# ROBOT NAVIGATION IN HUMAN ENVIRONMENTS

DISSERTATION

zur Erlangung des Doktorgrades (Dr. rer. nat.)
der Mathematisch-Naturwissenschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von
**STEFAN OSSWALD**
aus Lörrach

Bonn, Juni 2018

Angefertigt mit Genehmigung der
Mathematisch-Naturwissenschaftlichen Fakultät der
Rheinischen Friedrich-Wilhelms-Universität Bonn

| | |
|---|---|
| Erste Gutachterin: | Prof. Dr. Maren Bennewitz |
| | Rheinische Friedrich-Wilhelms-Universität Bonn |
| Zweiter Gutachter: | Prof. Dr. Wolfram Burgard |
| | Albert-Ludwigs-Universität Freiburg |
| Tag der Promotion: | 6. November 2018 |
| Erscheinungsjahr: | 2019 |

Dedicated to my grandparents Liesel and Herbert.

# ABSTRACT

For the near future, we envision service robots that will help us with everyday chores in home, office, and urban environments. In contrast to industrial applications, these robots need to work in environments that were designed for humans and they have to collaborate with humans to fulfill their tasks. In this thesis, we propose new methods for communicating, transferring knowledge, and collaborating between humans and robots in four different navigation tasks.

In the first application, we investigate how automated services for giving wayfinding directions can be improved to better address the needs of the human recipients. Route descriptions can contain several different types of information such as metric distances, cardinal directions, egocentric headings, or landmarks of various categories. Which information is most valuable depends on the recipient, for example street names are hardly of use for visually impaired persons as they cannot see the street signs. We propose a novel method based on inverse reinforcement learning that learns from a corpus of human-written route descriptions what amount and type of information a route description should contain. By imitating the human teachers' description style, our algorithm produces new route descriptions that sound similarly natural and convey similar information content as the human teachers do as we show in a user study.

In the second application, we investigate how robots can leverage background information provided by humans for exploring an unknown environment more efficiently. Typically, the first task of any mobile robot is to explore its surroundings for generating a map using a Simultaneous Localization and Mapping (SLAM) approach. The robot later uses this map for planning navigation tasks. In many cases, background information such as a floor plan of the building or a sketch provided by the user is available. We propose an algorithm for exploiting such background information given as a topo-metric graph by combining a global exploration strategy based on the solution of a traveling salesman problem with a local nearest-frontier-first exploration scheme. Our simulation experiments in both artificial and real-world environments show that the exploration tours are significantly shorter and that our system allows the user to effectively select the areas that the robot should explore. We show that the approach is robust against noise and errors in the provided graph.

In the second part of this thesis, we focus on humanoid robots in home and office environments. The human-like body plan allows humanoid robots to navigate in environments and operate tools that were designed for humans, for example to step across obstacles, climb

stairs, or open cupboards and drawers, making humanoid robots suitable for a wide range of applications. As localization and mapping are prerequisites for all navigation tasks, we first introduce a novel feature descriptor for RGB-D sensor data and integrate this building block into an appearance-based SLAM system that we adapt and optimize for the usage on humanoid robots. Our real-world experiments show that our optimized system is able to track a Nao humanoid robot more accurately and more robustly than existing approaches.

As the third application, we investigate how humanoid robots can cover known environments efficiently with their camera, for example for inspection or search tasks. We extend an existing next-best-view approach based on sampled raycasting by integrating the concept of inverse reachability maps. Pre-recording an inverse reachability map for a humanoid robot allows us to efficiently sample and check collision-free full-body poses so that the algorithm can take the full body pose into account already when sampling camera poses from where large portions of the environment are visible. Our approach enables the robot to bend over boxes and peak behind and below objects to inspect as much of the environment as possible.

In our fourth application, we extend the coverage scenario to environments that also include articulated objects that the robot has to actively manipulate to see behind them, for example to open a cupboard for investigating its contents. Introducing articulated objects increases the complexity for computing a coverage tour even further, demanding efficient approaches for navigation and camera pose sampling. Hence, we introduce algorithms for the computation of cost maps and utility maps and for estimating the information gain of a camera view pose that run highly parallelized on graphics processing units for embedded devices. Together with a novel heuristic for estimating utility maps, our system allows to find high-utility camera poses for efficiently covering environments with articulated objects.

All techniques presented in this thesis were implemented in software and thoroughly evaluated in user studies, simulations, and experiments in both artificial and real-world environments. Our approaches advance the state of the art towards universally usable robots in everyday environments.

# ZUSAMMENFASSUNG

In naher Zukunft erwarten wir Serviceroboter, die uns im Haushalt, im Büro und in der Stadt alltägliche Arbeiten abnehmen. Im Gegensatz zu industriellen Anwendungen müssen sich diese Roboter in Umgebungen zurechtfinden, die für Menschen gebaut wurden, und sie müssen mit den Menschen zusammenarbeiten um ihre Aufgaben erledigen zu können. In dieser Arbeit schlagen wir neue Methoden für die Kommunikation, Wissenstransfer und Zusammenarbeit zwischen Menschen und Robotern bei Navigationsaufgaben in vier Anwendungen vor.

In der ersten Anwendung untersuchen wir, wie automatisierte Dienste zur Generierung von Wegbeschreibungen verbessert werden können, um die Wegbeschreibungen besser an die Bedürfnisse der Empfänger anzupassen. Wegbeschreibungen können mehrere verschiedene Arten von Informationen enthalten, beispielsweise metrische Distanzen, Himmelsrichtungen, egozentrische Richtungsangaben oder Landmarken unterschiedlicher Kategorien. Welche Information für den Empfänger am wertvollsten ist, hängt von der Situation und dem Empfänger ab, beispielsweise sind Straßennamen kaum von Nutzen für sehbehinderte Menschen, die keine Straßenschilder lesen können. Wir schlagen eine neue Methode vor, die inverses bestärkendes Lernen nutzt, um aus einem Korpus von von Menschen geschriebenen Wegbeschreibungen zu lernen, wie viel und welche Art von Information eine Wegbeschreibung enthalten sollte. Indem unser Algorithmus den Stil der Wegbeschreibungen der menschlichen Lehrer imitiert, kann der Algorithmus neue Wegbeschreibungen erzeugen, die sich ähnlich natürlich anhören und einen ähnlichen Informationsgehalt vermitteln, wie wir in einer Benutzerstudie zeigen.

In der zweiten Anwendung untersuchen wir, wie Roboter von Menschen bereitgestellte Hintergrundinformationen ausnutzen können, um eine bisher unbekannte Umgebung schneller zu erkunden. Die erste Aufgabe eines mobilen Roboters besteht in der Regel darin, seine Umgebung zu erkunden um eine Karte mit einem simultanen Lokalisierungs- und Kartierungsverfahren (SLAM) zu erstellen, die der Roboter später zur Planung von Navigationsaufgaben nutzen kann. Oftmals stehen Hintergrundinformationen zur Verfügung, beispielsweise Gebäudegrundrisse oder von einem Benutzer gezeichnete Skizzen. Wir schlagen einen Algorithmus vor, der solche Hintergrundinformationen in Form eines topometrischen Graphen nutzt, indem er eine globale Explorationsstrategie basierend auf der Lösung eines Problems des Handlungsreisenden kombiniert mit einer lokalen Explorationsstrategie, welche die nächstliegende noch nicht explorierte Grenze zwischen Freifläche und noch unbekanntem Gebiet als nächs-

tes ansteuert. Unsere Simulationsexperimente mit künstlichen und realen Umgebungen zeigen, dass die Erkundungstouren signifikant kürzer werden und dass unser System dem Benutzer eine effektive Möglichkeit bietet, die Regionen zu spezifizieren, die der Roboter erkunden soll. Wir zeigen, dass der Ansatz robust ist gegenüber Ungenauigkeiten und Fehlern in dem zur Verfügung gestellten Graphen.

Im zweiten Teil dieser Arbeit legen wir den Fokus auf humanoide Roboter in Umgebungen zuhause und im Büro. Der menschenähnliche Körperbau ermöglicht es humanoiden Robotern, in Umgebungen zu navigieren und Werkzeuge zu benutzen, die für Menschen gebaut wurden. Humanoide Roboter können zum Beispiel über Hindernisse steigen, Treppen steigen oder Schranktüren und Schubladen öffnen, wodurch humanoide Roboter für eine große Bandbreite an Aufgaben eingesetzt werden können. Da Lokalisierung und Kartierung Grundvoraussetzungen für alle Navigationsaufgaben sind, führen wir zunächst einen neuen Merkmalsdeskriptor für RGB-D-Sensordaten ein und integrieren diesen Baustein in ein erscheinungsbasiertes SLAM-System, das wir an die Besonderheiten von humanoiden Robotern anpassen und optimieren. Unsere Experimente in einer realen Umgebung zeigen, dass unser optimiertes System in der Lage ist, die Position eines humanoiden Roboters genauer und robuster zu verfolgen als es mit existierenden Ansätzen möglich ist.

Als dritte Anwendung untersuchen wir, wie humanoide Roboter bekannte Umgebungen effizient mit ihrer Kamera abdecken können, beispielsweise zu Wartungs- und Inspektionszwecken oder zum Suchen eines Gegenstands. Wir erweitern ein bestehendes Verfahren, das mithilfe von Raycasting zufällig gezogener Strahlen die nächstbeste Beobachtungsposition berechnet, indem wir das Konzept von inversen Erreichbarkeitskarten integrieren. Das Vorberechnen einer inversen Erreichbarkeitskarte für einen humanoiden Roboter ermöglicht es, kollisionsfreie Ganzkörperposen effizient zu generieren und zu prüfen. Dadurch kann der Algorithmus Ganzkörperposen bereits berücksichtigen, wenn er Beobachtungspunkte in der Umgebung bestimmt, von denen aus große Teile der Umgebung sichtbar sind. Unser Ansatz ermöglicht es dem Roboter, sich über Kisten zu beugen und hinter und unter Objekte zu schauen um so viel wie möglich von der Umgebung zu untersuchen.

In unserer vierten Anwendung erweitern wir dieses Szenario bezüglich Umgebungen, die auch bewegbare Gegenstände enthalten, sodass der Roboter aktiv Objekte bewegen muss um hinter das Objekt zu sehen, beispielsweise einen Schrank zu öffnen um den Inhalt zu inspizieren. Bewegliche Gegenstände einzuführen erhöht die Komplexität des Problems eine Inspektionstour zu planen erheblich, wodurch effiziente Ansätze für die Navigation und das Bestimmen der Beobachtungspunkte notwendig werden. Daher führen wir Algorithmen für die Berechnung von Kosten- und Nutzenkarten und die Schätzung

des Informationsgewinns eines Beobachtungspunkts ein, die hoch parallelisiert auf Grafikkarten von eingebetteten Systemen ausgeführt werden. Zusammen mit einer neuen Heuristik zur Schätzung von Nutzenkarten ermöglicht dies unserem System Beobachtungspunkte mit hohem Nutzen zu finden, um Umgebungen mit bewegbaren Objekten effizient zu inspizieren.

Die vorgestellten Techniken wurden in Software implementiert und sorgfältig evaluiert in Benutzerstudien, Simulationen und Experimenten sowohl mit künstlichen als auch mit realen Umgebungen. Unsere Verfahren bringen den Stand der Forschung voran in Richtung universell einsetzbarer Roboter in alltäglichen Umgebungen.

# ACKNOWLEDGMENTS

On my journey as a PhD student, I met many people who taught, encouraged, and supported me and I would like to express my deepest thanks to them. First of all, I would like to thank Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard for giving me the opportunity to work in their labs and for supervising and counseling me throughout my studies. Their experience guided my research and they sparked my interest in robotics. I would also like to thank Olivier Stasse for giving me the opportunity of a research internship at the Gepetto team at LAAS-CNRS in Toulouse, it was a great experience and gave me many new ideas.

Many thanks to Armin Hornung for supervising me during my Bachelor and Master studies. I learned a lot from him and he prepared me well for life as a PhD student.

I would like to thank my colleagues and fellow students at the Universities of Freiburg and Bonn, at LAAS-CNRS, and my colleagues from the SFB/TR-8 in Bremen for their support and friendship.

For their contributions on our collaborative projects and research, I thank my co-authors Henrik Kretzschmar, Rasha Sheikh, Philipp Karkowski, Peter Regier, and Christian Dornhege.

Thanks to Philipp Karkowski and Rasha Sheikh for proof-reading earlier versions of this thesis.

Finally and most importantly, my deepest thanks go to my family who always encourage and support me.

The work on this thesis has been generously supported by the German Research Foundation (DFG) under contract number SFB/TR-8 "Spatial Cognition" and by the European Commission under contract number FP7-ICT-600890-ROVINA.

*We're fascinated with robots because they are reflections of ourselves.*

— Ken Goldberg

# CONTENTS

# 1 INTRODUCTION

Robots are about to step into our daily lives: Vacuum cleaner robots and autonomous lawn mowers have already conquered many homes, robots work as concierges in hotels, as servers in restaurants, and as tour guides in museums, they park our cars in automated garages or even drive cars themselves. For the future, we envision robots to become indispensable companions and everyday tools in business, education, entertainment, and public services much like the development of personal computers has revolutionized many domains of our lives. We believe that humans and robots will make great teams — not because of their similarities, but because of their differences that complement each other. Humans have creativity, intuition, emotions, and broad background knowledge that allows them to adapt to new environments and situations on a daily basis. Robots can process vast amounts of data, they are patient and persevering, and they can be designed to be strong, fast, and precise. Combining their strengths, humans and robots can unlock synergies to achieve higher productivity and higher standards of living. To leverage the full potential of this human-robot collaboration, humans and robots need to work closely together by sharing common workspaces, using the same facilities and tools, and communicating naturally and intuitively about tasks and observations. In this thesis, we present novel approaches for reaching these goals for successful human-robot collaboration in everyday navigation tasks.

