeled by a Bernoulli distribution with a cleaning unit-specific probability p_{clean} for cleaning the speck of dirt. To determine how many

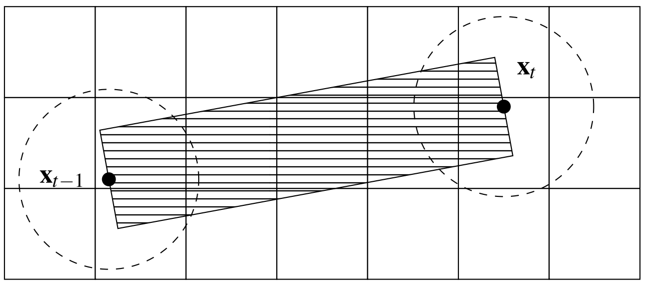


Figure 7.3.: Visualization of the area covered by the cleaning unit when the robot moves from pose $\mathbf{x}_{t-1}$ to pose $\mathbf{x}_t$. The dotted circles mark the diameter of the robot. The width of the covered rectangle depends on the width of the cleaning unit. We use unweighted area sampling to estimate the covered percentage of each grid cell.

such events occur in each cell of the dirt distribution, we need to consider the amount of dirt in each cell before cleaning. In addition, as the robot may not traverse a cell entirely, we need to consider the area of the cell covered by the cleaning unit. We calculate the fraction $\mathcal{F}_i(\mathbf{x}_{t-1}, \mathbf{x}_t)$ of each cell c_i that the cleaning unit passed over when traveling from pose $\mathbf{x}_{t-1}$ to pose $\mathbf{x}_t$ by applying unweighted area sampling [46] (see Figure 7.3 for an illustration). We assume that the dirt is distributed uniformly inside each single cell. Under this assumption, multiplying $\mathcal{F}_i(\mathbf{x}_{t-1}, \mathbf{x}_t)$ with the amount of dirt $\mathbf{d}_{i,t-1}$ that was in the cell at time $t-1$ yields the number of Bernoulli events in this cell. A Bernoulli event in this case corresponds to the event of either cleaning or not cleaning a speck of dirt that is passed by the cleaning unit. As a series of independent Bernoulli events follows a Binomial distribution (see Section 2.1), the reduction of dirt $(\mathbf{d}_{i,t-1} - \mathbf{d}_{i,t})$ in cell c_i is distributed according to $\text{Binom}(\mathbf{d}_{i,t-1} - \mathbf{d}_{i,t}; \mathcal{F}_i(\mathbf{x}_{t-1}, \mathbf{x}_t)\mathbf{d}_{i,t-1}, p_{\text{clean}})$, where $\text{Binom}(k; n, p)$ is the probability of drawing sample k from a Binomial distribution with parameters n and p. Assuming that the amounts of dirt cleaned in different cells are independent, this leads to the dirt actuator model

$$\begin{aligned} & p(\mathbf{d}_t \mid \mathbf{d}_{t-1}, \mathbf{x}_{t-1}, \mathbf{x}_t) \\ &= \prod_{i=1}^{M} \text{Binom}(\mathbf{d}_{i,t-1} - \mathbf{d}_{i,t}; \mathcal{F}_i(\mathbf{x}_{t-1}, \mathbf{x}_t)\mathbf{d}_{i,t-1}, p_{\text{clean}}). \end{aligned} \tag{7.4}$$

7.1.3. Dirt Sensor Model

The dirt sensor model in Eq. 7.2 is a probabilistic model describing the measurement $\mathbf{z}_t^{\text{d}}$ received if the dirt distribution changes from $\mathbf{d}_{t-1}$ to $\mathbf{d}_t$. The dirt sensor included in our robot, the iRobot Roomba, is a small metal plate inside the cleaning unit that sends a signal when it gets hit by a speck of dirt during cleaning. As dirt measurement $\mathbf{z}_t^{\mathrm{d}}$, we use the number of these signals received between the time steps $t-1$ and t. The rate with which the metal plate gets hit in this time interval depends on the amount of dirt sucked in. Let $\Delta\mathbf{d}$ denote change in the dirt distribution in the last time step, i.e.,

$$\Delta\mathbf{d} = \sum_{i=1}^{M} (\mathbf{d}_{i,t-1} - \mathbf{d}_{i,t}). \tag{7.5}$$

Assuming that the individual hitting events occur independently of each other, and that the average hitting rate given an amount of dirt cleaned is constant, this process can be described by a Poisson distribution with intensity parameter $\lambda = \Delta\mathbf{d}$. Therefore, we define the dirt sensor model as

$$p(\mathbf{z}_t^{\mathrm{d}} \mid \mathbf{d}_{t-1}, \mathbf{d}_t) \sim \mathrm{Poiss}(\Delta\mathbf{d}). \tag{7.6}$$

7.2. A Guarantee for Cleaning with High Confidence

The goal of our robotic cleaning approach is to guarantee that the maximum dirt value in the environment is reduced below the user-defined threshold $d_{\max}$ with confidence $1-\delta$. In this section, we describe our implementation of the recursive update rule in Eq. 7.2. Furthermore, we show how we use the resulting estimated dirt values to adapt the future cleaning path to achieve the desired cleaning guarantee.

7.2.1. Particle Filter Implementation

To implement the recursive estimation scheme, we use a particle filter [145] (see Section 2.2.2). The particle filter represents the probability distribution in Eq. 7.2 as a set of k sample hypotheses, called particles. The particle set has the form

$$\mathcal{Y}_t = \left\{ \langle (\mathbf{x}_{0:t}^{[k]}, \mathbf{d}_{1,0:t}^{[k]}, \ldots, \mathbf{d}_{M,0:t}^{[k]}), w_t^{[k]} \rangle \mid k \in 1, \ldots, K \right\}, \tag{7.7}$$

where $\mathbf{x}_{0:t}^{[k]}$ is the hypothesis of the k-th particle about the history of the robot pose, $\mathbf{d}_{i,0:t}^{[k]}$ is its hypothesis about the history of dirt values of cell c_i, and $w_t^{[k]}$ is its weight. At each time step, we create the updated hypothesis of each particle by drawing a random sample first from the robot motion model, and then, given its outcome, from the dirt actuator model.

We then assign to all K particles weights $w^{[k]}$ that describe the likelihood of the sensor readings given the sampled states. More formally, the particle weights are given by the posterior distribution divided by the proposal distribution, i.e.,

$$w_t^{[k]} = \frac{p(\mathbf{x}_{0:t}^{[k]}, \mathbf{d}_{0:t}^{[k]} | \mathbf{u}_{1:t}, \mathbf{z}_{1:t}^{\mathrm{r}}, \mathbf{z}_{1:t}^{\mathrm{d}})}{q(\mathbf{x}_{0:t}^{[k]}, \mathbf{d}_{0:t}^{[k]} | \mathbf{u}_{1:t}, \mathbf{z}_{1:t}^{\mathrm{r}}, \mathbf{z}_{1:t}^{\mathrm{d}})}. \tag{7.8}$$

As proposal distribution we choose the product of the robot motion model and the recursive term just as for the standard particle filter for robot localization. In addition, we include the dirt actuator model. We thus define the proposal distribution as

$$\begin{aligned} q(\mathbf{x}_{0:t}^{[k]}, \mathbf{d}_{0:t}^{[k]} | \mathbf{u}_{1:t}, \mathbf{z}_{1:t}^{\mathrm{r}}, \mathbf{z}_{1:t}^{\mathrm{d}}) = {} & p(\mathbf{d}_t \mid \mathbf{d}_{t-1}, \mathbf{x}_{t-1}, \mathbf{x}_t)\, p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t) \\ & p(\mathbf{x}_{0:t-1}, \mathbf{d}_{0:t-1} \mid \mathbf{u}_{1:t-1}, \mathbf{z}_{1:t-1}^{\mathrm{r}}, \mathbf{z}_{1:t-1}^{\mathrm{d}}). \end{aligned} \tag{7.9}$$

Substituting the posterior (Eq. 7.2) and proposal distribution (Eq. 7.9) into Eq. 7.8 and proceeding similarly to the derivation of the particle filter in (see Eq. 2.27 on page 17) yields the following weight for each particle k:

$$w_t^{[k]} = \eta\, p(\mathbf{z}_t^{\mathrm{r}} \mid \mathbf{x}_t^{[k]})\, p(\mathbf{z}_t^{\mathrm{d}} \mid \mathbf{d}_{t-1}^{[k]}, \mathbf{d}_t^{[k]}). \tag{7.10}$$

We thus compute the weight as the product of the densities of the dirt sensor model and the distance sensor model.