In recent years, technology for mobile and humanoid robots has evolved rapidly: The availability of lightweight and energy-efficient RGB-D cameras has improved the robots' sensing capabilities. Combined with fast map representation techniques such as OctoMaps [75], a large toolbox of point cloud processing techniques collected in the open source Point Cloud Library (PCL) [147], and advances in 3D Simultaneous Localization and Mapping (SLAM) techniques, these sensors allow robots to map and navigate complex environments. Advances in neural networks and deep learning techniques [137] together with the increasing availability of dedicated parallel computing devices have opened up new opportunities for computer vision and scene understanding that allow a robot to act in domestic environments. Progress in the area of natural language processing and generation allows for a more natural interaction and communication between humans and robots. With the Robot Operating System (ROS), these advances in technology have been made accessible for research teams world-wide, fostering development in the whole field of robotics.

(a) Cosero (b) Nao cleaning up (c) Pepper tour bot

Figure 1.1: Robots acting in human environments. (a): University of Bonn's Cosero opening a fridge to investigate its contents (source: [169]). (b): Our Nao robot cleaning up a children's room as part of the SQUIRREL project (source: [158]). (c): Pepper robot providing an interactive tour of a lab at Queensland University (source: [166]).

This progress in developing general-purpose robots for daily applications is particularly visible in the RoboCup@Home championship. The RoboCup@Home championship puts emphasis on two aspects: First, the scenarios for the challenges are set up in everyday environments such as homes, offices, supermarkets, or restaurants that are unknown to the robots and the developers before the competition. In contrast to the RoboCup Soccer championship or industrial applications, RoboCup@Home does not abstract the environments or tailor the environments to the needs of robots, for example by adding fiducials. As "uncertainty is part of the concept" [15], the robots have to deal with human environments that they have to understand and manipulate.

Second, large parts of the RoboCup@Home challenge focus on human-robot interaction, for example the *cocktail party* task where a robot has to serve drinks to previously unknown people, or the *help me carry* task where a robot has to cooperate with a human operator by following the person and executing voice commands. Natural interaction is key for the success of domestic robots, as untrained persons have to be able to work with and command the robots without having any programming skills.

In this thesis, we explore how humans and robots can collaborate to solve navigation tasks in human environments by leveraging synergies in four different scenarios. In the first application (Chapter 2), we investigate how humans and automated computer systems can join forces to create better route descriptions. Wayfinding is a task that we face every day, for example when we travel to an unknown city or when we describe how to reach the train station to a foreign visitor. While we already often rely on technical devices such as GPS navigation devices or web-based routing services, these automated systems do not speak our language: Automated routing services typically produce phrases like "Turn left after 250 m" even though metric

distances are rather difficult to estimate for humans, making the route descriptions hard to follow. If you ask a random passer-by how to get to the train station, they will rather refer to salient landmarks that they choose depending on the recipient: Which landmarks catch someone's attention is different for children and adults, locals and non-locals, or blind and sighted people. Hence, we propose to combine the strengths of humans and automated systems: Humans know the target audience and have an intuition about what types of information are most useful for the person receiving the description, but on the other hand, they frequently make mistakes such as confusing left and right, miscounting intersections, and inaccurately estimating distances. Computer systems, by contrast, have access to a large amount of information such as an accurate, detailed map, but they are bad at judging what information is "ergonomic" for the recipient. The core idea of our approach is that automated systems should learn from humans how to describe routes: From a corpus of descriptions written by humans for a specific audience, our algorithm learns how much and what type of information it should include in a route description. Afterwards, the algorithm uses the learned policy to create new route descriptions that are correct and complete according to the map data, but also easy to understand and useful for the targeted audience.

In this first application, we investigate how automated systems can communicate navigation strategies to humans in a natural way. In the second application (Chapter 3), we consider the opposite scenario: How can humans effectively communicate background knowledge to a robot to help the robot better navigate in an unknown environment? The prerequisite for robot navigation is an accurate representation of the environment. In most scenarios considered in the literature, robots start with zero knowledge about the environment and use a SLAM approach to build an environment model from scratch following an exploration strategy that determines which areas will be explored next. If the robot does not have access to any background information, the decision on which area to explore next can lead to an arbitrarily inefficient exploration behavior. In many cases, however, some background information about the environment would be available, for example a sketch drawn by a human operator or a floor plan showing the basic building layout. We propose an algorithm to make background information from sources like these available to the robot and investigate how the robot can exploit this information to optimize the exploration tour, leading to shorter exploration times. We show an application where our algorithm uses data generated from archeological sketch maps to compute efficient navigation strategies for exploring a historic Roman catacomb.

While Chapters 2 and 3 target the communication aspect of navigating in human environments, the remaining chapters focus on the question of how robots can navigate in environments that are tailored

to humans. As we work towards universally usable robots working in everyday environments, these environments can hardly be modified to suit the needs of robots. In contrast to industrial environments, for example, it would be impractical to fit homes, shops, and even whole cities with induction loops or reflective fiducials that robots can use to navigate. Hence, robots have to use the environment itself to localize and navigate. In continuation of our previous work on robot navigation based on visual floor features [126], recognizing staircases in laser scans [128], and fine-positioning on staircases based on edge features [127, 129], we present a new feature descriptor for RGB-D data in Chapter 4 and optimize an existing appearance-based SLAM system for the usage in humanoid robots. Robust localization and mapping is a prerequisite for robots that execute tasks for humans in their homes.

One example for such a task is to search for an object in a known environment, for example to look for the car keys in an apartment. In Chapters 5 and 6, we consider the task of covering a known environment with a robot's camera, for example for mapping, inspection, or search tasks in user-defined regions of the environment. We extend an approach that Dornhege et al. [40, 41] originally developed for locating trapped persons in disaster rescue scenarios to make the technique usable on humanoid robots. In contrast to wheeled or tracked platforms, humanoid robots are better suited for human environments because they have a similar body plan as humans and can thus better use facilities and tools designed for humans. They can, for example, climb stairs, bend over boxes, crouch down to peek under objects, or open doors to see inside cupboards. The higher dexterity and degrees of freedom, however, come with higher planning complexity. In particular, bipedal robots have to maintain their balance while moving, introducing many constraints in the planning space, making sampling-based planning techniques as used in Dornhege's original approach infeasible. Hence, we introduce inverse reachability maps into the framework which allows us to quickly enumerate and verify stable body poses already in the camera viewpoint planning phase so that our algorithm generates coverage plans that are feasible, flexible, and efficient in static scenes.

As we envision our robots to help us with daily chores, the robots will also have to manipulate *articulated* objects, for example open drawers and cupboards or moving obstacles aside. Hence, we introduce articulated objects into our coverage planning scenario in Chapter 6. In addition to the high-dimensional planning problem of planning full-body poses, the robot now also has to decide in which order to manipulate multiple objects, for example when opening drawers and doors of a kitchenette that might occlude each other, increasing the planning complexity further. In order to be able to determine regions visible from the robot's perspective for a substantial amount of articulation configurations, we formulate the path planning and visibility

determination subproblems as rendering problems and use Graphics Processing Unit (GPU) acceleration to speed up the computation. In contrast to existing approaches that use dedicated high-end computing devices, we only use features available in the embedded systems subset of the Open Graphics Library (OpenGL) standard and show that the algorithm runs efficiently on a smartphone. In combination with a novel heuristic for estimating utility maps, our algorithm generates efficient plans for covering a known environment, which includes manipulating objects to gain access to hidden regions.

In Chapter 7, we summarize the results of this thesis and discuss open questions for future research.

## 1.1 KEY CONTRIBUTIONS

In this thesis, we investigate several aspects of human-robot collaboration and robot navigation in environments designed for humans. In summary, our key scientific contributions are:

- an inverse reinforcement learning approach for learning how to describe routes from a corpus of human-written route descriptions (Chapter 2),
- a planning approach that exploits background information given by the user to create an efficient exploration strategy for exploring unknown environments (Chapter 3),
- a feature descriptor that combines RGB and depth information for the usage in appearance-based SLAM systems on humanoid robots (Chapter 4),
- an approach that combines an existing next-best-view approach for planning a coverage tour through a known environment with inverse reachability maps for generating full-body poses for a humanoid robot (Chapter 5), and
- a heuristic and techniques leveraging GPUs to compute utility maps for covering known environments that contain also articulated objects (Chapter 6).

## 1.2 PUBLICATIONS

Parts of this thesis have been published in international journals and conference proceedings. The following list gives an overview about the individual publications.

- Chapter 2:
  S. Oßwald, H. Kretzschmar, W. Burgard, and C. Stachniss. "Learning to give route directions from human demonstrations." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2014, pp. 3303–3308.
  DOI: 10.1109/ICRA.2014.6907334.

  Best Cognitive Robotics Paper–Finalist.

- Chapter 3:
  S. Oßwald, M. Bennewitz, W. Burgard, and C. Stachniss. "Speeding-up robot exploration by exploiting background information." In: *IEEE Robotics and Automation Letters (RA-L)* 1.2 (2016), pp. 716–723. ISSN: 2377-3766.
  DOI: 10.1109/LRA.2016.2520560.

- Chapter 4:
  R. Sheikh, S. Oßwald, and M. Bennewitz. "A combined RGB and depth descriptor for SLAM with humanoids." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. In press. 2018.

- Chapter 5:
  S. Oßwald, P. Karkowski, and M. Bennewitz. "Efficient coverage of 3D environments with humanoid robots using inverse reachability maps." In: *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*. 2017, pp. 151–157.
  DOI: 10.1109/humanoids.2017.8239550.

- Chapter 6:
  S. Oßwald and M. Bennewitz. "GPU-accelerated next-best-view coverage of articulated scenes." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. In press. 2018.

The following publications were written while employed as a research assistant, but are not covered by this thesis:

- S. Oßwald, A. Hornung, and M. Bennewitz. "Improved proposals for highly accurate localization using range and vision data." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. 2012, pp. 1809–1814.
  DOI: 10.1109/IROS.2012.6385657.

- A. Hornung, S. Oßwald, D. Maier, and M. Bennewitz. "Monte Carlo localization for humanoid robot navigation in complex indoor environments." In: *Int. Journal of Humanoid Robotics* 11.2 (2014).
  DOI: 10.1142/S0219843614410023.

- P. Regier, S. Oßwald, P. Karkowski, and M. Bennewitz. "Foresighted navigation through cluttered environments." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. 2016, pp. 1437–1442. DOI: 10.1109/IROS.2016.7759234.
- P. Karkowski, S. Oßwald, and M. Bennewitz. "Real-time footstep planning in 3D environments." In: *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*. 2016, pp. 69–74. DOI: 10.1109/HUMANOIDS.2016.7803256.

## 1.3 COLLABORATIONS

Parts of this thesis are the results of collaborative work with other researchers, hence we consistently use "we" within this thesis. The collaborations on the work in this thesis and my contributions are as follows:

- Chapter 2: The system for learning to give route directions was developed in collaboration with Henrik Kretzschmar and supervised by Cyrill Stachniss and Wolfram Burgard. I designed and implemented the system. Henrik Kretzschmar contributed knowledge and experience about the general inverse reinforcement learning approach. I designed, executed, and evaluated the experiments jointly with Henrik Kretzschmar.
- Chapter 3: The development of the system for robot exploration with background knowledge was supervised by Cyrill Stachniss and data for the experiments has been provided by the ROVINA project.
- Chapter 4: The combined depth and RGB descriptor was originally developed within the Master's thesis of Rasha Sheikh. The design, implementation, and evaluation of the system was done entirely by Rasha Sheikh. I developed the project idea, supervised and advised the work on her thesis. We jointly executed the real-world experiments.
- Chapter 5: The system for covering 3D environments with humanoid robots was developed as an extension of Christian Dornhege's framework [41] in collaboration with Philipp Karkowski and was supervised by Maren Bennewitz. Philipp Karkowski contributed the design and implementation of the method for systematic sampling of rays described in Section 5.6.1.
- Chapter 6: The work on GPU-accelerated coverage tour planning was supervised by Maren Bennewitz.

## 1.4 NOTATION

We will use the following notation throughout this work:

| Symbol | Description |
|---|---|
| $a, b, \ldots$ | scalar value |
| $\mathbf{x}, \mathbf{y}, \ldots$ | vector |
| $A, B, \ldots$ | matrix |
| $\mathbf{x}^T, \mathbf{A}^T$ | transpose of a vector or a matrix |
| $\|x\|$ | norm (length) of a vector |
| $\det(A)$ | determinant of a matrix |
| $[a, b]$ | continuous range including $a$ and $b$ |
| $S = \{s_1, s_2, \ldots\}$ | set |
| $\|S\|$ | cardinality (number of elements) of a set |
| $S \setminus T$ | set subtraction |
| $x \leftarrow a$ | assignment of value $a$ to a variable $x$ |
| $p(x)$ | probability distribution of a random variable $x$ |
| $p(x \mid y)$ | conditional probability of $x$ given $y$ |
| $\mathbb{E}[X]$ | expectation value of random variable $X$ |
| $\mathbb{1}_{a=b}$ | indicator function, returns 1 if $a = b$ and 0 otherwise |
| $\tilde{a}$ | empirical value from training data |
| $a \vee b$ | logical disjunction (OR operation) |
| $a \wedge b$ | logical conjunction (AND operation) |
| $(\phi, \theta, \psi)$ | Euler angles roll, pitch, yaw |

# 2

# LEARNING TO GIVE ROUTE DIRECTIONS FROM HUMAN DEMONSTRATIONS

*For several applications, robots and other computer systems must provide route descriptions to humans. These descriptions should be natural and intuitive for the human users. In this chapter, we present an algorithm that learns how to provide good route descriptions from a corpus of human-written directions. Using inverse reinforcement learning, our algorithm learns how to select the information for the description depending on the context of the route segment. The algorithm then uses the learned policy to generate directions that imitate the style of the descriptions provided by humans, thus taking into account personal as well as cultural preferences and special requirements of the particular user group providing the learning demonstrations. We evaluate our approach in a user study and show that the directions generated by our policy sound similar to human-given directions and substantially more natural than directions provided by commercial web services.*

## 2.1 INTRODUCTION

Providing and following route directions is a task we face on a daily basis, for example when asking staff members how to get to the right aisle in a library or shopping center, or when moving in an unknown city following instructions provided by a navigation device. When driving cars, we already rely on web services such as Google Maps [61] and satellite navigation systems that provide turn-by-turn instructions at every decision point. For some applications, however, giving turn-by-turn instructions in real-time is impractical or impossible, for example in indoor navigation where global positioning is not available. In such applications, computer systems should be able to give spoken directions of the complete route in advance. This is relevant for information kiosks in public buildings, tour guide robots in museums, or navigation devices for visually impaired persons who are unable to read maps.

Existing commercial web services typically follow rigid patterns for giving directions, usually referring to the metrical distance to the forthcoming turning and the name of the street that the user has to take, e.g., "Turn left after 267 m onto Market Street." While these descriptions are technically correct, precise, and complete, they are still hard to follow for human users and they do not resemble the directions a human would give. Instead of providing precise metric

information, humans usually refer to salient landmarks along the route [177], trade off the amount of information that the recipient requires against the memory load caused by long descriptions, and take into account personal and cultural preferences when giving route directions [77].

The goal of this work is to develop a system that gives route directions in spoken or written form that humans can naturally and intuitively understand. We propose an approach that learns how to generate route descriptions from a corpus of human-generated descriptions. In this way, the user can train our system to use a particular style of instructions that is appropriate for the use case at hand. One of the key challenges is to choose the right amount and type of information that a route description should contain. What constitutes a "good" description depends on the user and the situation. Street names, for instance, are hardly of use for visually impaired, blind, or illiterate recipients as they cannot read the corresponding street signs. Street names are also challenging for foreigners who cannot decipher street signs written in Cyrillic, Arabic, or other scripts unknown to them. For blind recipients, tactile pavements and audible cues are much better suited description elements. By imitating the style of existing descriptions written for a particular user group, our system can learn to choose the type of information that best suits the needs of the users.

The person giving the directions also has to trade off the amount of information to give to the recipient. More detailed or even redundant descriptions may help the user to resolve ambiguities and recover from errors, but on the other hand require more memory and are thus harder to remember. As mentioned above, Hund et al. [77] found in a user study that cultural backgrounds influence the style of route directions. Participants from the United States who are accustomed to the typical grid-like layout of US cities with numbered streets would be more aware of street names and cardinal directions and use these elements when giving directions. Participants from the Netherlands, by contrast, would use these elements less, as the cities they are familiar with are typically irregular without any systematic street name scheme. They would instead refer to landmarks such as rivers, bridges, railroad crossings, and salient buildings.

Many other factors influence the decision of which ingredients to choose for composing a route description, such as personal preferences, the environment, familiarity of the recipient with the city, available tools such as sketches or maps, and the situation in which the description is given. Designing a rule-based expert system that takes all these factors into account is infeasible. Hence, we propose an adaptive system that can be trained to generate route descriptions tailored to a particular situation and audience by imitating the style of descriptions of a corpus of human-written descriptions.

The contribution of this work is a novel approach to learn a model of human-like directions for routes. We formulate the problem of giving directions as a Markov decision process. Hence, we assume that humans seek to optimize an unknown reward function when giving directions. We apply maximum entropy inverse reinforcement learning to infer the reward function from a set of directions provided by humans. The reward function is expressed in terms of features that capture relevant properties of the directions such as the amount and type of information given in the individual instructions. We collected a corpus of directions in a user study, trained our system on that corpus, and applied the learned policy to new routing description tasks. In a second user study, we asked human subjects to rate the descriptions generated by our system with respect to how natural they sound. The results suggest that the directions generated by our approach are significantly more human-like than the directions provided by existing route description services.

## 2.2 RELATED WORK

Studies in psychology, cognition, geo sciences, linguistics, and computer science have investigated the principles underlying the process of giving route descriptions. The work by Allen [6] and Lovelace et al. [100] identify basic structures and common characteristics of good route descriptions. For instance, humans typically give directions in linear order along the projected route, focus on critical choice points along the route such as intersections where the user has to turn, and increase the amount of information towards the end of the route.

In his dissertation [144], Kai-Florian Richter presents a detailed study on the differences and similarities between human-written and automatically generated route directions, mainly focusing on cognitive, linguistic, and representation-theoretic aspects. He identifies and systematizes commonly used components of route descriptions such as path representation, reference systems, roles of landmarks, chunking of subsequent instructions, and levels of granularity when giving directions. Based on these components, Richter formulates the task of giving route descriptions as an optimization problem that has to be solved by partially or exhaustively generating multiple possible route descriptions and choosing the best one according to an optimization criterion. As the main optimization criterion, Richter opts for minimizing the number of distinct blocks of information conveyed to the recipient of the description, arguing that shorter descriptions are easier to memorize and to process. In our work, we use similar components to describe routes. Most notably, we use the same mechanisms for chunking multiple subsequent instructions into a single instruction. We also formulate the problem as an optimization problem, but we choose a Markov decision process as a framework allowing to opti-

mize more general optimization criteria. While the conciseness of the description is an important criterion, we believe that other aspects must be taken into account as well. The shortest description is not necessarily the best description, as redundant information might help to disambiguate situations or prevent errors, for example when a landmark is temporarily obstructed by a construction site. By introducing a learning and imitation component, our approach is more flexible and can better adapt to the needs of the users compared to a hand-crafted optimization criterion.

Landmarks are generally considered an essential ingredient of good route descriptions and there are many studies investigating their roles and properties in verbal descriptions, e.g. [148, 177, 181]. Landmarks can be used to negotiate the initial heading at the start of the route, as reference points along the route, as distant "lighthouse" landmarks helping the user to maintain their global sense of orientation, or as a hint that the user has overshot the goal ("When you reach the church, you've gone too far"). However, there is no clear consensus on how to select appropriate landmarks for including them into route descriptions. The saliency of a category of landmarks is often a largely subjective measure depending on a person's interests, physical abilities, age, experience, and familiarity with the environment. The same also holds for other components of route descriptions. Hund et al. [77] showed in a user study that participants from the US are more likely to use cardinal directions when describing routes compared to participants from the Netherlands, illustrating that cultural backgrounds influence the style of giving route descriptions. Ward et al. [180] found evidence for gender differences, as male participants of their study were more inclined to use cardinal directions and metric distances compared to females and that this difference is a stylistic preference rather than due to spatial representation ability. Lovelace et al. [100] showed that familiarity with the route affects the types of landmarks that a speaker chooses and that the choice of landmarks likely changes when the speaker gets more familiar with a route. These examples illustrate that there is a large number of influences on how humans describe routes, and it is obviously impossible to model all these influences explicitly in an automated system. Hence, we instead propose a learning algorithm that adapts the style of giving route directions to the preferences of the particular user group by imitating humans. This allows us to exploit the knowledge and intuition of humans on how to describe routes to a particular audience for creating similar descriptions with an automated system.

Several research groups investigated how natural language processing can be used to understand and follow route directions [88, 103], which can be seen as the inverse problem to generating directions. The main challenge these works address is the problem of *grounding* natural language, that means the association of a word or phrase to

a physical object (e.g., a landmark) or a spatial concept (e.g., a direction). A system that learns to describe routes by imitating existing descriptions also needs to execute such a grounding step. However, as natural language processing and interpretation is a difficult problem on its own that is beyond the scope of our work, we instead use semi-automated assignment by generating a set of similar phrases matching the corresponding route segment and have a human operator select the most similar one.

Look [99] implemented a system for generating overview descriptions consisting of subgoals and neighborhoods along the route that complement and structure turn-by-turn instructions according to cognitive principles. Dale et al. [36] manually analyzed references and structures in a corpus of human-written directions and derived a set of rules for generating route descriptions that appear to be more human-like. In our work, we follow a similar approach but attempt to learn the user preferences directly from the human-generated corpus instead of manually inferring a set of rules.

Several algorithms have been proposed for altering the route in order to generate better descriptions, for example, for finding the most reliable path [67], or the simplest instructions [145]. Goeddel et al. [59] propose to use a particle filtering technique for maximizing the chances for arriving at the goal despite possible errors the user makes while traveling. Such approaches optimizing the route can be combined with our approach in a two-step process to create a system that optimizes both the route and its description.

Our algorithm learns policies that cater to specific user groups with particular needs and preferences. The generated descriptions are not necessarily optimal with respect to efficiency, reliability, simplicity, or similar metrics, as descriptions given by humans are not optimal according to these metrics either. We do not intend to optimize for such metrics, as we imitate the style of a corpus of human-written descriptions assuming that the corpus has been tailored to the needs and preferences of a particular audience.

Cuayáhuitl et al. [33] propose a hierarchical reinforcement learning approach for choosing a route and generating suitable directions along that route. The learned policy minimizes the number of instructions necessary to describe the route and the "user confusion" based on hand-crafted reward functions and transition models that consider the user's familiarity with the environment, the distance between subsequent decision points, and the saliency of the landmarks that the directions refer to. In contrast to that, our approach learns how much information and which pieces of information the route description should include by imitating descriptions given by humans without the need for estimating parameters of the underlying model, such as the "user confusion" probabilities.

Inverse reinforcement learning techniques have been used to address a variety of imitation learning problems including autonomous helicopter aerobatics [2], learning pedestrian navigation behavior [92, 191], and learning the preferences of cab drivers [190]. In particular, Abbeel and Ng [1] suggest to match features that capture relevant aspects of the behavior that is to be imitated. However, feature matching does not lead to a unique cost function. To resolve this ambiguity, maximum entropy inverse reinforcement learning [189] relies on the principle of maximum entropy [80] and, hence, aims to find the policy with the highest entropy subject to feature matching. In this work, we apply maximum entropy inverse reinforcement learning to learn a model of "human-like" route descriptions from a set of directions given by humans. Our approach is able to imitate their style of route directions in order to generate route descriptions that appear natural and intuitive to the recipients.

## 2.3 LEARNING TO GIVE DIRECTIONS FOR ROUTES FROM HUMAN DEMONSTRATIONS

The objective of this work is to learn a model of natural-language route directions from human demonstrations. We model the problem of giving directions for routes as a reinforcement learning problem in terms of a Markov Decision Process (MDP) [16]. We apply inverse reinforcement learning to elicit the unknown reward function from a set of directions provided by humans. The reward function captures relevant characteristics of the route directions and allows us to apply reinforcement learning to generate directions for new routes that mimic the human demonstrations.

### 2.3.1 *Giving directions as a reinforcement learning problem*

A MDP is commonly represented as a tuple $(S, A, P, r, P_0)$, where $s \in S$ is a set of states and $a \in A$ is a set of actions that allow to transition between the states. In the general case, actions do not necessarily have to be deterministic and a probability distribution $P(s' \mid s, a)$ specifies the probability that the system transitions from state $s$ to state $s'$ when executing action $a$. The reward function $r : S \times A \times S \to \mathbb{R}$ specifies an immediate reward for transitioning from state $s$ to state $s'$ by executing action $a$. Finally, $P_0$ is the probability distribution over the start states. A trajectory $\tau = \{s_{1:T}, a_{1:T}\}$ is a sequence of states and actions that adheres to the state transition probabilities. A policy $\pi(a \mid s)$ is given by a probability distribution over actions $a$ to take when in state $s$. The goal of reinforcement learning is to compute a policy $\pi(a \mid s)$ that maximizes the expected cumulative reward

$$R = \sum_{i=1}^{\infty} \gamma^t r(s_i, a_i, s_{i+1}) \tag{2.1}$$

where $\gamma$ is a discount factor trading off between current and future rewards.

MDPs are a commonly used representation of learning problems where costs and rewards can be defined for individual actions or for reaching a certain state, and the goal of the learning approach is to discover strategies for minimizing costs and maximizing rewards. In learning to play soccer, for example, scoring a goal leads to an immediate reward and moving on the soccer field comes with costs for decreasing stamina. Scoring a goal, however, is only possible through a chain of events and decisions and it is not obvious what benefit a particular action will bring in the long run. Through experience, though, a human player or an algorithm can learn to estimate which actions are the most promising ones in a particular situation towards the long-term objective of scoring a goal.

In our problem of learning how to describe routes, however, it is not clear how to define immediate or long-term rewards, as it is difficult to formalize when descriptions are "good" or "bad", in particular as this depends on many factors such as the familiarity of the recipient with the environment, whether or not the recipient has to memorize the description, which landmarks the recipient would consider as salient, etc. While it would be possible to define rewards for the effectiveness of the description, i.e., whether or not the recipient reaches the destination when following the description, such a definition would be impractical in a learning framework as learning algorithms need a large amount of training data, so a large number of user studies where a human actually navigates to a destination would be required, which is too time-consuming and costly in practice.

We assume that humans still optimize a reward function, consciously or not, as they know by intuition or by experience what information would be useful for the recipient to solve the task and they try to optimize the description to match the recipient's needs. It is also easy to collect a body of sample descriptions written by humans, for example by soliciting descriptions from participants of a user study, by collecting freely available descriptions on the Internet, or by using model descriptions from foreign language learning textbooks. Hence, the problem of learning to describe routes can be formulated as an Inverse Reinforcement Learning (IRL) problem, which is the problem of recovering an unknown reward function given a set of demonstrations $\{\tau_1, \ldots, \tau_n\}$.