Finally, in a *resampling step*, we create a new set of particles from the actual set by drawing K times with replacement from the probability distribution induced by the weights $w^{[k]}$. Compared to the standard particle filter for robot localization as described in Section 2.2.2, the extended particle state defined in Eq. 7.7 has an increased dimensionality and thus an increased computational complexity since it requires more particles. However, as changes in the dirt distribution happen only in parts in which particles are at the moment, we can reduce the computational complexity by sampling dirt values only from those parts. In this efficient approximation, we initialize the dirt distribution of all cells c_i in each particle k with the mean λ_0^i of the initial dirt distribution $\mathbf{d}_{0,i}$. Only if the robot pose hypothesis of a particle enters a cell c_i at some time step s, we sample a concrete instantiation of the dirt value of this cell from the Poisson distribution with parameter λ_{s-1}^i. This value is updated according to the dirt actuator model resulting in the sample $\mathbf{d}_{s,i}^{[k]}$. When, at some time step $t-1$, the pose hypotheses of all particles are outside of c_i again, we estimate the new mean λ_t^i of the Poisson distribution of c_i and set the values $\mathbf{d}_{t,i}^{[k]}$

of all particles to λ_t^i. As an estimator, we use the weighted mean of the dirt values in all particles:

$$\lambda_t^i = \sum_{k=1}^{K} w^{[k]} \mathbf{d}_{t-1,i}^{[k]}. \tag{7.11}$$

Although some values $\mathbf{d}_{t-1,i}^{[k]}$ are not samples but contain the previously estimated mean, this is an unbiased estimator. Due to this efficient implementation, the computational complexity of our approach does not depend on the size of the environment, but only on the size of the particle cloud. In the experiments we show that our approach runs online and is effective with a reasonable number of particles.

7.2.2. Localization-Based Replanning of the Cleaning Path

To achieve the desired cleaning guarantee, the robot starts cleaning by following the *bounded dirt* cleaning path introduced in the previous chapter (see Section 6.2.1). This path is created from the policy $\pi(\mathbf{d}_0) = \{c_{i_1}, \ldots, c_{i_n}\}$ that consists of all cells in which the $(1-\delta)$ quantile of the dirt distribution is above $d_{\max}$. Given $\pi(\mathbf{d}_0)$, we generate the path by solving the traveling salesman problem (TSP) on the fully connected graph with all cells in $\pi(\mathbf{d}_0)$ using the actual robot positions as vertices. We assign the Euclidean distances of the shortest collision-free paths between the cells as edge costs.

The contribution of the revised approach presented in this chapter is that during cleaning, we re-compute the cleaning path to re-visit previously imperfectly cleaned cells based on the current dirt distribution. Concretely, at fixed intervals during operation, we consider the parameters λ_t^i of all cells c_i of the dirt distribution that are currently empty of particles. The currently traversed cells are not considered as their cleaning is in progress and their state might be changing. From the values λ_t^i, we calculate the $(1-\delta)$ quantiles of the dirt distributions in every cell. From the quantiles, we compute the cells that have to be cleaned, i.e., the policy $\pi_t(\hat{\mathbf{d}}_t)$ at time t, and compare it with the policy $\pi_{t-1}(\hat{\mathbf{d}}_{t-1})$ that is currently being followed. If the new policy $\pi_t(\hat{\mathbf{d}}_t)$ includes cells that the policy $\pi_{t-1}(\hat{\mathbf{d}}_{t-1})$ would not visit anymore, we re-compute the TSP path using the previous path as an initial guess. Generally only a few cells are added to the path, in which case the runtime of the TSP solver is often reduced substantially.

The robot ends the cleaning cycle if, after a final re-planning step, the policy contains no cells, i.e., the $(1-\delta)$ quantiles of the dirt distributions in every cell are below $d_{\max}$. Thus, after cleaning, our approach can guarantee that the dirt levels in the entire environment are below $d_{\max}$ with confidence $1-\delta$.

7.3. Evaluation

We evaluated the performance of our approach in extensive experiments both in simulation and with a real cleaning robot. In all experiments, we used a confidence level of $1-\delta = 99\%$, a maximum allowed dirt level of $d_{\max} = 5$ and set the number of particles to $K = 500$. To compute cleaning paths, we applied the same state-of-the-art TSP solver [61] used in the previous chapter.

7.3.1. Evaluation on a Real Cleaning Robot

To evaluate the effectiveness of our approach in practice, we used an iRobot Roomba 560 vacuuming cleaning robot, which is by default equipped with a dirt sensor. This dirt sensor is a small metal plate inside the suction unit that generates a measurement whenever a speck of dirt hits the plate. In the experiment, we used large-grained flour as dirt. A calibration of the sensor yielded that every measurement (and therefore also every unit of λ_t^i and of $d_{\max}$) corresponds to approximately 0.16 g of flour. As in the previous chapter, we additionally equipped the robot with an Asus Xtion Pro Live sensor that we used for distance measurements, with a gyroscope to improve the odometry readings of the robot, and with a notebook with an Intel Core 2 Duo 1.60 GHz CPU.

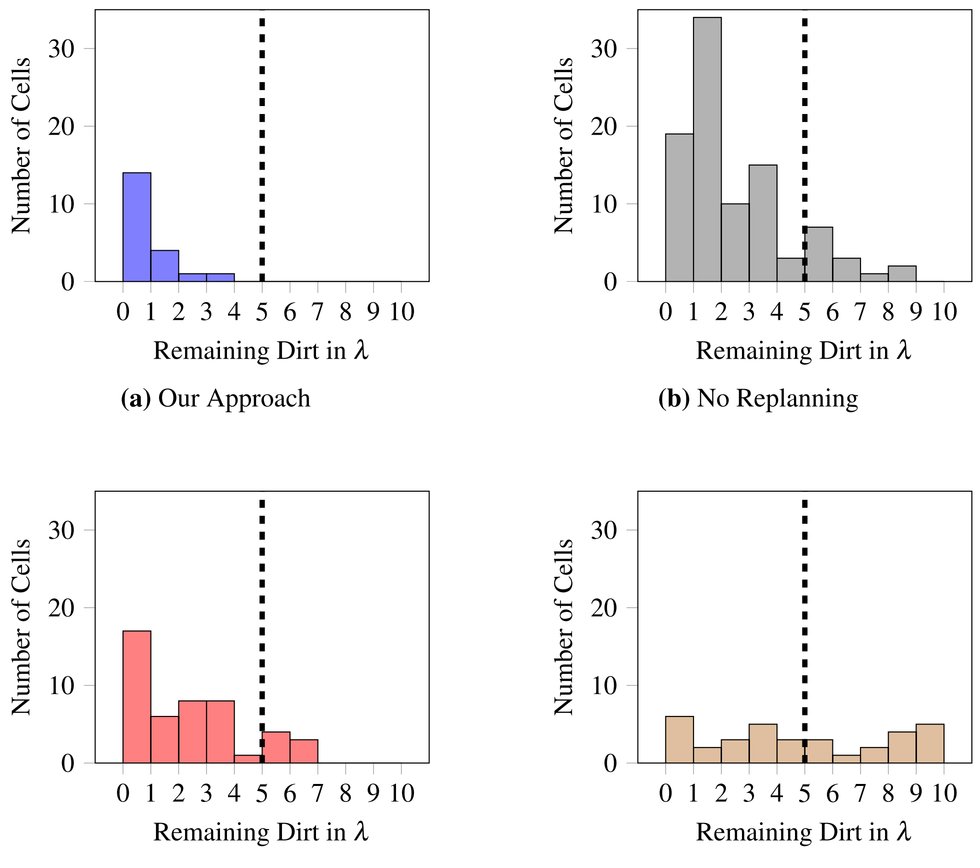
The upper part of Figure 7.1 shows the robot and the experimentation area. The lower part of the figure shows the occupancy grid map and the initial dirt distribution used in the experiment. The resolution of the cells of the dirt distribution is 0.10 m. For this experiment, we manually specified the initial values of the dirt distribution. Concretely, we selected 25 cells c_i (see Figure 7.1), for which we specified an initial λ_0^i value of the Poisson distribution of 10, 15, 20, or 25. For all other cells of the dirt distribution, we set λ_0^i to zero. To generate the ground truth for verifying the cleaning result, we proceeded as follows. For each run of the robot, we sampled a concrete dirt value from the specified initial dirt distribution for each cell. Given the previous calibration, we then computed the amount of dirt corresponding to this sample, and distributed it evenly in the cell. To measure the amount of flour that we put into the environment and the amount that was still there after cleaning, we used a precision balance.