As illustrated in Figure 2.1, we model the environment as a graph $G = (V, E)$, where the nodes $v \in V$ represent intersections, and the edges $e \in E$ represent streets. The space of all directions that guide a user through the environment from their current location $v_1$ to a specific destination location $v_{\text{goal}}$ along a given route is encoded by a deterministic MDP $M$. In particular, each action $a$ of the MDP corresponds to a single instruction that guides the user from node $v_i$

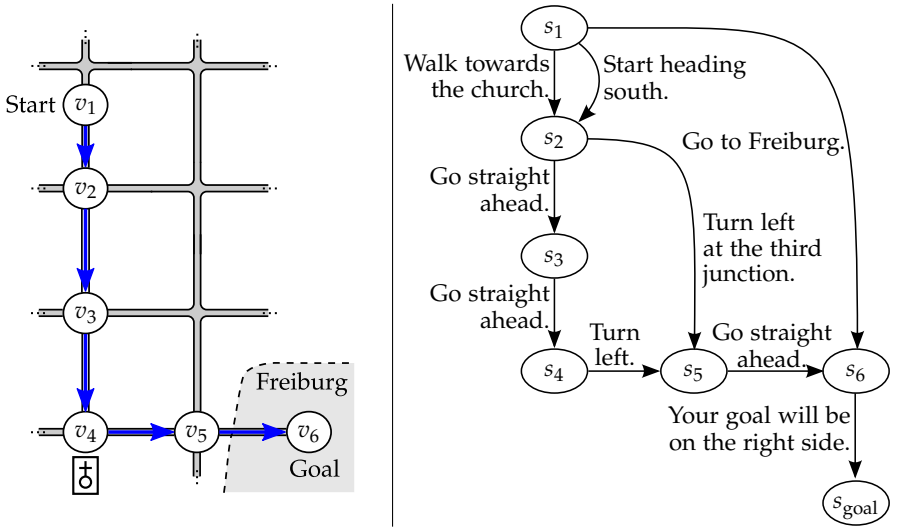



Figure 2.1: The problem of giving route directions as a reinforcement learning problem. Left: The route to be described (blue). Right: The corresponding Markov decision process encodes all valid descriptions of the route.

to node $v_j$. Consequently, each trajectory $\tau$ in the MDP $M$ corresponds to valid directions that guide the user all the way from its current location to its target location. For example, the descriptions "Walk towards the church. Go straight ahead. Go straight ahead. Turn left." and "Start heading south. Turn left at the third junction" are both complete, valid, unambiguous descriptions for guiding a user from the start node $v_1$ to $v_5$, but they have a different level of granularity and amount of information that the recipient has to memorize. Hence, every path through the MDP from the start node to the goal node represents a valid natural-language description of the *same* route, but in *different* words.

We assume that all actions are deterministic, meaning that we expect that the recipient correctly executes the instructions. In practice, this might not be the case, as the user might miscount intersections, confuse left and right, or there might be changes in the environment such as a new street that had not yet been registered in the map that the algorithm used to generate the description. However, it would be difficult to incorporate user mistakes, as the person or algorithm giving the directions can hardly foresee what error the recipient will make. Without knowing the error, it is not possible to give meaningful instructions for correcting the error in advance. This problem could be alleviated by altering the route to reduce the risk for errors as Goeddel et al. [59] propose, or by including "When you arrive at ..., you've gone too far" instructions, as Lovelace et al. [100] suggest. Both approaches, however, are computationally expensive and difficult to integrate into a common system.

The MDP defined here is acyclic, as returning to a previously visited state would be redundant. As there are no cycles and each action advances the user's state towards the goal, the MDP has a finite horizon, thus discounting factors are not necessary and $\gamma$ can be set to 1. These properties are desirable as they make the learning algorithm more efficient.

### 2.3.2 *Maximum entropy inverse reinforcement learning*

If the reward function of an MDP is known, reinforcement learning can be used to derive a policy that optimizes the expected cumulative reward. In our case, however, the reward function is inaccessible as we do not have an explicit representation of the goals that human route describers optimize for. Instead, we only have access to a body of demonstrations represented as a set of trajectories through the MDP and we assume that human subjects created these demonstrations by following an underlying unknown reward function. The goal of Inverse Reinforcement Learning (IRL) [119] is to determine a new reward function and a corresponding policy that can be transferred to new problems for creating trajectories with similar properties as the demonstrated trajectories.

The properties in which the newly created trajectories should be similar to the demonstrated are called *features*. A feature $f : (S, A)^T \to \mathbb{R}$ maps a trajectory $\tau = \{(s_1, a_1), \ldots, (s_T, a_T)\}$ to a real value. Examples for features are the length of the trajectory, frequencies of particular states or actions, or frequencies of repeating elements along the trajectory. All features combined form a *feature vector* $\mathbf{f}_\tau \in \mathbb{R}^n$ of a trajectory $\tau$.

The reward function $R$ decides on the distribution of the feature values when sampling new trajectories and we consider the distribution of these feature values to choose the reward function that best matches the demonstrations. Hence, we define $R$ as a function of the feature values parameterized by a parameter vector $\boldsymbol{\theta} \in \mathbb{R}^n$. From each possible reward function $R_{\boldsymbol{\theta}}$, a policy $\pi_{\boldsymbol{\theta}}$ can be derived using reinforcement learning. When this policy $\pi_{\boldsymbol{\theta}}$ is applied to an MDP, it induces a probability distribution $p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$ over all trajectories $\boldsymbol{\tau}$ leading through the MDP's graph. Given this probability distribution, the goal of IRL is to find suitable parameters $\boldsymbol{\theta}$ of the reward function so that the following necessary condition is satisfied:

$$\tilde{\mathbf{f}}_{\boldsymbol{\tau}} = \mathbb{E}_{p_\theta}[\mathbf{f}(\boldsymbol{\tau})]. \tag{2.2}$$

$\tilde{\mathbf{f}}_{\boldsymbol{\tau}}$ is the empirical feature vector summed up over all demonstrations $\boldsymbol{\tau}$, and $\mathbb{E}_{p_\theta}[\mathbf{f}(\boldsymbol{\tau})]$ is the expectation value of the feature vector over all demonstrations.

While this necessary condition guarantees that the learned policy reproduces the same feature value distribution when applied to the

training data, it does not lead to a unique solution. There is an infinite number of reward functions that produce the same feature distributions, including solutions that overfit to the training data or introduce preferences for certain policies that are not part of the training demonstrations and do not reflect the hidden true reward function. Hence, we follow the principle of maximum entropy [80] that suggests selecting the distribution that least favors any particular outcome while still satisfying the necessary constraint from Equation 2.2. Ziebart applied this maximum entropy principle to IRL, leading to the Maximum Entropy Inverse Reinforcement Learning (MaxEnt IRL) approach [189].

The MaxEnt IRL approach optimizes the objective function

$$p_{\boldsymbol{\theta}}^{\star} = \underset{p_{\boldsymbol{\theta}}}{\operatorname{argmax}} H_{p_{\boldsymbol{\theta}}(\tau)} \text{ such that } \tilde{\mathbf{f}}_{\boldsymbol{\tau}} = \mathbb{E}_{p_{\boldsymbol{\theta}}}[\mathbf{f}(\boldsymbol{\tau})], \tag{2.3}$$

where $H(p_{\boldsymbol{\theta}})$ is the Shannon entropy of distribution $p_{\boldsymbol{\theta}}$.

Solving this optimization problem in the general case is hard, as it requires to determine a probability distribution over the large space of all possible trajectories in the MDP. To make this problem tractable, we adopt the same assumptions and simplifications that Ziebart proposed. First, we transfer the Markov property of the MDP also to features, requiring that all features must be the sum of values assigned to individual states or actions and that the value assigned with a state must not depend on previous or future states. Instead of having to estimate the probability of each complete trajectory, this assumption allows us to estimate the visitation frequency $D_{\boldsymbol{\theta}}(s, a)$ of each state-action pair $(s, a)$ in the MDP and calculate the expected feature distribution as

$$\mathbb{E}_{p_{\boldsymbol{\theta}}}[\mathbf{f}(\boldsymbol{\tau})] = \sum_{(s_i, a_i) \in \tau} D_{\boldsymbol{\theta}}(s_i, a_i) \mathbf{f}(s_i, a_i). \tag{2.4}$$

If we also assume that the reward function is a linear combination of the feature values

$$R_{\boldsymbol{\theta}} = \sum_{(s_i, a_i) \in \tau} \boldsymbol{\theta}^T \mathbf{f}(s_i, a_i), \tag{2.5}$$

then the solution of Equation 2.3 is given according to Ziebart [189] as

$$p_{\boldsymbol{\theta}}^{\star}(\tau) \;\propto\; e^{-\boldsymbol{\theta}^T \mathbf{f}(\boldsymbol{\tau})}. \tag{2.6}$$

Our goal is now to maximize the likelihood $p_{\boldsymbol{\theta}}^{\star}(\tau)$ of the trajectories demonstrated by the user. To solve this problem, we apply gradient-based optimization on the logarithmic likelihood to iteratively refine the parameters $\boldsymbol{\theta}$ to achieve feature matching. The gradient with respect to the parameters $\boldsymbol{\theta}$ of the reward function is

$$\nabla_{\boldsymbol{\theta}} \ln p_{\boldsymbol{\theta}}^{\star}(\tau) = \tilde{\boldsymbol{f}}_{\boldsymbol{\tau}} - \mathbb{E}_{p_{\boldsymbol{\theta}}}[\mathbf{f}(\boldsymbol{\tau})] \tag{2.7}$$

The empirical feature count $\tilde{\boldsymbol{f}}_{\boldsymbol{\tau}}$ can be determined directly by summing up the features in the demonstrated trajectories.

To compute the expected feature count $\mathbb{E}_{p_\theta}[\mathbf{f}(\boldsymbol{\tau})]$, we use Ziebart's efficient dynamic programming algorithm [189] to compute the expected visitation frequency $D_{\boldsymbol{\theta}}(s_i, a_i)$ of each state-action pair $(s_i, a_i) \in \tau$. Algorithm 2.1 shows the original algorithm proposed by Ziebart. The algorithm has four stages: First, a backward pass propagates a probability mass $Z$ from the goal nodes towards the start nodes. This can be interpreted as a "flooding" the graph with mass injected at the goal nodes and transporting the mass along the transition edges in opposite direction one node at a time. Hence, this backward step has to be repeated $N$ times for a finite time horizon $N$. The second phase computes the policy $\pi\left(a_{i,j} \mid s_i\right)$ that specifies which action to take in each state. This policy is computed as the ratio of the probability masses $Z_{a_{i,j}}$ of the outgoing actions and the mass $Z_{s_i}$ of the state. In the third phase, a forward pass propagates the state visitation frequencies $D_{s_i}$ from the initial nodes towards the goal nodes according to the computed policy and transition probabilities. Again, this step has to be repeated for a finite time horizon. In the fourth step then accumulates the visitation frequencies of the individual time steps to compute the overall state frequencies.

As explained above, our MDP graph is acyclic with a finite horizon. Hence, we can reduce the computation time for the visitation frequencies even further by processing the states in topological order. If the states are processed in topological order, only a single sweep is necessary to propagate the visitation frequencies along the graph, so the visitation frequencies converge faster. Mathematically speaking, we define a partially ordered set $(S, \leq)$ over the set of states $S$ so that $s_i \leq s_j$ if there is a path from $s_i$ to $s_j$ in the directed graph representing the MDP. Propagating the probability masses in this order eliminates the need for repeating the propagation step as the probability masses will not change anymore after the first pass. Algorithm 2.2 shows our modified version of Ziebart's algorithm. In Ziebart's original formulation, rewards are attributed to states only, hence the algorithm computes state visitation frequencies only. As the policy is also explicitly computed, however, computing the visitation frequencies of state-action pairs can be done straightforwardly, allowing us to attribute rewards to any combination of state, action, and successor state of a transition (but not to other states and actions before or after the transition due to the Markov property).

### 2.3.3 *Features*

Inverse reinforcement learning uses feature vectors to characterize each possible solution of the task to be learned and tries to imitate the style of the given demonstrations by matching the feature value distribution. Hence, the choice and definition of features is critical for the learning task. In our scenario, the features represent properties

**Algorithm 2.1** Ziebart's algorithm for computing state visitation frequencies from [189]

*// Backward pass*

$Z_{s_{\text{terminal}}} \leftarrow 1$

for $N$ iterations compute recursively:

$\quad Z_{a_{i,j}} \leftarrow \sum_k P\left(s_k \mid s_i, a_{i,j}\right) e^{\text{reward}(s_i|\theta)} Z_{s_k}$

$\quad Z_{s_i} \leftarrow \sum_{a_{i,j}} Z_{a_{i,j}} + \mathbb{1}_{\{s_i = s_{\text{terminal}}\}}$

*// Local action probability computation*

$\pi_{\boldsymbol{\theta}}\left(a_{i,j} \mid s_i\right) \leftarrow \frac{Z_{a_{i,j}}}{Z_{s_i}}$

*// Forward pass*

$D_{s_i,t} \leftarrow P_0\left(s_i\right)$

for $N$ iterations recursively compute recursively:

$\quad D_{s_k,t+1} \leftarrow \sum_{s_i} \sum_{a_{i,j}} D_{s_k,t} \pi_{\boldsymbol{\theta}}\left(a_{i,j} \mid s_i\right) P\left(s_k \mid a_{i,j}, s_i\right)$

*// Summing frequencies*

$D_{s_i} \leftarrow \sum_t D_{s_i,t}$

**Algorithm 2.2** Our modified algorithm for computing visitation frequencies of state-action pairs for acyclic finite-horizon MDPs

*// Pre-process graph*

Sort and re-index states so that $s_i \leq s_{i+1}$ according to topologic order of the MDP graph

*// Backward pass*

$Z_{s_{\text{terminal}}} \leftarrow 1$

for $i$ from $|S|$ to 1 do

$\quad Z_{a_{i,j}} \leftarrow \sum_k P\left(s_k \mid s_i, a_{i,j}\right) e^{\boldsymbol{\theta}^T f(s_i,a_i)} Z_{s_k}$