We performed ten cleaning cycles using our approach. The first row of Table 7.1 shows the number of successful runs, i.e., the number of runs after which the 99% quantile of ground truth dirt was not above $d_{\max}$. It also shows the average distance traveled per cleaning cycle as estimated by the particle filter and the average travel time. As the table shows, our approach successfully met the threshold $d_{\max}$ in every run, in every cell. During one run, it performed on average 22.7 replanning operations that actually changed the path traveled by the robot. For comparison, we also applied three other cleaning strategies

	Successes	Distance [m]	Time [s]
Our approach	10/10	20.54	321.54
No replanning	4/10	10.98	138.51
Systematic	6/10	81.12	891.85
Roomba	0/10	-	321.54

Table 7.1.: Results of the real-world experiments. Our approach with replanning succeeded in all trials while the others failed to meet the desired dirt threshold in all cases. For comparison to our approach, we only let the Roomba strategy run for the same time as our approach.

in the same scenario. The three other strategies were the bounded dirt strategy without replanning (no replanning) as described in Section 6.2.1, a systematic approach, and the built-in strategy of the Roomba robot. For the systematic approach, we let the robot execute a sweeping path that covers the whole environment and overlaps 0.025 m parallel to the direction of movement of the robot, to account for some of the localization error in the path execution. The Roomba strategy mostly performs randomized movements but also makes use of the dirt sensor. If the robot detects dirt, it performs rotational movements to clean more thoroughly. We started the Roomba strategy at the same pose as our approach and let it run for the same amount of time as our approach. Table 7.1 shows the results of the comparison. Our approach succeeded in all trials while all other approaches did not meet the desired threshold $d_{\max}$ in at least four out of ten runs. Figure 7.4 shows a detailed histogram of the total number of cells with remaining dirt for each strategy over all runs. Clearly, our approach results in the lowest number of cells with remaining dirt all of which have dirt reduced below the threshold. In addition, for those cells the absolute amount of dirt left is small, resulting in λ values between zero and one in most cases. Due to the localization uncertainty, the no replanning approach often misses a small part of a cell and thus results in the largest number of cells with remaining dirt. Although the systematic approach plans a sweeping path that visits the entire environment, this approach still results in some cells that do not meet the threshold. Overall, the systematic approach still does not succeed in four cases. The Roomba strategy always missed at least one dirty cell entirely in the time in which our approach cleaned the whole environment. In addition, more cells with high amounts of dirt remain compared to the other strategies. As t-tests showed, the 99% quantile of the dirt values after cleaning with our approach is significantly lower than the 99% quantiles for all other approaches on a 5% level. Considering the running time and distance, Table 7.1 shows that the no replanning strategy needed shorter times and distances than our approach. This result is expected, as the no replanning strategy visits



(a) Our Approach **(b)** No Replanning **(c)** Systematic **(d)** Roomba

Figure 7.4.: Histograms of the cells with remaining dirt for each strategy over all ten runs measured in λ. The black vertical line marks the maximal allowed dirt value $d_{\max}$. Clearly, our approach (a) outperforms the other approaches and meets the maximally allowed dirt threshold every time. In addition, the number of cells with remaining dirt is smaller than for the other approaches and for those cells the absolute amount of dirt left is also small.

each selected cell only once. The systematic strategy needed more than double the time of our approach in this scenario. Our approach needs significantly shorter times and distances than the systematic strategy on a 5% level. For the Roomba strategy, no average distance is available, as our distance estimate is based on the particle filter data.

Still, the results of this experiment raise additional questions: Does the structure of the environment influence the result? Is our approach still efficient if the dirt is more widely distributed? Does the incorporation of the dirt sensor measurements in the particle filter influence the localization performance? To answer these questions, we performed a number of simulation experiments.

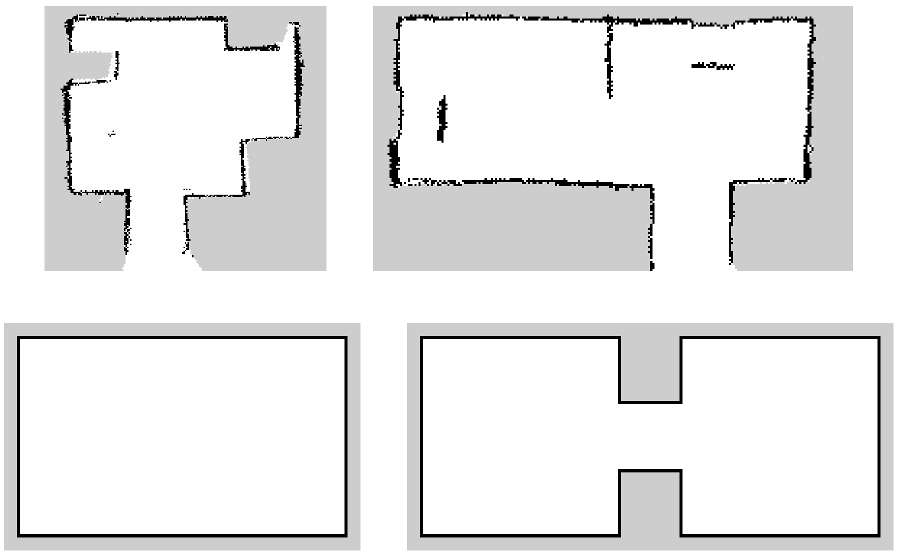


Figure 7.5.: Environments used for the simulation experiments. The maps in the upper row were recorded using the real robot.

7.3.2. Evaluation of the Cleaning Performance in Simulation

In the first set of simulation experiments, we evaluated the influence of the structure of the environment and of the amount of dirt in the environment on the performance of our approach. We applied our approach in four different environments, two maps obtained from real-world robotic experiments and two artificial ones (see Figure 7.5). In addition to the different environments, we evaluated three different percentages (5%, 10% and 20%) of dirty cells in the environments and five different ranges of λ_0^i values (5-10, 10-15, 15-20, 20-25, and 25-30) of expected dirt in the dirty cells. We uniformly sampled the dirty cells in the environment from the set of all cells of the dirt distribution, and we uniformly sampled λ_0^i values of each dirty cell from the considered range of values. For the sensors and actuators of the simulated cleaning robot, we used the parameters obtained in the real-world experiments. For comparison, we also applied the no replanning and the systematic approach described in the previous section. As the Roomba strategy was not available in simulation, we considered another random strategy. This strategy drives straight ahead until it hits a wall, randomly rotates on the spot, and drives forward again. Similar to the real-world experiment, in each run, we stopped the random strategy as soon as the robot traveled the same Euclidean distance as our approach. For each strategy, environment, percentage of dirty cells, and range of dirt levels, we performed 100 trials. We evaluated the distance traveled by the robot and the 99% quantile of the ground truth

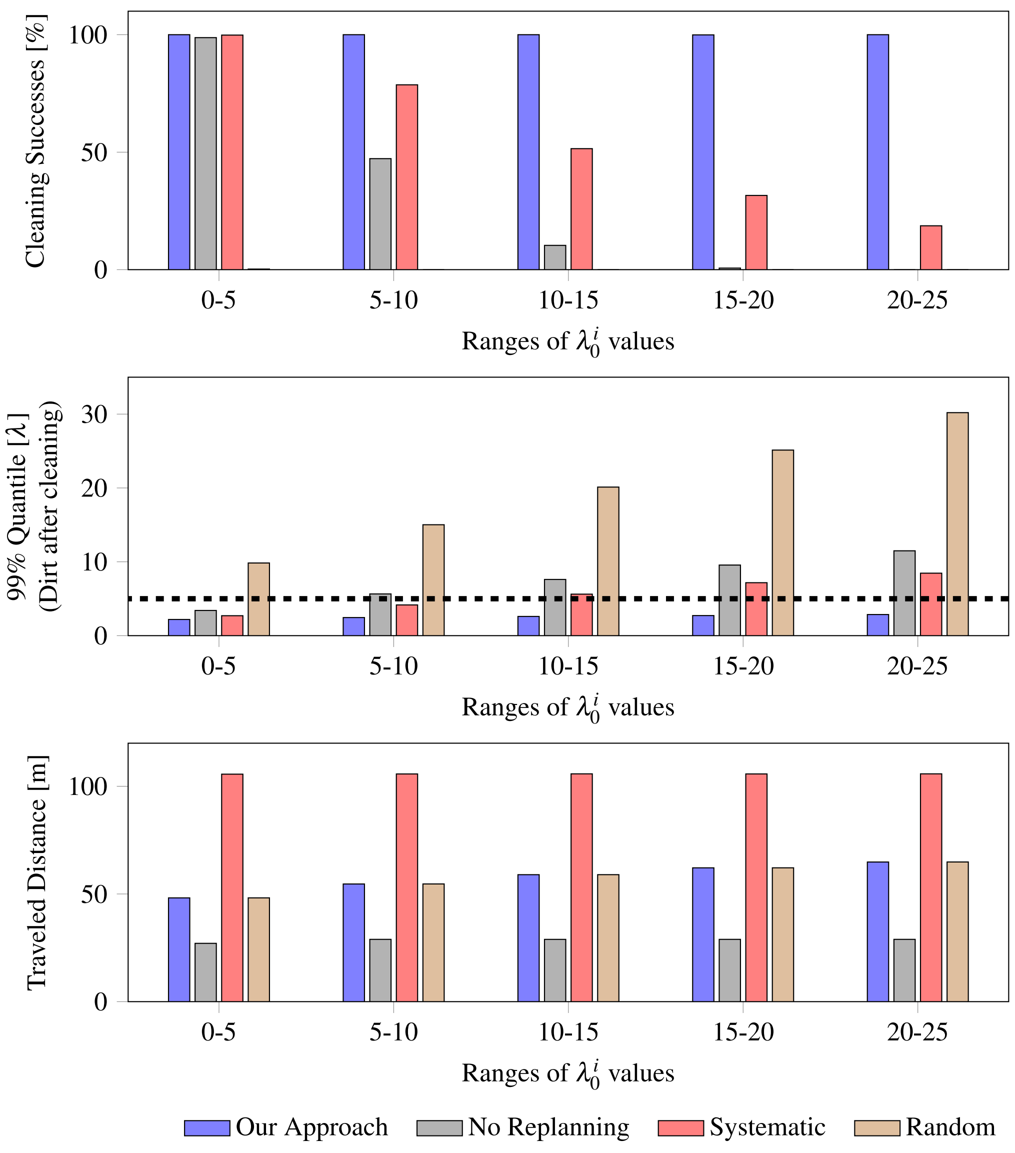


Figure 7.6.: Results of the simulation experiments averaged over all environments and percentages of dirty cells. The horizontal line in the middle plot marks the maximum allowed dirt level $d_{\max}$. As can be seen, our approach resulted in a 100% success rate and thus succeeded in every single trial. In contrast, the no replanning and the systematic approach only result in a high success rate for low dirt levels. Their success rate decreases substantially with increasing amounts of dirt per cell. As the amount of dirt per cell in the environment increases, the traveled distance of our approach also increases as more cleaning actions are needed to reduce the dirt to the desired level.

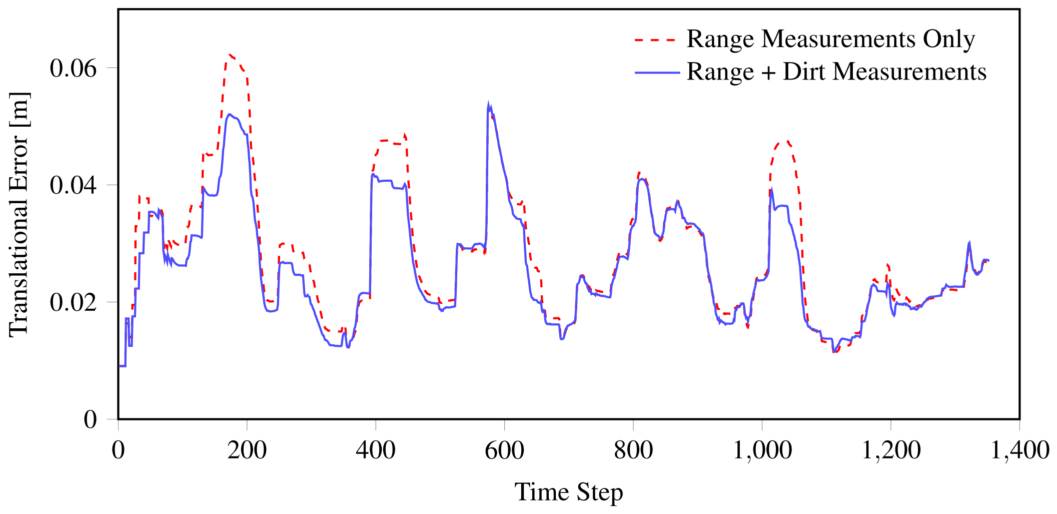


Figure 7.7.: Translational localization error with and without dirt measurements. Using the dirt measurements in addition does not disturb the localization but can reduce the localization error for large parts of the trajectory. Note the reduction in error when the robot enters the dirty parts of the environment around the time steps 200, 400, and 1.000.

dirt remaining after each run. We considered a run as successful if the 99% quantile was not above $d_{\max} = 5$.

Figure 7.6 shows the results of the simulation experiments averaged over all environments and percentages of dirty cells in the environments. The individual results for the different environments and percentages were qualitatively similar. Only the random strategy performed slightly better in the plain rectangular environment than in the others, but in general resulted in a high value of remaining dirt. As the figure shows, our approach succeeded in every single trial and maintained its success rate for increasing values of dirt per cell, while the success rates of all other approaches decreased. For every range of λ_0^i values, t-tests showed that the 99% quantiles of the dirt remaining in the environment after cleaning with our approach are significantly lower than the 99% quantiles for all other approaches on a 5% level. As Figure 7.6 shows, the no replanning approach traveled the shortest distance. As our approach re-visits previously imperfectly cleaned cells, it needs to travel a longer distance, which increases with the amount of dirt per cell. Still, compared to the systematic approach, t-tests showed that our approach resulted in significantly shorter traveled distances for every evaluated range of λ_0^i values on a 5% level. In real-world environments, the dirty cells are typically not uniformly distributed but clustered, e.g., around the dinner table or the entrance area. This would reduce the distance traveled by our approach even more, as the average distance between two dirty cells would be smaller.

7.3.3. Evaluation of the Localization Performance

In a second set of simulation experiments, we evaluated the influence of the dirt sensor measurements on the localization performance of the robot. We used the occupancy grid map and the dirt distribution from the real-world experiments described in Section 7.3.1 and simulated 100 runs with the no replanning strategy. In each run, we applied a particle filter localization both without integrating the dirt sensor measurements and with integrating the dirt sensor measurements as described in Section 7.1.1. For each time step, we recorded the translational error between the ground truth and the most likely particle.

Figure 7.7 shows the resulting errors averaged over all runs. Especially in parts of the trajectory with high localization errors, integrating the dirt measurements can substantially reduce the localization error. This can be observed around the time steps 200, 400, and 1,000 for example, which correspond to parts of the traversed trajectory shown in Figure 7.1 at which the robot enters dirty areas. A t-test revealed that averaged over all time steps, our state estimation approach results in significantly smaller localization errors than the particle filter localization without dirt sensor measurements on a 5% level.