$\quad Z_{s_i} \leftarrow \sum_{a_{i,j}} Z_{a_{i,j}} + \mathbb{1}_{\{s_i = s_{\text{terminal}}\}}$

*// Local action probability computation*

$\pi_{\boldsymbol{\theta}}\left(a_{i,j} \mid s_i\right) \leftarrow \frac{Z_{a_{i,j}}}{Z_{s_i}}$

*// Forward pass*

$D_{s_i} \leftarrow P_0\left(s_i\right)$

for $i$ from 1 to $|S|$ do

$\quad D_{s_i} \leftarrow D_{s_i} + \sum_k \sum_{a_{k,i}} D_{s_k} \pi_{\boldsymbol{\theta}}\left(a_{k,i} \mid s_k\right) P\left(s_i \mid a_{k,i}, s_k\right)$

*// Calculate state-action pair frequency*

$D_{\boldsymbol{\theta}}\left(s_i, a_i\right) \leftarrow D_{s_i} \pi_{\boldsymbol{\theta}}\left(a_{i,j}, s_i\right)$

| Feature | Description |
|---|---|
| number of instructions | total number of instructions in the route description |
| number of slots | number of pieces of information an instruction contains (see Section 2.3.3) |
| abstraction level | turn-by-turn instruction, chunked instruction, or destination description |
| segment length | length of the described route segment in meters |
| number of intersections | number of intersections covered by the instruction |
| street name references | number of street names mentioned |
| street name saliency | saliency of the street name if mentioned according to the web search metric described in Section 2.4.1 |
| cross street references | number of crossing street names mentioned |
| cross street saliency | saliency of crossing street if mentioned |
| street category references | number of street categories mentioned (footpath, residential road, motorway, etc.) |
| cross street category ref. | number of crossing street categories mentioned |
| "head towards" ref. | number, saliency and distance of landmarks that indicate the heading, but are not part of the route, e.g. "head towards the ocean" |
| "head towards" saliency | |
| "head towards" distance | |
| landmark reference | number of landmarks mentioned |
| landmark distance | distance of landmark if mentioned |
| landmark direction | direction of landmark relative to recipient (back, side, ahead, etc.) |
| landmark saliency | saliency of landmark based on the number of hits returned from a web search for the landmark's name (see Section 2.4.1 for details) |
| turn anchor | turn at landmark / after metric distance / etc. |
| metric reference | discretized mileage mentioned, e.g. "turn right after 200 m" |
| counter reference | counter value mentioned, e.g. "turn right at the second intersection" |
| reference frame at turns | cardinal (e.g. "turn north"), allocentric (e.g. "turn left"), or using counter (e.g. "take the second exit in the roundabout") |
| reference frame at start | cardinal, relative to landmark, or using street name (e.g. "Start on Main St.") |
| "n times" chunking | e.g. "turn right twice" |

Table 2.1: Features characterizing route descriptions. See Section 2.3.3 for detailed explanations.

of the route directions such as the length of the description, the landmarks mentioned in the instructions, or the frame of reference used to indicate turn directions. Table 2.1 shows all features implemented in our system. They can be grouped into three categories:

AMOUNT OF INFORMATION The amount of information that the directions should include depends on the situation in which the user gives the directions to the recipients. For example, if the recipients will receive the directions as a printed text that they can take along while traveling, the descriptions can be more detailed compared to situations in which they have to memorize the whole description. We introduce two features to measure the amount of information on different levels: First, we count the total number of instructions. Second, we define a feature measuring the number of pieces of information within each instruction inspired by Mark's concept of *slots* [107]. For instance, the instruction "Turn right at the second intersection into Market Street" contains three pieces of information that the recipient has to remember.

ABSTRACTION LEVELS Instructions can occur at different granularity levels. At the lowest level, turn-by-turn instructions refer to each decision point individually (e.g. "Turn right"). Chunked instructions, by contrast, direct the recipient across multiple decision points (e.g. "Go straight ahead until you get to the church"), which is a more concise description, but may be more difficult to follow as the user has to count intersections or watch out for a particular landmark. At the highest level, destination descriptions just give an intermediate goal without specifying how to get there (e.g. "Drive to Freiburg"), assuming that the user either already knows how to get there, or can find the way by other means, for example by following road signs. We define features that count the frequencies of these abstraction levels as well as features measuring the number of decision points and the length of the route segment covered by an instruction.

DESCRIPTION STYLE Personal and cultural preferences also influence the description style, e.g., the person giving the directions can use cardinal directions, allocentric directions, or directions relative to landmarks to indicate in which direction the recipient has to travel. Furthermore, they can choose to include street names, street categories, mileages, and landmarks in the description. These elements can occur in several roles. Landmarks, for example, can be used to negotiate a common reference frame (e.g. "Start facing the cathedral"), mark a waypoint that the recipient has to pass, or indicate the heading without being part of the route (e.g. "Head towards the ocean"). Therefore, we define features that count the frequencies of these types of information.

| Context | Description |
|---|---|
| is at start | true if recipient is at start location of the route |
| is at goal | true if recipient is at goal location of the route |
| is dead end | true if recipient's position is in a dead end |
| is turn | true if recipient has to change direction |
| turn angle | discretized turn angle (straight ahead, 90° turn, slight turn, hard turn) |
| navigation complexity | complexity measure of the intersection (see Section 2.3.4) |
| landmark present | true if any suitable landmark is nearby |
| best landmark saliency | highest saliency of nearby landmarks |
| street name saliency | saliency of the current street name according to the web search metric described in Section 2.4.1 |
| street category | category of the current street (e.g. foot path, living road, residential road, tertiary road, primary road, motorway) |
| street category difference | difference between street category of current and next road segment $(-2, \ldots, +2)$ |

Table 2.2: Contexts characterizing the route segment and its environment. See Section 2.3.4 for detailed explanations.

For learning how to choose suitable landmarks, we define features of the saliency of the landmarks (see Figure 2.4.1), the distance between the landmarks and the decision points, and the directions of the landmarks relative to the user.

### 2.3.4 *Contexts*

The description of a particular route segment typically depends on the environment of the route segment. For example, turning at complex intersections requires more information than simply going straight ahead. To integrate the context information into the learning process, we introduce contexts as functions $c : A \to K$ that map properties of the route segment covered by actions $A$ to discrete, finite sets $K$ (e.g. the set of discretized turn angles). In contrast to features, contexts only depend on the route and its surroundings, but not on the user's description.

For example, we calculate a complexity measure of each intersection based on the number and geometry of the streets meeting at that intersection, the direction in which the route continues, and the street names of the inbound and outbound segment of the route. Similarly, we define discretized measures for the turn direction angle, for the street category ranging from foot paths to motorways, and for the presence of landmarks near the decision point.

In line with Ziebart's approach for context-aware inverse reinforcement learning [190], we consider features $f^k$ that are only active in a particular context, i.e.,

$$\forall k \in K : f^k(a) := \begin{cases} f(a) & \text{if } c(a) = k \\ 0 & \text{otherwise.} \end{cases} \tag{2.8}$$

To determine which features depend on which contexts, we define for each feature $f$ the probability distribution $P(f)$ as the histogram of feature values observed during the demonstrations. We then compute for each feature and each context the mutual information (information gain)

$$MI(f; c) = H(f) - H(f \mid c), \tag{2.9}$$

where $H(\cdot)$ denotes the Shannon entropy, and $P(c)$ is the distribution of context values in the training data. The mutual information $MI(f; c)$ measures how informative the context is for determining the distribution of feature values. Following the approach commonly used in decision tree learning [142], we combine the features and contexts yielding the highest information gain. For instance, the "is turn" context provides a high information gain for the feature "street name references", as humans mention street names more often when turning than when going straight ahead.

## 2.4 EXPERIMENTAL EVALUATION

We conducted a two-part user study for evaluating our approach. In the first part, we collected a corpus of directions for a set of given routes from the participants. Using this corpus, we learned a reward function, which we then used to generate descriptions for a set of test routes. In the second part of the experiment, we presented human-written and computer-generated directions to other participants and asked them to rate how natural the descriptions sound to them.

### 2.4.1 *Acquiring training data*

In order to acquire training data for learning a route description policy, we asked 13 participants in a web-based survey to describe up to three routes within Freiburg. All participants were locals and fluent in English. We provided an unmodified, interactive OpenStreetMap map to the participants and asked them to describe the route marked on the map. Initially, the map showed only the first part of the route, forcing the participants to interact with the map by moving it and zooming in or out, and they were allowed to (but not limited to) use any information they found in the map. The instructions introducing

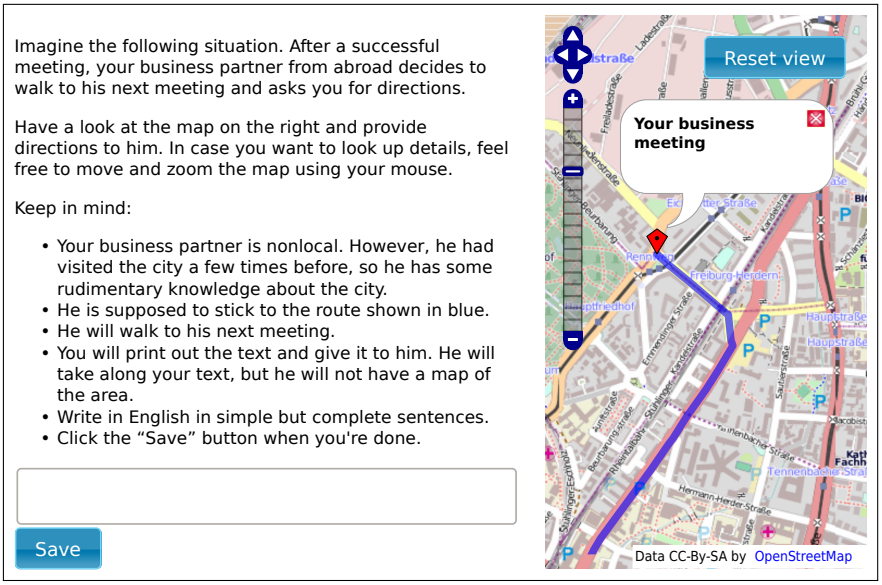



Figure 2.2: Screenshot of the first experiment for soliciting route descriptions from participants.

the experiment stated that the recipient of the description was a non-local business partner who had some rudimentary knowledge about the city. The recipient would print out the directions and take them along while walking, but would not have a map at his disposal. See Figure 2.2 for a screen capture of the experiment.

In this experiment setup, we collected a corpus of 28 descriptions for ten routes ranging from 0.6 km to 2.9 km in both urban and downtown environments.

For each route, our system generated natural-language instructions corresponding to the route segments based on OpenStreetMap data. All generated descriptions are valid in the sense that they correctly describe the navigation actions the recipient has to execute in order to reach the target. The corresponding MDPs used in the experiments contained between 11 and 50 nodes representing decision points and between 4,614 and 86,643 actions corresponding to single natural-language instructions.

In order to estimate the saliency of landmarks, we use a two-step metric. First, the algorithm counts the frequency of each category of landmarks in the human-written descriptions, yielding a saliency value for each landmark category. We then weight the individual saliency of each landmark within its group. For estimating landmark saliency, commonly used metrics rely on the visual appearance and semantic attraction of the landmark (e.g. [121]). Unfortunately, publicly available data sources such as OpenStreetMap do not provide the information these metrics require. Hence, our algorithm resorts to querying a web search engine to get a rough saliency estimate. For landmarks with unique proper names (e.g. *Eiffel Tower*), the al-

gorithm searches for the location and exact name of the landmark (e.g. *Paris "Eiffel Tower"*) and estimates the saliency of the landmark based on the relative number of hits. We argue that landmarks that are often mentioned on the web are likely to be better known and more salient. For unnamed landmarks and landmarks without unique names, such as branches of banks or fast-food chains, we estimated the individual saliency based on the density of landmarks of the same type in the vicinity, arguing that landmarks are only valuable if they are distinguishable and not ambiguous. If the map contained enough information on the shape and physical appearance of the landmarks, then the distinctiveness and the visibility of the landmarks from the user's view point could also be incorporated into the saliency measure.

As natural language processing is beyond the scope of this work, we manually matched each description in the collected corpus to the closest description generated by the MDP corresponding to the route. This step implicitly corrects obvious errors in the human-provided descriptions, such as left-right confusions or miscounted intersections, as the generating model always produces correct instructions. We also replaced descriptions of the appearance of landmarks with the name of the respective landmark as we only consider which landmarks to include in the directions, but not how to describe them in words. For example, we rewrite "you reach a square with a statue of a rider in the middle" as "you reach the Bertoldsbrunnen fountain." In future work, an additional step could be introduced in the text generation procedure that substitutes the names of the landmarks back to descriptions generated from additional sources, such as the Wikipedia entry of the landmark or information from tourist guide books.

We implemented our approach in Java using the *Traveling Salesman* framework [183] and accessing the OpenStreetMap database via its public API. Based on the corpus of route descriptions, our algorithm learned a weight vector. To generate a description for a novel route, the algorithm generates the corresponding MDP, computes a policy for the MDP using Ziebart's algorithm, and samples a path through the MDP according to the learned policy to generate a route description.