7.4. Related Work

In the area of robotic cleaning, many systems have been developed [76, 81, 92, 121], ranging from cleaning robots for chain stores [92] to pipe cleaning robots [76]. Most robotic cleaners either perform random movements or do coverage path planning, i.e., they seek the shortest path that covers the entire environment (see Section 4.5). However, they typically do not explicitly consider the motion uncertainty of the robot and can thus not guarantee that each part of the environment was visited. Bretl and Hutchinson [13] present an approach that proves that if the position and velocity errors of the robot are bounded, complete coverage can be guaranteed for a particular choice of the coverage path. In contrast, we do not aim at complete coverage, considering worst case errors beforehand, but re-plan the path during operation according to our estimate of the remaining dirt. In the area of surveillance

To estimate the joint distribution of the trajectory of the robot and the dirt in the environment, we factor the distribution similar to approaches in the area of simultaneous localization and mapping (SLAM) [145]. There exist several SLAM approaches for cleaning robots [59, 74, 157] (see Section 6.4). In contrast to typical SLAM approaches, which initially have no information about the environment and estimate a static map, our approach uses an initially available occupancy grid map and an initial guess of the dirt distribution. It simultaneously estimates the pose of the robot and the changing state of the dirt distribution. Closer to our estimation framework is the work of Stachniss and Burgard [136], who jointly estimate the pose of the robot and the actual configuration of dynamic areas.

In this chapter, we derive probabilistic models for the dirt actuator (cleaning unit) of the robot and for the dirt sensor inside the cleaning unit. There exists a large body of literature on modeling novel sensors and actuators for robots, e.g., air flow sensors [110], whisker sensors [118], or flippers for underwater robots [52]. For window cleaning robots, Katsuki et al. [81], developed a dirt detection sensor and aim at guaranteeing that the robot cleans the complete window using a specialized motion control method. In contrast to our approach, they do not consider motion and sensor uncertainties. To the best of our knowledge, there exists no other approach that models dirt actuators and sensors inside a probabilistic framework.

In Chapter 5, we provided an approach to cleaning with a manipulation robot. As a result of learning the transition model, the robot can identify the dirty parts of the environment as those that can be changed by cleaning actions. We then planned a path for the end effector and traversed it entirely before re-planning. In contrast to this previous method, the workspace of the vacuum cleaning robot is very large and the cost of travel cannot be neglected. In addition, we can only observe the change induced by the robot at its current position. In this chapter, we therefore re-plan the path online given the current estimate of the dirt distribution.

7.5. Conclusions

In this chapter, we presented a novel approach for jointly estimating the trajectory of the cleaning robot and the distribution of dirt in the environment. For this, we developed probabilistic models of the cleaning unit as well as the dirt sensor of the robot. We showed how to use our approach to adapt the cleaning path during operation, leading to guaranteed low dirt levels in the entire environment with high confidence. Extensive experiments, carried out in simulation as well as with a real vacuum cleaning robot, demonstrated that our approach outperforms other approaches in terms of cleanliness after operation. As a side effect, it improves the localization performance of the robot.

Chapter 8

Conclusions

In this thesis, we presented several innovations to robot cleaning. The main objective was to devise novel approaches that enable service robots to clean arbitrary surfaces, perform cleaning tasks more efficiently and guarantee a clean environment for the user.

A central task of mobile robots is to safely navigate in the environment, a task which typically requires a map thereof. In addition, service robots require a dense estimate of the surface structure, for tasks such as dusting or wiping of objects. Using RGB-D cameras as a sensor has several advantages for the task of mapping. They provide dense depth and color information at a high frame rate, a low weight, and a cheap price. Furthermore, the acquisition of depth information is independent of the texture of the environment. These properties render RGB-D cameras especially suitable for low cost household robots and for obtaining dense representations of indoor environments in general. We therefore investigated a SLAM approach to dense 3D object reconstruction and mapping based on the data provided by an RGB-D camera. In the first step, our system estimates the relative motion between camera frames based on the depth information of matched visual features. For the application of robot mapping, it also integrates the odometry information of the robot for estimating the camera motion. In the second step, we optimize the local motion estimates using a nonlinear least-squares framework. This results in a globally consistent estimate of the camera trajectory. The trajectory allows for computing different task-dependent environment representations such as point clouds or voxel maps. In extensive experiments, performed on a publicly available benchmark dataset, we demonstrated the accuracy of our approach and characterized its performance in challenging real-world environments and settings.

Given a dense 3D point cloud of an object, we aimed at extending the capabilities of service robots to tasks such as dusting or wiping of objects. For this, we investigated techniques that allow a manipulation robot to cover arbitrarily complex object surfaces with a tool. Current cleaning robots mostly act on planar environments such as floor surfaces.

They often aim at covering the entire workspace while minimizing the Euclidean distance traveled. For a manipulation robot, however, minimizing the Euclidean distance on the surface does not necessarily yield the smallest execution time. Instead, cost functions in the joint space of the robot have to be considered. Framing the problem of covering the entire surface with a tool as a generalized traveling salesman problem allowed us to consider user-defined cost functions in the joint space of a manipulation robot at the time of planning. Furthermore, we proposed a computationally more efficient hierarchical approximation to our approach. Experiments with real data obtained from a manipulation robot showed that, using our approach, a manipulation robot can cover the entire reachable surface of complex objects. Additionally, the general approach as well as the proposed approximation outperform Euclidean coverage in terms of manipulation effort and time.

Some tasks do not require cleaning the entire surface, e.g., removing a stain left by a cup on a table or wiping something away that was spilled on the floor. If a service robot had knowledge about the dirty parts in the environment, it could focus on these parts and thus clean more efficiently. We therefore investigated an unsupervised vision-based approach that enables a manipulation robot to observe its own actions and learn the transition model of different parts of the environment. This allows the robot to identify the dirty parts in the environment and to verify its cleaning success.

We furthermore transfered the idea of selectively cleaning the dirty parts to vacuum cleaning robots. These robots, however, can only observe a very small part of a potentially large workspace and therefore need to be able to make predictions about the dirt in the entire environment. For this prediction it is essential to consider the fact that some parts in the environment become dirty more quickly than others, e.g., the kitchen compared to the bedroom. We thus addressed the problem of estimating the distribution of dirt in the environment as well as its dynamics over time. For modeling this distribution, we used cell-wise Poisson processes and showed how their parameters can be learned. This representation allows a cleaning robot to predict the increase of the dirt in the entire environment since the last cleaning run and thus to estimate the current amount. Based on this prediction, we proposed two efficient cleaning policies. Compared to other approaches, they minimize the cleaning time given the areas to be cleaned or optimize the cleaning result given a limited cleaning time.

Due to imperfect cleaning units, however, robots may not remove the dirt entirely on the first pass. In addition, low cost consumer robots typically exhibit high motion and sensor noise, which can yield a high localization uncertainty. In turn, this means that the robot might miss visiting some of the planned locations. For guaranteeing a clean environment, a robot therefore has to consider the uncertainties involved when performing the cleaning task. We therefore proposed a fully probabilistic framework for jointly estimating the pose of the robot as well as the current distribution of dirt in the environment during the execution of the cleaning task. For specifying the uncertainty of both the dirt sensor and the cleaning unit of a vacuum cleaning robot, we provided probabilistic models for them. The estimate about the current dirt distribution allows the robot to adapt its cleaning path during operation. As a result, our approach guarantees low dirt levels in the entire environment with high confidence. Through extensive experiments performed in simulation as well as with a real vacuum cleaning robot, we showed that our cleaning strategy outperforms other approaches in terms of cleanliness and can even improve the localization performance of the robot.

In summary, the approaches presented in this thesis allow a robot to:

- obtain an accurate, globally consistent 3D model of the environment or an object,
- plan a coverage path for the surface of arbitrary 3D objects that minimizes a user-defined cost function in the joint space of a manipulation robot,
- identify those parts of the environment that currently need cleaning,
- exploit this information for efficient cleaning strategies and guaranteeing a clean environment.

All presented techniques were implemented and thoroughly evaluated in simulation as well as on real-world data. By implementing our approaches on real robotic hardware we demonstrated their applicability. To support our claims, we performed extensive experiments and compared the presented approaches to the state of the art whenever possible. We are convinced that our techniques extend the tasks future service robots can perform, increase their robustness and make them more efficient.

Future Work

Despite the encouraging results, several open questions and directions for further research remain.

In the area of 3D mapping of an environment, future research could further increase the accuracy and robustness of the presented system in even more challenging settings. The presented mapping approach assumes that the environment is static during the process of mapping. For the practical application of household cleaning, this is often not the case as people move around in the presence of the robot or make changes in environment by shifting furniture, for example. Although our system is robust to a few dynamic objects with major parts of the sensor measurement still being static, the explicit filtering of dynamic objects would contribute to an increased robustness and accuracy. In addition, the voxel maps resulting from several runs could be compared to extract the static background and identify movable objects such as chairs.

For other applications, such as autonomously cleaning industrial halls, the short range of the RGB-D camera poses the problem that depth measurements are often not available. In these situations the robustness may be increased by performing bundle adjustment given the feature locations in the color image only, without relying on depth information. In addition, the loop closure search could be modified to be more suitable for larger environments by comparing to a database of previously collected images or visual features. For the application of robot mapping, the robustness may be increased by trying to access the quality of the motion estimates obtained by the odometry or through feature matching. This may be realized through active detection of disturbances by analyzing the acceleration information provided by an inertial measurement unit.

In the area of covering a 3D surface, it would be interesting to extend our approach to include the movement of the base. Currently, our presented approach only considers covering the reachable surface of an object given the base position of the robot. It thus requires higher-level planning of the base positions. In addition, our coverage technique could also be extended to bi-manual manipulation needed for tasks such as cleaning a plate that the robot is holding in one of its arms.

For robot cleaning, the idea of the robot learning from its own actions should be further exploited. Using force torque sensors, for example, a manipulation robot could measure and learn about the force that needs to be applied for cleaning different kinds of surfaces. In addition to learning from its own actions, the robot could also learn from user demonstrations. Special movement patterns may be taught by a human, enabling the robot to quickly adapt to other tasks such as scraping or polishing. Furthermore, future research is required to more robustly visually identify the dirty parts of a surface, such as the stain left by a cup.

We are confident that our approach to learning the dirt distribution as well as the efficient cleaning strategies we proposed could be utilized by commercial cleaning robots. One open question, however, is how the robot can automatically adapt or learn the actuator model for the various types of dirt and surfaces to be cleaned in different household environments.

Service robotics in general remains an exciting research area, due to the many possible applications for future service robots such as tidying up or doing the laundry. Apart from the above suggestions regarding the presented techniques, we are convinced that our approaches contribute to bringing applications in service robotics closer to practical realization.

List of Figures

Bibliography

[1] P. Agarwal, G. D. Tipaldi, L. Spinello, C. Stachniss, and W. Burgard. Robust map optimization using dynamic covariance scaling. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 62–69, 2013.

[2] P. Agarwal, G. Grisetti, G. D. Tipaldi, L. Spinello, W. Burgard, and C. Stachniss. Experimental analysis of dynamic covariance scaling for robust map optimization under bad initial estimates. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 3626–3631, 2014.

[3] S. Agarwal, K. Mierle, and Others. Ceres solver. https://code.google.com/p/ceres-solver/ [Last accessed on 2014-12-05].

[4] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-D point sets. *IEEE Transactions on Pattern Analysis and Machine Learning*, (5):698–700, 1987.

[5] P. Atkar, A. Greenfield, D. Conner, H. Choset, and A. Rizzi. Hierarchical segmentation of surfaces embedded in r3 for auto-body painting. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 572–577, 2005.

[6] P. Atkar, A. Greenfield, D. Conner, H. Choset, and A. Rizzi. Uniform coverage of automotive surface patches. *International Journal of Robotics Research*, 24(11): 883–898, 2005.

[7] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, 110:346–359, 2008.

[8] A. Behzad and M. Modarres. A new efficient transformation of the generalized traveling salesman problem into traveling salesman problem. In *Proc. of the Intl. Conf. of Systems Engineering*, pages 6–8, 2002.

[9] C. Bishop. *Pattern recognition and machine learning*. Springer, 2007.

[10] G. Bradski. The OpenCV library. *Dr. Dobb's Journal of Software Tools*, pages 120–126, 2000.

[11] G. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 2008.

[12] A. Breitenmoser, J. Metzger, R. Siegwart, and D. Rus. Distributed coverage control on surfaces in 3D space. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 5569–5576, 2010.

[13] T. Bretl and S. Hutchinson. Robust coverage by a mobile robot of a planar workspace. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 4582–4587, 2013.

[14] W. Burgard, C. Stachniss, G. Grisetti, B. Steder, R. Kümmerle, C. Dornhege, M. Ruhnke, A. Kleiner, and J. D. Tardós. A comparison of slam algorithms based on a graph of relations. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 2089–2095, 2009.

[15] A. Censi. Scan matching in a probabilistic framework. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 2291–2296, 2006.

[16] P. Cheng, J. Keller, and V. Kumar. Time-optimal UAV trajectory planning for 3D urban structure coverage. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 2750–2757, 2008.

[17] H. Choset. Coverage for robotics – a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31(1):113–126, 2001.

[18] H. Choset and P. Pignon. Coverage path planning: The boustrophedon cellular decomposition. In *Int. Conf. on Field and Service Robotics*, pages 203–209, 1997.

[19] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, 1976.

[20] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[21] J. J. Craig and M. Raibert. A systematic method of hybrid position/force control of a manipulator. In *The IEEE Intl. Conf. on Computer Software and Applications (COMPSAC)*, pages 446–451, 1979.

[22] M. J. Cummins and P. M. Newman. Fab-map: Appearance-based place recognition and mapping using a learned visual vocabulary model. In *Proc. of the Intl. Conf. on Machine Learning (ICML)*, pages 3–10, 2010.

[23] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Proc. of the Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 303–312. ACM, 1996.

[24] A. J. Davison and D. W. Murray. Simultaneous localization and map-building using active vision. *IEEE Transactions on Pattern Analysis and Machine Learning*, 24(7): 865–880, 2002.

[25] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Learning*, 29(6):1052–1067, 2007.

[26] F. Dellaert and M. Kaess. Square Root SAM: Simultaneous localization and mapping via square root information smoothing. *International Journal of Robotics Research*, 25(12):1181–1204, 2006.

[27] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 1322–1328, 1999.

[28] R. Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, 2010. URL http://www.programmingvision.com/rosen_diankov_thesis.pdf.

[29] V. Dimitrijević and Z. Šarić. An efficient transformation of the generalized traveling salesman problem into the traveling salesman problem on digraphs. *Information Sciences*, 102(1–4):105–110, 1997.

[30] M. Do, J. Schill, J. Ernesti, and T. Asfour. Learn to wipe: A case study of structural bootstrapping from sensorimotor experience. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 1858–1864, 2014.

[31] A. Doucet, N. DeFreitas, and N. Gordon. *Sequential monte carlo methods in practice*. Springer Verlag, 2001.

[32] T. Duckett, S. Marsland, and J. Shapiro. Fast, on-line learning of globally consistent maps. *Journal of Autonomous Robots*, 12(3):287–300, 2002.

[33] F. Endres, J. Hess, N. Engelhard, J. Sturm, and W. Burgard. ROS documentation of RGBD-SLAM. http://www.ros.org/wiki/rgbdslam [Last accessed on 2014-12-05], 2011.

[34] F. Endres, J. Hess, N. Engelhard, J.Sturm, and W. Burgard. 6D visual SLAM for RGB-D sensors. *at - Automatisierungstechnik*, 60:270–278, 2012.

[35] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evaluation of the RGB-D SLAM system. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 1691–1696, 2012.

[36] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard. 3D mapping with an RGB-D camera. *IEEE Transactions on Robotics*, pages 177–187, 2 2014.

[37] N. Engelhard, F. Endres, J. Hess, J. Sturm, and W. Burgard. Real-time 3D visual SLAM with a hand-held camera. In *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum*, 2011.

[38] B. Englot and F. Hover. Inspection planning for sensor coverage of 3d marine structures. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 4412–4417, 2010.

[39] C. Eppner, J. Sturm, M. Bennewitz, C. Stachniss, and W. Burgard. Imitation learning with generalized task descriptions. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 3968–3974, 2009.

[40] L. Erickson, J. Knuth, J. O'Kane, and S. LaValle. Probabilistic localization with a blind robot. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 1821–1827, 2008.

[41] U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[42] M. Fischler and R. 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.

[43] W. D. Fisher and M. S. Mujtaba. Hybrid position/force control: a correct formulation. *International Journal of Robotics Research*, 11(4):299–311, 1992.