### 2.4.2 *Learned policies*

One of the advantages of our formulation of the reward function as a linear combination of context-aware features is that the learned coefficients can be interpreted and provide insight on the style of direction giving the participants used. Table 2.3 shows the reward weights that determine whether or not a street name is mentioned in an instruction depending on three of the contexts listed in Table 2.2. If the reward is higher, then it is more likely that a street name is mentioned in the given context. Most weights are negative, indicating that the participants used street names rather sparingly. Despite negative

| Context | Context value | Reward for using street name |
|---|---|---|
| turn angle | continue straight ahead | $-0.82$ |
| | slight turn | $-0.47$ |
| | 90° turn | $-0.10$ |
| navigation complexity | 1 (simple) | $-0.98$ |
| | 2 | $-0.53$ |
| | 3 | $-0.07$ |
| | 4 (difficult) | $-0.37$ |
| street category change | $-2$ (e.g. primary $\rightarrow$ residential road) | $+0.01$ |
| | $-1$ (e.g. residential $\rightarrow$ living road) | $-0.16$ |
| | $\pm 0$ (continue on same category) | $-0.28$ |
| | $+1$ (e.g. primary road $\rightarrow$ motorway) | $+0.09$ |
| | $+2$ (e.g. residential $\rightarrow$ primary road) | $+1.83$ |

Table 2.3: Learned reward weights for using street names in instructions depending on the context of the instruction.

weights, using street names can still occur in descriptions as there are also rewards for including a certain amount of information in the descriptions and other types of information may have higher penalties. The weights are high when the route follower has to turn at an intersection, when the navigation complexity of the intersection is high, or when the street category changes considerably, for example when turning from a primary road into a small residential road, indicating that street names should be mentioned in these cases.

Other insights include that the participants in our study generally avoided cardinal directions for describing turns along the route, which is in line with studies showing that European participants generally avoid cardinal direction terms unless they are explicitly primed to do so (e.g. [77] for Dutch participants). However, some subjects resorted to cardinal direction terms for negotiating the start orientation of the recipient if there was no landmark nearby that they could refer to. The reward weights also reflect that complex intersections require more instructions, whereas the recipient needs less information at simpler intersections. Regarding the question of which landmarks to mention, the algorithm learned some correlations from the corpus as we expected, for example that humans prefer landmarks ahead of the user over landmarks in the recipient's back or to the side, that humans mention landmarks with higher saliency more often than landmarks with lower saliency, and that the importance of landmarks rapidly decays with increasing distance to the decision point. Counting intersections as in "Turn right at the third intersection." never appeared in the corpus with numbers higher than three, hence the resulting

weights are low for the features measuring the counter references higher than three.

The learned weight vector reflects these properties and provides distributions over the discretized features. However, the learned weights only match the style of the particular user group we asked to give the directions. We expect the weights to be different if other user groups provide the directions, e.g. visually impaired people or people from other cultural backgrounds.

### 2.4.3 *Experimental setup for evaluating the generated descriptions*

The goal of our work is to imitate the way humans give directions. A popular approach for evaluating how well a system imitates human behavior is to present test instances created by humans and instances created by a machine to human subjects in a blind study and have the participants predict whether the instance was created by a human or a machine. If the participants cannot identify the machine-created instances with a hit rate better than chance, then the imitation would be perfect. This approach, however, would contravene our intended use case: In order to disguise the fact that a route description was written by a machine, the algorithm would have to deliberately add errors to make up for the errors that are quite frequent in human-written descriptions. A typical type of error is left-right confusion, as around 20 % of the human population report that they frequently confuse left and right [65]. While adding errors would make the descriptions more human-like, it would defeat the purpose of helping the recipient with a wayfinding task. Hence, we designed our experiment for evaluating how well the learned policy imitates the direction given by humans by asking the participants how "natural" the descriptions sound without cueing them to look for hints on whether the description was written by a human or a machine.

As the screen capture of the experiment in Figure 2.3 shows, we provided the participants with a route marked on an interactive OpenStreetMap map together with either a human-written description gathered in the first experiment, a description generated automatically according to the learned policy, or a description generated by one of three popular web services (Google Maps [61], Bing [113], and MapQuest [106]). We did not inform the participants beforehand that the descriptions originate from different sources.

For the computer-generated descriptions, we used the same routes as in the first experiment to allow for direct comparison with the human-written descriptions, as well as four novel routes. We asked the participants to answer the question "How natural does this description sound to you?" by dragging a continuous slider between "sounds like a computer" and "sounds natural (human-like)." Each subject rated twelve descriptions in random order, each description corresponding

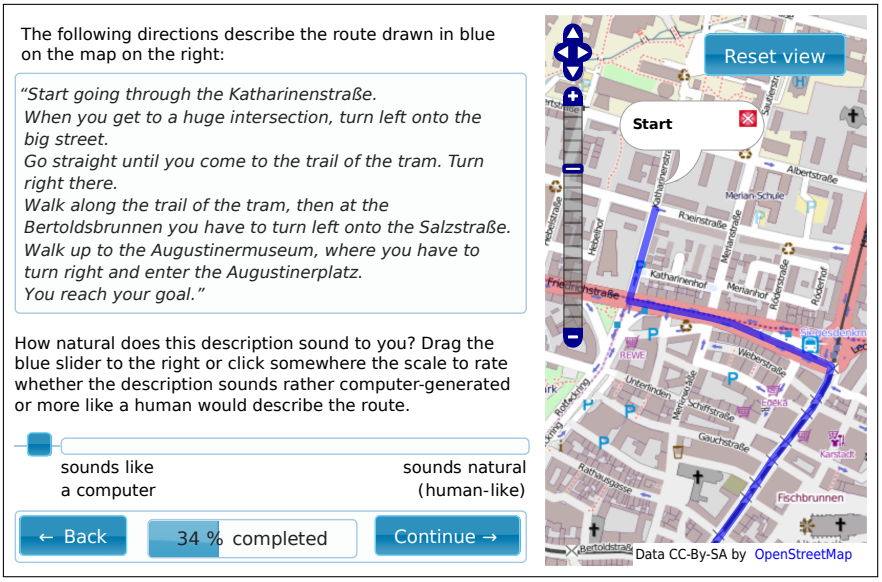



Figure 2.3: Screenshot of the second experiment for evaluating how natural directions sound.

to a different route. The subjects were allowed to go back to previous routes to revise their answer.

We processed the human-written descriptions according to the manual annotations that the algorithm used for learning the policy. We furthermore fixed typos and spelling errors. As a result of that, the regenerated descriptions have the same structure and contain the same information as the original input, but use the same formatting as the computer-generated instructions. Table 2.4 provides example descriptions for a given route.

### 2.4.4 *How human-like are the directions generated by our approach?*

Figure 2.4 shows the distribution of the ratings that the participants of the second experiment assigned to the routes and the corresponding box plots. The participants generally rated the descriptions generated by our approach to sound more natural than the descriptions provided by the commercial web services. The difference is statistically highly significant (t-test with 99.9 % significance).

Overall, the participants rated the generated descriptions and the human-written descriptions similarly. The mean of the slider values are slightly higher for the human-written descriptions (71.7) compared to the generated descriptions (69.5), but the difference is not statistically significant. Both distributions have high variances, reflecting that there is no clear consensus among the participants what constitutes "natural" instructions.

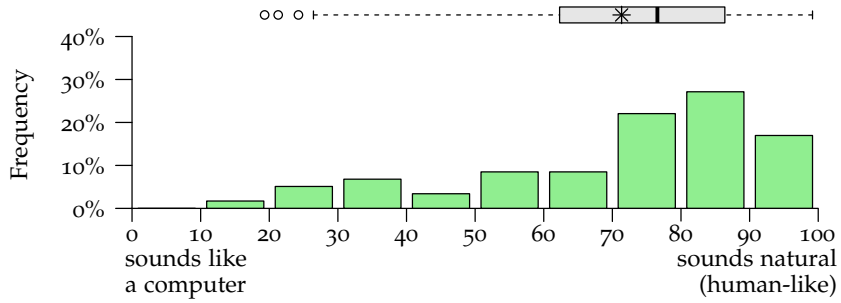
We furthermore analyzed the number of training instances our system requires. Figure 2.5 suggests that a low number of training

| Human-provided description |
| --- |
| Start going through the Katharinenstraße. When you get to a huge intersection, turn left onto the big street. Go straight until you come to the trail of the tram. Turn right there. Walk along the trail of the tram, then at the Bertoldsbrunnen you have to turn left onto the Salzstraße. Walk up to the Augustinermuseum, where you have to turn right and enter the Augustinerplatz. |

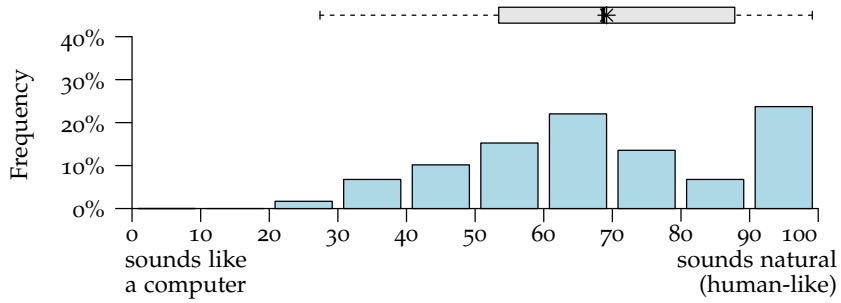
| Generated by our algorithm |
| --- |
| On Katharinenstraße start heading towards the Altstadt. Go along the street until you encounter a large intersection. Turn left onto the Friedrichstraße, and pass the Vapiano on your right. Go ahead until you come to the tram track. Turn right there onto the Kaiser-Joseph-Straße. Turn left after the Drogerie Müller. Turn right at the second possibility and enter the Augustinerplatz. You are at your target when you get there. |

| Google Maps description (similar result for Bing and MapQuest) |
| --- |
| Head north on Katharinenstraße toward Rheinstraße. Turn right onto Rheinstraße. Turn right onto Merianstraße. Continue onto Rathausplatz. Continue onto Universitätsstraße. Turn left onto Bertoldstraße. Continue onto Salzstraße. Turn right onto Augustinerplatz. |

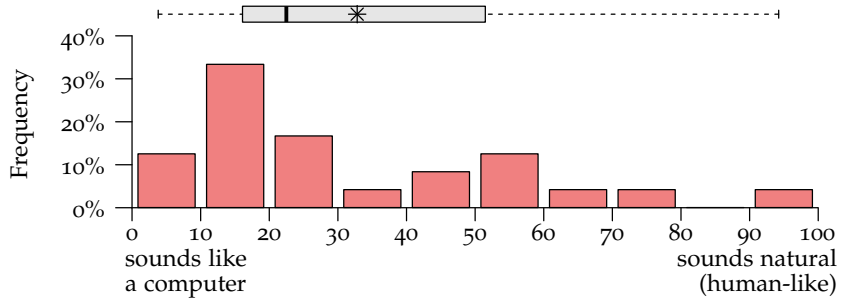
Table 2.4: Example instructions for a given route.




(a) Directions written by human participants.




(b) Directions generated according to the learned policy.




(c) Directions generated by routing web services.

Figure 2.4: Results of the second user study. The values on the horizontal axis represent the user ratings of the directions from 0 ("sounds like a computer") to 100 ("sounds natural (human-like)"). The box plots show the minimum, lower quartile $Q_1$, median (thick line), mean (asterisk), upper quartile $Q_3$, and maximum, and outliers (circles). Data points are considered to be outliers if they are outside the 1.5 interquartile range $[Q_1 - 1.5(Q_3 - Q_1), Q_3 + 1.5(Q_3 - Q_1)]$. $N = 54$ for (a) and (b), $N = 22$ for (c).

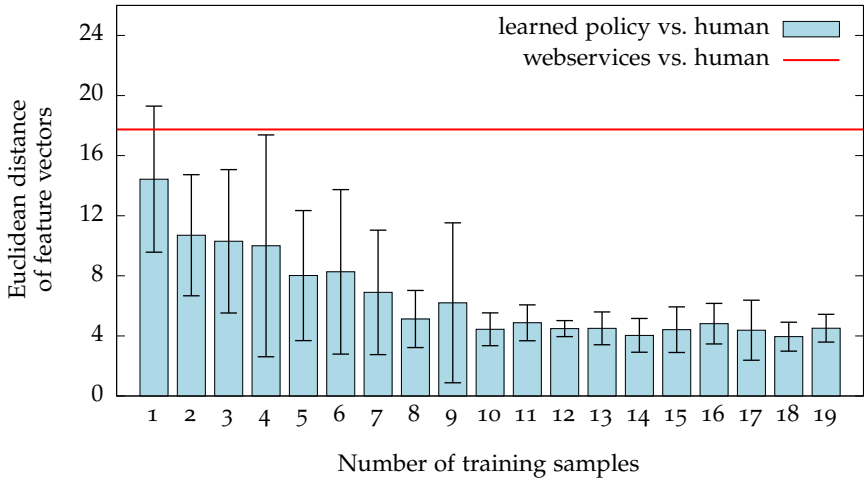



Figure 2.5: The difference between the feature vector of the demonstrations and the learned feature expectations quickly converges.

samples is sufficient to achieve this level of similarity between human and generated directions, as the Euclidean distance between the feature vector of the demonstrations and the feature expectations quickly converges.

The generated descriptions sometimes contain repetitions, for example if two subsequent instructions refer to the same landmark or if the description contains instructions like "Keep going straight ahead" multiple times in a row. According to participants of the second experiment, repetitions make the directions appear less natural. In our framework, however, the Markov property of the MDP prohibits to define a suitable feature for measuring repetitions, as features must only depend on the current action, but not on the previous path leading up to the current state. An additional filtering step after generating the descriptions could address this problem by combining subsequent repetitive instructions to a single instruction that sounds more natural.

## 2.5 DISCUSSION

In our work, we focused on the aspect of which information to include in a route description rather than the linguistic aspects of producing a natural-language description. Hence, we used sentence templates with placeholders that the algorithm filled with the sampled information pieces. For each instruction type, we created multiple templates that the algorithm can fill in and combine so that there is some diversity in the generated descriptions, resulting in 228 templates overall. The templates were selected by random shuffling to avoid repeating the same instruction template multiple times. However, the human-written descriptions still feature higher sentence complexity and higher diversity in sentence formulations. Daniele et al. [37] addressed this issue by

extending our IRL approach for learning how to describe routes by adding a subsequent step of "translating" the instructions generated in a formal language to natural language. They achieve this surface realization step using a recurrent neural network trained on human-written descriptions. This approach eliminates the need for templates and allows the surface realizer to produce sentence constructs that are similar to sentences that humans produce. The authors evaluate the resulting descriptions both according to scoring metrics commonly used in machine translation research and in a human user study showing that their approach produces high-quality route descriptions.