[44] A. W. Fitzgibbon. Robust registration of 2D and 3D point sets. *Image and Vision Computing*, 21(13):1145–1153, 2003.

[45] R. W. Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

[46] J. Foley, A. Van Dam, S. Feiner, J. Hughes, and R. Phillips. *Introduction to computer graphics*. Addison-Wesley Reading, 1994.

[47] B. Frank, R. Schmedding, C. Stachniss, M. Teschner, and W. Burgard. Real-world robot navigation amongst deformable obstacles. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 1649–1654, 2009.

[48] Y. Gabriely and E. Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, number 12 in Combinatorial Optimization, pages 1927–1933, 2001.

[49] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[50] M. R. Garey, R. L. Graham, and D. S. Johnson. Some np-complete geometric problems. In *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing*, STOC '76, pages 10–22, New York, NY, USA, 1976. ACM.

[51] J. Geweke. Bayesian inference in econometric models using monte carlo integration. *Econometrica*, 57(6):1317–1339, 1989.

[52] P. Giguere, C. Prahacs, and G. Dudek. Characterization and modeling of rotational responses for an oscillating foil underwater robot. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 3000–3005, 2006.

[53] A. Gil, O. Reinoso, O. M. Mozos, C. Stachniss, and W. Burgard. Improving data association in vision-based slam. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 2076–2081, 2006.

[54] S. Griffith, J. Sinapov, M. Miller, and A. Stoytchev. Toward interactive learning of object categories by a robot: A case study with container and non-container objects. In *Proc. of the Int. Conf. on Development and Learning (ICDL)*, pages 1–6, 2009.

[55] G. Grisetti, C. Stachniss, and W. Burgard. Non-linear constraint network optimization for efficient map learning. *IEEE Transactions on Intelligent Transportation Systems*, 10:428–439, 2009.

[56] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard. A tutorial on graph-based SLAM. *IEEE Transactions on Intelligent Transportation Systems Magazine*, 2: 31–43, 2010.

[57] G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, and C. Hertzberg. Hierarchical optimization on manifolds for online 2D and 3D mapping. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 273–278, 2010.

[58] G. Gutin and A. P. Punnen. *The traveling salesman problem and its variations*, volume 12. Springer, 2002.

[59] J. Gutmann, E. Eade, P. Fong, and M. Munich. A constant-time algorithm for vector field slam using an exactly sparse extended information filter. In *Proc. of Robotics: Science and Systems (RSS)*, 2010.

[60] A. Harrison and P. Newman. Image and sparse laser fusion for dense scene reconstruction. In *Field and Service Robotics*, pages 219–228. Springer, 2010.

[61] K. Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126:106–130, 2000.

[62] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using kinect-style depth cameras for dense 3D modeling of indoor environments. *International Journal of Robotics Research*, 31(5):647–663, 2012.

[63] A. Henry-Labordere. Record balancing problem-a dynamic programming solution of a generalized traveling salesman problem. *Revue Francaise D Informatique De Recherche Operationnelle*, (B-2):673–743, 1969.

[64] J. Hess, J. Sturm, and W. Burgard. Learning the state transition model to efficiently clean surfaces with mobile manipulation robots. In *Proc. of the Workshop on Manipulation under Uncertainty at the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011.

[65] J. Hess, D. Tipaldi, and W. Burgard. Null space optimization for effective coverage of 3D surfaces using redundant manipulators. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 1923–1928, 2012.

[66] J. Hess, M. Beinhofer, D. Kuhner, P. Ruchti, and W. Burgard. Poisson-driven dirt maps for efficient robot cleaning. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 2245–2250, 2013.

[67] J. Hess, M. Beinhofer, and W. Burgard. A probabilistic approach to high-confidence cleaning guarantees for low-cost cleaning robots. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 5600–5605, 2014.

[68] B. K. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, 4(4):629–642, 1987.

[69] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Journal of Autonomous Robots*, 34:189–206, April 2013.

[70] G. Hu, S. Huang, L. Zhao, A. Alempijevic, and G. Dissanayake. A robust RGB-D slam algorithm. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 1714–1719, 2012.

[71] W. Huang. Optimal line-sweep-based decompositions for coverage algorithms. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 27–32, 2006.

[72] D. Q. Huynh. Metrics for 3D rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35(2):155–164, 2009.

[73] iRobot Corp. iRobot: Our history. http://www.irobot.com/us/Company/About/Our_History.aspx [Last accessed on 2014-12-05], 2014.

[74] W. Jeong and K. Lee. CV-SLAM: A new ceiling vision-based SLAM technique. In *Proc. of Robotics: Science and Systems (RSS)*, pages 3195–3200, 2005.

[75] D. Johnson. Approximation algorithms for combinatorial problems. *Computer and System Sciences*, 9(3):256–278, 1974.

[76] C. Jung, W. Chung, J. Ahn, M. Kim, G. Shin, and S. Kwon. Optimal mechanism design of in-pipe cleaning robot. In *Proc. of the IEEE Intl. Conf. on Mechatronics and Automation (ICMA)*, pages 1327–1332, 2011.

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

[78] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert. iSAM2: Incremental smoothing and mapping using the Bayes tree. *International Journal of Robotics Research*, 31:217–236, 2012.

[79] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[80] R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[81] Y. Katsuki, T. Ikeda, and M. Yamamoto. Development of a high efficiency and high reliable glass cleaning robot with a dirt detect sensor. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 5133–5138, 2011.

[82] C. Kerl, J. Sturm, and D. Cremers. Dense visual SLAM for RGB-D cameras. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 2100–2106, 2013.

[83] G. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *IEEE and ACM Intl. Symposium on Mixed and Augmented Reality*, pages 225–234. IEEE, 2007.

[84] K. Konolige, G. Grisetti, R. Kummerle, W. Burgard, B. Limketkai, and R. Vincent. Efficient sparse pose adjustment for 2d mapping. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 22–29, 2010.

[85] B. Korte and J. Vygen. *Combinatorial optimization*, volume 12. Springer, 2012.

[86] T. Krajník, P. J. Fentanes, G. Cielniak, C. Dondrup, and T. Duckett. Spectral analysis for long-term robotic mapping. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 1–27, 2014.

[87] E. Kruse and F. Wahl. Camera-based observation of obstacle motions to derive statistical data for mobile robot motion planning. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 662–667, 1998.

[88] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 3607–3613, 2011.

[89] M. Lagoudakis, E. Markakis, D. Kempe, P. Keskinocak, A. Kleywegt, S. Koenig, C. Tovey, A. Meyerson, and S. Jain. Auction-based multi-robot routing. In *Proc. of Robotics: Science and Systems (RSS)*, pages 343–350, 2005.

[90] J. Latombe. *Robot Motion Planning*. Springer Verlag, 1990.

[91] S. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[92] G. Lawitzky. A navigation system for cleaning robots. *Journal of Autonomous Robots*, 9(3):255–260, 2000.

[93] J. Le Ny, E. Feron, and E. Frazzoli. On the Dubins traveling salesman problem. *IEEE Transactions on Automatic Control*, 57(1):265–270, 1 2012.

[94] D. Leidner, A. Dietrich, F. Schmidt, C. Borst, and A. Albu-Schäffer. Object-centered hybrid reasoning for whole-body mobile manipulation. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 1828–1835, 2014.

[95] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[96] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proc. of the Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 163–169, 1987.

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

[98] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Journal of Autonomous Robots*, 4(4):333–349, 1997.

[99] M. Luber, G. Tipaldi, and K. Arras. Place-dependent people tracking. In *Proc. of the Intl. Symposium of Robotics Research (ISRR)*, pages 280–293, 2009.

[100] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM (JACM)*, 41(5):960–981, 1994.

[101] D. Macharet, A. Neto, V. da Camara Neto, and M. Campos. Nonholonomic path planning optimization for Dubins' vehicles. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 4208–4213, 2011.

[102] J. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. L. Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297. University of California Press, 1967.

[103] R. Mannadiar and I. Rekleitis. Optimal coverage of a known arbitrary environment. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 5525–5530, 2010.

[104] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics*, 11(2):431–441, 1963.

[105] G. Metta and P. Fitzpatrick. Early integration of vision and manipulation. *Adaptive Behavior*, 11(2):109–128, 2003.

[106] S. Miller. Code for integrating, raytracing, and meshing a tsdf on the cpu. https://github.com/sdmiller/cpu_tsdf [Last accessed on 2014-12-05], 2013.

[107] H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 116–121, 1985.

[108] MotionAnalysis. Raptor-E Digital RealTime System. http://www.motionanalysis.com/html/industrial/raptore.html [Last accessed on 2014-12-05].

[109] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009.

[110] J. Müller, O. Paul, and W. Burgard. Probabilistic velocity estimation for autonomous miniature airships using thermal air flow sensors. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 39–44, 2012.

[111] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Proc. of the IEEE Symposium on Mixed and Augmented Reality (ISMAR)*, pages 127–136, 2011.

[112] C. E. Noon and J. C. Bean. A lagrangian based approach for the asymmetric generalized traveling salesman problem. *Operations Research*, 39(4):623–632, 1991.

[113] J. Oh, Y. Choi, J. Park, and Y. Zheng. Complete coverage navigation of cleaning robots using triangular-cell-based map. *IEEE Transactions on Industrial Electronics*, 51(3):718–726, 6 2004.

[114] E. Olson, J. Leonard, and S. Teller. Fast iterative optimization of pose graphs with poor initial estimates. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 2262–2269, 2006.

[115] E. B. Olson. Real-time correlative scan matching. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 4387–4393, 2009.

[116] C. Papadimitriou. The Euclidean traveling salesman problem is NP-complete. *Theoretical Computer Science*, 4(3):237–244, 1977.

[117] F. C. Park. Distance metrics on the rigid-body motions with applications to mechanism design. *Journal of Mechanical Design*, 117(1):48–54, 1995.

[118] M. Pearson, C. Fox, J. Sullivan, T. Prescott, T. Pipe, and B. Mitchinson. Simultaneous localisation and mapping on a multi-degree of freedom biomimetic whiskered robot. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 586–592, 2013.

[119] D. Pelleg and A. Moore. X-means: Extending K-means with efficient estimation of the number of clusters. In *Proc. of the Intl. Conf. on Machine Learning (ICML)*, pages 727–734, San Francisco, USA, 2000.

[120] E. Prassler and K. Kosuge. Domestic robotics. In B. Siciliano and O. Khatib, editors, *Springer Handbook of Robotics*, pages 1253–1281. Springer, 2008.

[121] E. Prassler, A. Ritter, C. Schaeffer, and P. Fiorini. A short history of cleaning robots. *Journal of Autonomous Robots*, 9(3):211–226, 2000.

[122] H. Roth and M. Vona. Moving volume kinectfusion. In *British Machine Vision Conf. (BMVC)*, pages 1–11, 2012.

[123] J. Röwekämper, C. Sprunk, G. Tipaldi, C. Stachniss, P. Pfaff, and W. Burgard. On the position accuracy of mobile robot localization based on particle filters combined with scan matching. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 3158–3164, 2012.

[124] D. Rubin. Using the SIR algorithm to simulate posterior distributions. *Bayesian statistics*, 3(1):395–402, 1988.

[125] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: an efficient alternative to SIFT or SURF. In *Proc. of the IEEE Intl. Conf. on Computer Vision*, volume 13, pages 2564–2571, 2011.

[126] M. Ruhnke, R. Kümmerle, G. Grisetti, and W. Burgard. Highly accurate 3D surface models by sparse surface adjustment. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 751–757, 2012.

[127] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice Hall, 1995.

[128] M. Saha, T. Roughgarden, J.-C. Latombe, and G. Sánchez-Ante. Planning tours of robotic arms among partitioned goals. *International Journal of Robotics Research*, 25(3):207–223, 2006.

[129] R. Sahai, S. Griffith, and A. Stoytchev. Interactive identification of writing instruments and writable surfaces by a robot. In *Proc. of the Workshop on Mobile Manipulation in Human Environments at the Robotics Science and Systems Conf. (RSS)*, Seattle, WA, 2009.

[130] K. Savla, E. Frazzoli, and F. Bullo. On the point-to-point and traveling salesperson problems for Dubins' vehicle. In *Proc. of the American Control Conference*, pages 786–791, 2005.

[131] M. Schönbein and A. Geiger. Omnidirectional 3D reconstruction in augmented manhattan worlds. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2014.

[132] A. Segal, D. Haehnel, and S. Thrun. Generalized-ICP. In *Proceedings of Robotics: Science and Systems*, 2009.

[133] B. Siciliano and O. Khatib, editors. *Handbook of Robotics*. Springer, 2008.

[134] V. Simaiaki, P. Giataganas, and G.-Z. Yang. Robot assisted endomicroscopic image mosaicing with optimal surface coverage and reconstruction. In *IEEE Intl. Symposium on Biomedical Imaging (ISBI)*, pages 1432–1435, 2013.

[135] P. Slavík. A tight analysis of the greedy algorithm for set cover. In *Proc. of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 435–441. ACM, 1996.

[136] C. Stachniss and W. Burgard. Mobile robot mapping and localization in non-static environments. In *Proc. of the National Conf. on Artificial Intelligence (AAAI)*, pages 1324–1329, 2005.

[137] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard. NARF: 3D range image features for object recognition. In *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

[138] F. Steinbrücker, J. Sturm, and D. Cremers. Real-time visual odometry from dense RGB-D images. In *Workshop on Live Dense Reconstruction with Moving Cameras at the Intl. Conf. on Computer Vision (ICCV)*, pages 719–722, 2011.

[139] F. Steinbrücker, C. Kerl, J. Sturm, and D. Cremers. Large-scale multi-resolution surface reconstruction from RGB-D sequences. In *Proc. of the IEEE Intl. Conf. on Computer Vision*, pages 3264–3271, 2013.

[140] H. Strasdat, J. M. M. Montiel, and A. Davison. Scale drift-aware large scale monocular SLAM. In *Proceedings of Robotics: Science and Systems*, 2010.

[141] J. Stückler and S. Behnke. Integrating depth and color cues for dense multi-resolution scene mapping using RGB-D cameras. In *IEEE Conf. on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pages 162–167, 2012.

[142] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 573–580, 2012.

[143] S. Thrun and M. Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *International Journal of Robotics Research*, 25 (5-6):403–429, 2006.

[144] S. Thrun, D. Fox, W. Burgard, and D. F. Robust Monte-Carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2):99–141, 2001.

[145] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.

[146] S. Thrun et al. Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium*, pages 1–35, 2002.

[147] G. Tipaldi and K. Arras. Please do not disturb! minimum interference coverage for social robots. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 1968–1973, 2011.

[148] G. D. Tipaldi, D. Meyer-Delius, and W. Burgard. Lifelong localization in changing environments. *International Journal of Robotics Research*, 32(14):1662–1678, 2013.

[149] M. Tomono. Robust 3D SLAM with a stereo camera based on an edge-point ICP algorithm. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 4306–4311, 2009.

[150] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Learning*, 13 (4):376–380, 1991.

[151] H. Urbanek, A. Albu-Schäffer, and P. van der Smagt. Learning from demonstration: repetitive movements for autonomous service robotics. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 424–431, 2009.

[152] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald. Kintinuous: Spatially extended KinectFusion. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2012.

[153] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. McDonald. Robust real-time visual odometry for dense RGB-D mapping. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 5724–5731, 2013.

[154] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, pages 80–83, 1945.

[155] C. Wu. SiftGPU: A GPU implementation of scale invariant feature transform (SIFT). http://cs.unc.edu/~ccwu/siftgpu [Last accessed on 2014-12-05], 2007.

[156] A. Xu, C. Viriyasuthee, and I. Rekleitis. Optimal complete terrain coverage using an unmanned aerial vehicle. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 2513–2519, 2011.

[157] Y. Zhang, J. Liu, G. Hoffmann, M. Quilling, K. Payne, P. Bose, and A. Zimdars. Real-time indoor mapping for mobile robots with limited sensing. In *Proc. of the 3rd Intl. Workshop on Mobile Entity Localization and Tracking*, pages 636–641, 2010.

[158] B. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J. Bagnell, M. Hebert, A. Dey, and S. Srinivasa. Planning-based prediction for pedestrians. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 3931–3936, 2009.

[159] R. Zlot, A. Stentz, M. Dias, and S. Thayer. Multi-robot exploration controlled by a market economy. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 3016–3023, 2002.