Our approach depends on the correctness and completeness of the map data more than classic turn-by-turn navigation. As an example, the instruction "Turn right at the third traffic lights" may be incorrect in case the map is incomplete and some traffic lights along the way are not registered on the map, whereas "Turn right after 250 m" is unambiguous as long as the metric distance is correct. As road networks and landmarks such as shops change frequently, this is a problem with all map sources, but with OpenStreetMap this issue is particularly visible as the map data is collected by volunteers and map coverage consequently varies substantially between regions. However, this issue is not limited to automatic route generation as humans also frequently miscount landmarks or miss that a landmark they choose may be ambiguous due to similar landmarks in the vicinity.

Generating correct verbal instructions from raw map data is also a challenging problem that is not entirely solved. In historically grown cities, for example in European medieval city centers, intersections can appear in various irregular shapes that are difficult to describe in words. Modern motorway junctions with multi-level bridges, large intersections with pedestrian, cyclist, and rail crossings and are also confusing even for humans. Additionally, the data representation and tagging of route elements in maps is often inconsistent, as the data may be aggregated from different providers in commercial map systems or contributed by different volunteers in open data projects. It is difficult to design rule sets for describing such complicated situations. Hence, it might be worth to investigate whether learning approaches from automatic description generation from images [18] can be transferred to the problem of generating verbal descriptions of routes through intersections in future work.

## 2.6 CONCLUSIONS

In many applications, robots and other computer systems are required to provide route descriptions to humans. It is desirable that these descriptions are natural and intuitive to human users. In order to generate such natural descriptions, we presented an approach that uses inverse reinforcement learning to compute a reward function based

on human demonstrations. This approach allows for generating route directions that sound natural to humans by imitating the description style of a particular user group. Our algorithm learns from a corpus of human-written route descriptions which pieces of information are relevant depending on the route and its environment. We carried out a user study to evaluate the quality of the descriptions generated by our approach in terms of how natural they appear to human participants. The results suggest that our learning algorithm produces descriptions that sound similarly natural as human-written descriptions and clearly outperform the descriptions generated by commercial routing web services.

# 3

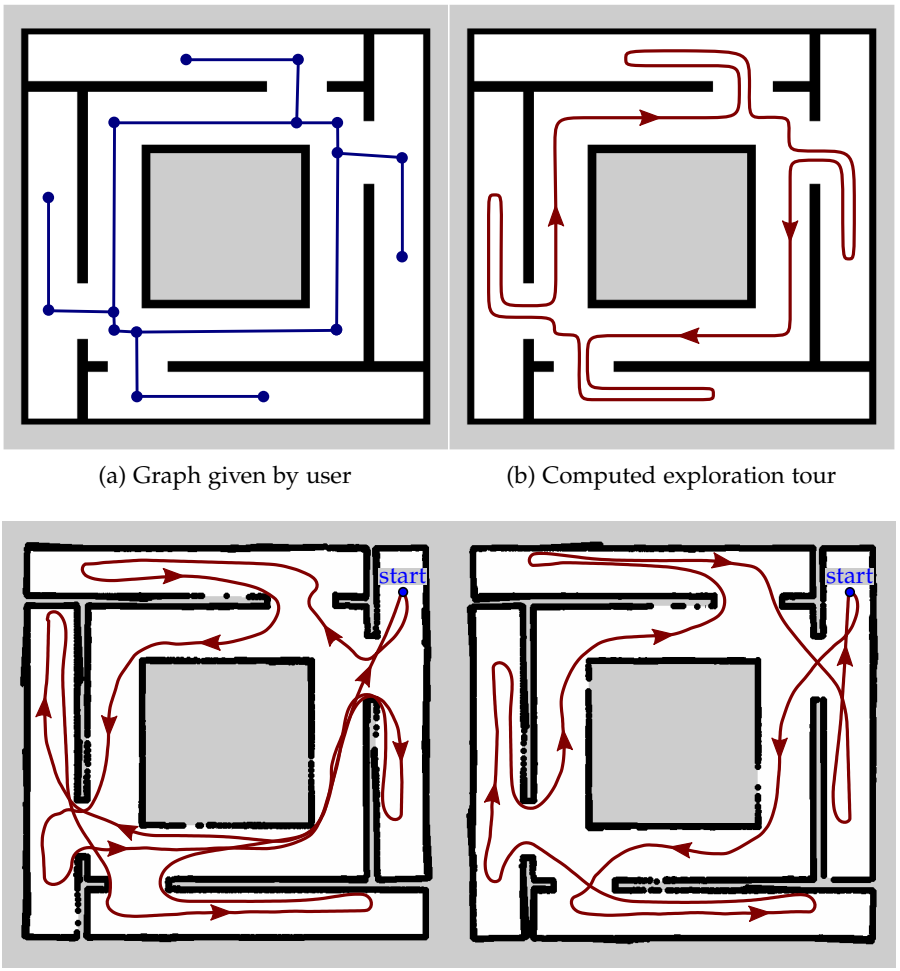
# SPEEDING-UP MOBILE ROBOT EXPLORATION USING BACKGROUND KNOWLEDGE

*In this chapter, we present a solution to the question of how robots can leverage background knowledge provided by the user to speed up autonomous exploration of the environment. Our method is relevant for several real-world applications in which the rough structure of the environment is known beforehand. We present an approach that exploits such background information given as a topometric graph and enables a robot to cover the environment with its sensors faster compared to a greedy exploration system without this information. We implemented our exploration system in ROS and evaluated it in different environments. As the experimental results demonstrate, our proposed method significantly reduces the overall trajectory length needed to cover the environment with the robot's sensors and thus yields a more efficient exploration strategy compared to state-of-the-art greedy exploration, if the additional information is available.*

## 3.1 INTRODUCTION

In the previous chapter, we investigated how automated systems can help human users with wayfinding tasks by providing the information that is most useful to the human users. Now we will turn to the opposite scenario and answer the question of how humans can help robots with exploration tasks by providing background information on the coarse structure of the environment. Acquiring a map of its surroundings is one of the very first tasks that a mobile robot has to execute after being unboxed in an unknown environment, as having access to an accurate map is a prerequisite for all navigation tasks. We assume that the robot already has means for solving the underlying SLAM problem, i.e., it can learn a map and track its pose in this map given sensor data. The exploration task then reduces to generating motion commands that guide the robot to build a complete model of the environment.

Typical approaches to mobile robot exploration assume zero prior knowledge about the world. Accordingly, the robot starts with an empty map and seeks to find a sequence of motion commands to cover the full environment with its sensors. Only few approaches consider additional background knowledge, such as semantic information [161] or information retrieved from dialog with a human user [50].




(a) Graph given by user (b) Computed exploration tour

(c) Greedy exploration (d) Our approach

Figure 3.1: Stages of computing an exploration tour. Based on the graph given by the user (a), our algorithm computes an exploration tour for visiting the nodes using a traveling salesman problem solver (b). While the greedy nearest-first exploration scheme has to revisit rooms (c), our approach exploits the background knowledge by using the computed exploration tour as a guideline for minimizing the overall traveled distance (d).

In this chapter, we take a different approach to exploration than the majority of existing methods. We investigate means that allow a robot to *explore an environment faster if additional information is available*. We seek to answer the question: How much faster can a robot cover an environment with its sensors if it knows the rough layout of the environment beforehand? This problem is relevant for a large number of real-world exploration problems, as in several application scenarios the approximate layout of the environment is known beforehand. This holds, for example, when exploring underground structures such as abandoned mines [171], archaeologically relevant tunnels such as ancient catacombs [62], but also for inspection tasks such as mapping visually intact but possibly unstable buildings after an earthquake [91]. Thus, we do *not* address the problem of exploring a completely unknown environment in this work. Instead, we provide an efficient exploration strategy given prior information about the layout of the environment.

Our work relies on a topo-metric map. This can be seen as a simplified Voronoi-style graph modeling the environment. Figure 3.1a shows an example of such a user-provided graph. Using this information, our approach seeks to find an efficient exploration strategy (see Figure 3.1b) to cover the scene with the robot's proximity sensor as fast as possible. Such a topo-metric graph can be provided by humans or can be automatically derived from floor plans, from previously built maps, or from hand-drawn maps. By knowing the topology of the environment including an estimate of the metric distances, the robot can generate more effective exploration trajectories that avoid redundant work. The robot typically explores dead ends, small loops, and similar structures first so that it will not have to return to these locations later during the mission. In the environment in Figure 3.1, for example, it is advantageous to explore the rooms on the outside first, as the robot then has to traverse the corridor only once. We formulate the problem as a Traveling Salesman Problem (TSP) and use its solution to guide the robot through the environment. The length of the path that our approach generates (Figure 3.1d) is significantly shorter than the one resulting from greedy exploration (Figure 3.1c) because it avoids traversing the corridor multiple times. Finally, our approach combines the TSP solution with frontier-guided exploration. This combination avoids redundant work on a global scale and at the same time ensures that all areas are covered by the robot's sensors. In addition to that, exploiting such a graph structure also allows operators to easily exclude parts of the environment that the robot should not explore by simply labeling nodes or edges in the graph as "no go areas". As we will show in our experimental evaluation, our approach leads to significantly shorter overall exploration trajectories and thus significantly reduces the time needed to cover the terrain.

## 3.2 RELATED WORK

Typical approaches to mobile robot exploration aim at selecting view points that minimize the time needed to cover the whole terrain. The most popular approach is probably frontier-based exploration by Yamauchi [187] in which the robots always move towards the closest unexplored location. Another option is using stochastic differential equations for goal-directed exploration [152]. The idea is to seed particles in the workspace and subject them to forces so that they move toward unexplored areas. The particles themselves follow molecular dynamics and eventually enter unknown areas and provide candidate goal positions for the exploring robot. A usual assumption in exploration systems is the knowledge about the pose of the robot during the mission, but there exist also exceptions such as information-theoretic methods that seek to minimize the uncertainty in the belief about the map and the trajectory [104, 159] simultaneously. In a similar way, Sim et al. [154] select vantage points to optimize the map accuracy by striving for loop closures.

The majority of exploration approaches start from scratch, i.e., they do not exploit background information or any assumptions about the structure of the environment. There are only a few techniques that incorporate background information into the decision-making process of where to move next. An interesting approach by Fox et al. [52] aims at incorporating knowledge about other environments into a cooperative mapping and exploration system for multiple robots. Related to that, Perea et al. [136] exploit already explored spaces to predict future loop-closures and to guide the exploration in this way. Others use semantic information [160, 161] or an environment segmentation [184] as background knowledge to optimize the target location assignment. Such approaches received considerable attention in multi-robot exploration. In addition to that, Zlot et al. [192] propose a multi-robot target assignment architecture that follows the ideas of a market economy. The assignment considers sequences of potential target locations and trades individual tasks among the robots using single-item first-price sealed-bid auctions. Similar approaches using auctions for task allocation processes have been applied by Gerkey and Matarić [56]. In contrast to that, Ko et al. [86] present an approach that uses the Hungarian method [94] to compute the optimal assignments of open frontier cells to robots in a given instance. The work of Ko et al. furthermore focuses on aligning the robot trajectories in case the start locations of the robots are unknown. Wurm et al. [184] propose a coordinating technique for teams of robots using a segmentation of the environment to determine exploration targets for the individual robots. This is related to the spatial semantic hierarchy introduced by Kuipers and Byun [95]. Thus, semantic information [161] or segmentations [184] approaches show that by assigning robots to unexplored

segments instead of frontier targets, a more balanced distribution of the robots over the environment is obtained. A related approach has been presented in the work by Holz et al. [73], which first analyses different exploration approaches and then proposes heuristics for improving the performance of the exploration strategies.

In this chapter, we investigate means for allowing robots to explore an environment faster if additional information is available. We target application scenarios in which the approximate layout of the environment to explore is known beforehand. This is also related to coverage techniques [105, 185] but we do not assume a grid map or polygon to be given beforehand for planning view points. In contrast to purely graph-based coverage techniques as in [186], we also consider the surroundings around the graph nodes in a local exploration strategy. Patrolling approaches such as [135, 138] focus on multi-robot coordination and global exploration strategies, while our approach combines global with local strategies. Exploiting abstracted map information has also been investigated in other navigation problems. For example, Chronis and Skubic [31] exploit hand-drawn sketch maps for navigation, which enables robots to derive and use qualitative spatial relations to move along the given path. Related to that, Freksa et al. [54] use schematic maps to derive qualitative spatial relations for supporting navigation. Our work also exploits topo-metric information in form of a graph but the user is not expected to specify the path for the robot. Instead, the environment information is used for computing an optimal exploration strategy at a global scale once in the beginning, which is then used for visiting the individual areas in the environment. Our approach is also related to works by Brummit and Stentz [27] as well as Faigl et al. [44] as both approaches formulate the problem of coordinating multiple robots during exploration as a Multiple Traveling Salesman Problem (MTSP). Our approach also uses the TSP formulation but on the user-provided graph and combines the solution with local exploration at the individual locations.

Kulich et al. [96] also try to minimize the length of the remaining exploration tour by solving TSPs. In their approach, the nodes of the TSP consist of open frontiers and the edges represent shortest paths between the frontiers. To reduce the size of the graph and thus the complexity of the TSP, they choose a subset of frontiers from where the remaining frontiers are within the robot's viewing range. They directly use the TSP solution to determine the next frontier and repeat the process after exploring the frontier. In a related approach, Férmin-León et al. [45] construct a TSP from topological segmentation online while exploring an unknown environment. In contrast to Kulich's and Férmin-León's approaches, our approach constructs a TSP on the user-provided graph which contains more a-priori information about the topology of the environment so that better plans can be generated. Our approach solves a TSP only once in the beginning or

after detecting and correcting errors in the graph. As each node of the TSP potentially represents multiple frontiers, the TSP graph is typically small and simple in structure, thus easier to solve than a general TSP.

## 3.3 GUIDING EXPLORATION WITH BACKGROUND KNOWLEDGE

In this section, we describe our approach to generating navigation actions for acquiring sensor data about the complete environment. We hereby exploit background knowledge in form of a graph-structure provided to the robot. We model the global navigation problem as a Traveling Salesman Problem (TSP) in order to find a tour that covers the whole area and is as short as possible. Our approach solves the TSP once in the beginning and uses the TSP tour to guide a frontier-based exploration. In case the robot detects changes in the environment, it updates the graph and re-computes the TSP solution.

### 3.3.1 *Representation of background information about the environment*

We represent the background knowledge as a graph $G(V, E)$ consisting of nodes $V = \{v_1, \ldots, v_n\}$ and undirected edges $E \subseteq V \times V$. The nodes represent rooms or other convex regions in the environment and the edges indicate the connections between the regions, for example doors or passages. We hereby assume that each room to be explored contains at least one node and corridors contain a node in front of each door and intersection (note that the terms "room", "corridor", "intersection", etc. are only used for illustrative purposes here, the algorithm itself does not contain any notion of these concepts). The lengths of the edges need to reflect only roughly the true metric distances (see Section 3.4.5 for an evaluation of the robustness against noise). The resulting graph is a topo-metric representation of the environment and is used to optimize the exploration tour with the starting position and orientation of the robot with respect to the graph given by the user.

In order to guide the exploration, the user may wish to explicitly exclude regions from the exploration, for example regions that are dangerous for the robot or regions that are not of interest for the task. We represent the user's instructions by annotating the nodes of the graph with labels $A : V \to \{explore, skip\}$ indicating whether the region close to the node should be explored or not.

### 3.3.2 *Representation of the exploration problem*

The robot's objective is to optimize the tour length given the user-provided graph. The tour must visit each node at least once in order to cover the surrounding region. In most cases, there will be rooms that the robot has to pass through more than once, for example, the

robot will have to enter the corridor multiple times to access the individual rooms. Hence, our goal is to find the shortest path $T$ in the graph $G$ that visits all nodes at least once and returns to the start node. This problem is closely related to the well-known Traveling Salesman Problem (TSP): Given a list of cities and the distances between each pair of cities, find the shortest tour that visits all cities exactly once and finally returns to the starting point. A TSP solver is able to provide the optimal solution to this problem.

To represent this problem as a traveling salesman problem, we complete the graph $G$ to a clique by adding edges between each pair of nodes. We define the length of the new edge $(v_i, v_j)$ as the length of the shortest path between the positions of nodes $v_i$ and $v_j$ and use Johnson's algorithm [81, 153] to compute the shortest path distances between all pairs of nodes in time $\mathcal{O}(|V|^2 \log|V| + |V||E|)$.

The resulting graph $G'$ is symmetric and the triangle inequality holds, as the length of newly inserted edges between two nodes is never longer than the distance of the shortest path between these nodes. $G'$ is a clique, thus there exists a shortest tour $T'$ in $G'$ that visits each node exactly once and returns to the start node. The shortest tour $T$ in the original graph $G$ that visits all nodes at least once cannot be shorter than $T'$ due to the triangle inequality. The tour $T$ with revisiting nodes can be easily retrieved from $T'$ by replacing the edges added using Johnson's algorithm by the shortest path between the edge's end nodes.

In some scenarios, it might not be required that the robot returns to the starting location after completing the exploration task. In this case, the robot only requires a shortest path from the given start node $v_s \in V$ that visits all nodes and leads to an arbitrary end node. This case can also be modeled as an asymmetric TSP by setting the length of the edges from $v_i$ to $v_s$ to zero for all $v_i \in V \setminus \{v_s\}$, so the robot can "teleport" from an arbitrary node back to the start node at no costs. After determining the shortest tour, the zero-length edge $(v_e, v_s)$ is removed so that the remaining path leads from $v_s$ to the arbitrary end node $v_e$ without returning. The case where both the start node $v_s$ and the end node $v_e$ are fixed can be modeled similarly by setting the costs or $(v_i, v_s)$ to infinity for all $v_i \in V \setminus \{v_s, v_e\}$. Note that in both cases the modified graph is not symmetric and consequently not metric anymore.

### 3.3.3 *Solving the TSP*

Unfortunately, solving the TSP is NP-complete in the general case. For metric TSPs, however, there exist polynomial-time approximation schemes. For example, the Christofides algorithm [30] is able to find a tour that is guaranteed to be at most 1.5 times longer than the optimal solution and requires a run time of $\mathcal{O}(|V|^3)$.

The problem instances we consider, however, typically generate TSP instances of moderate complexity. State-of-the-art TSP solvers such as Concorde [9] can determine the exact solution to such problems within a few seconds. As our algorithm has to solve a TSP only once at the beginning of the mission, this step only adds a marginal amount to the overall run time in comparison to the time the robot needs to actually move through the environment and acquire sensor data. Thus, we use the Concorde solver in our implementation instead of resorting to suboptimal approximation schemes.

### 3.3.4 *Frontier-based exploration exploiting the TSP solution*

The tour determined by the TSP solver provides a *global strategy* for visiting all regions of the environment by ordering the corresponding nodes. Although the tour is the optimal solution to the TSP, the resulting trajectory is likely to be a suboptimal exploration trajectory for a real robot. The reason for that is that the TSP formulation does not consider visibility information of the robot's sensor. For example, parts of the environment may be visible from multiple nodes, so that the robot only has to visit one of the nodes. At the same time, the exploration system must ensure that the surroundings of a node in the TSP is fully explored.

Therefore, we combine the TSP and the problem of exploring the *surroundings of each node* taking into account the sensing capabilities of the platform. To this end, we use a frontier-based exploration approach [187] for the local exploration but using the TSP solution on the global scale. In order to incorporate the sensor information into the world model, we rely on a standard graph-based SLAM system for 2D range sensing. It builds a grid map while moving through the environment and acquiring sensor data. The robot's grid map consists of cells that are initially labeled as unknown. While traveling, the robot updates the cells within its field of view towards free or occupied with a standard occupancy model. Cells on the border between free and unknown space that have not been covered yet are called *frontier cells*. The robot can extend its knowledge about how the environment looks like by navigating to frontier cells in order to inspect the cells laying beyond the frontier. During this exploration, multiple frontiers appear and the robot has to decide to which frontier to move next. A common approach for choosing a frontier is to apply a cost function to the frontiers that takes into account cost factors such as the distance between the robot's current pose $r$ and the frontier position $f$, the relative orientation, and the expected Information Gain (IG):

$$\begin{aligned} \text{cost}(f) \quad := \quad & \alpha_1 \cdot \text{distance}\,(r, f) \\ & + \alpha_2 \cdot \text{orientation}\,(r, f) \\ & - \alpha_3 \cdot \text{IG}\,(f) \end{aligned} \tag{3.1}$$

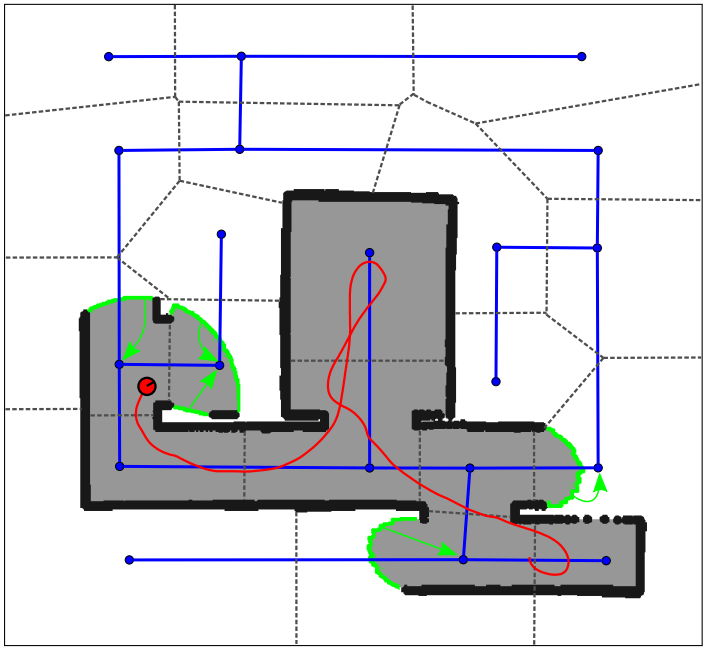


Figure 3.2: The graph is used to guide the exploration and this requires an assignment of frontiers to graph nodes. The algorithm assigns each new frontier (green) to the nearest graph node (green arrow) by considering the length of the shortest path from the frontier to any node of the graph. The areas for the assignment are illustrated through the dashed lines and walls of the map. In unexplored areas, only the graph can be used to estimate the assignment. While the explored region grows, the assignment of a frontier to a node can change due to newly sensed walls. The thickness of the frontiers has been increased artificially in this figure to improve visibility.

For applications which require additional or more complex cost terms, in particular probabilistically dependent or synergic cost terms, a multi-criteria decision making framework [12] could be used to design a suitable cost function. The robot updates the map based on the perceptions, computes the frontiers, and decides to navigate to the frontier with the lowest cost at a fixed frequency of 1 Hz.

In order to account for the global navigation strategy determined by the TSP tour, we add another cost factor to the cost function. Whenever a new frontier appears, we determine the nearest node $v^\star$ using Dijkstra's algorithm on the grid map, see Figure 3.2 for an illustration of the assignment process.

The robot may explore the frontier at any time when passing by that node. When the robot passes by $v^\star$ for the last time while following the global tour computed by the TSP, however, our strategy enforces that the robot explores the frontier in order to avoid having to return to the node another time, as that would increase the tour length unnecessarily. Hence, we follow the TSP tour from the robot's original

start node $v_s$ up to the point when the TSP tour passes through $v^\star$ for the last time:

$$p := \left( (v_s, v_{k_1}), (v_{k_1}, v_{k_2}), \ldots, (v_{k_m}, v^\star) \right) \tag{3.2}$$

We then sum up the lengths of the edges along the tour

$$d := \sum_{(v_i, v_j) \in p} \text{length}\left( (v_i, v_j) \right) \tag{3.3}$$

and add this distance $d$ to the cost function in Equation 3.1 with a high weight $\alpha_4$ so that it dominates the cost function. In this way, the robot explores the frontiers in the order determined by the solution of the TSP and applies the original cost function within each room, as $d$ is the same for all frontiers within the neighborhood of the nearest node.

During exploration, the robot might encounter frontiers that are unreachable because of obstacles. If the robot's path planner determines that it cannot navigate to a frontier, the robots adds this frontier to a blacklist. If the user has annotated nodes to be skipped during exploration, the algorithm adds all frontiers associated with those nodes to the blacklist. When the association of frontiers to nodes changes, the algorithm modifies the blacklist accordingly.

### 3.3.5 *Re-planning*

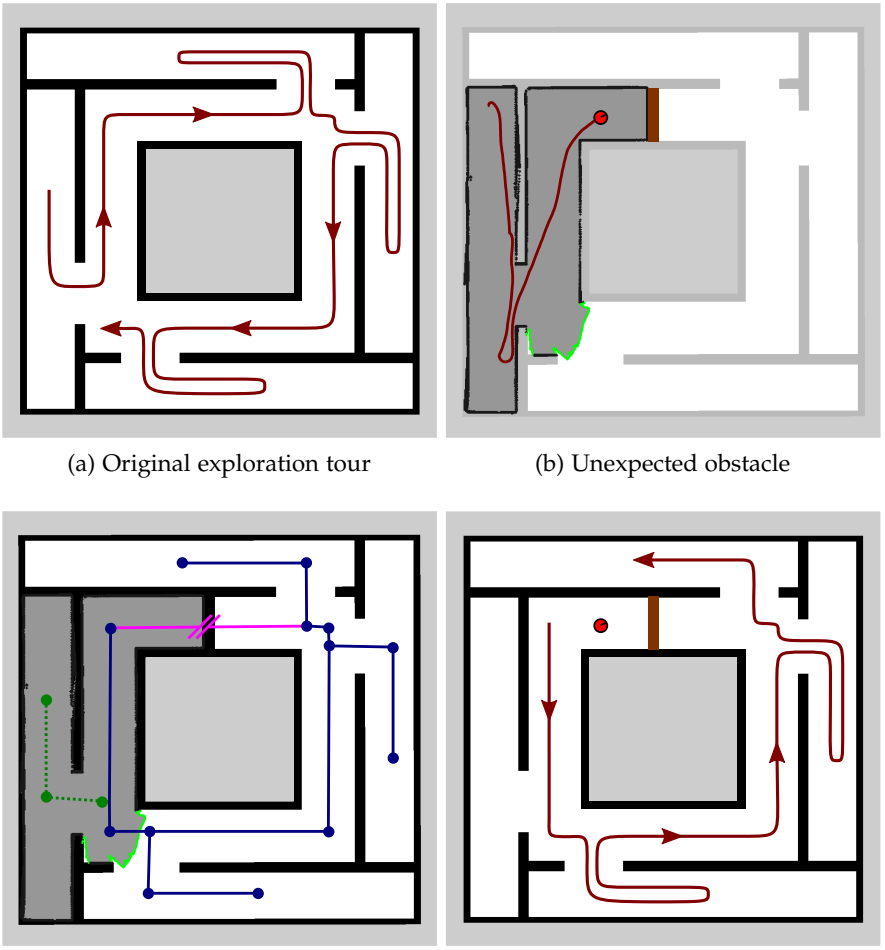
Our approach relies on the assumption that the user-provided graph correctly represents the topology of the environment. If the graph contains non-traversable edges, the resulting global strategy will be suboptimal as the robot would make a detour around the obstacle and continue to explore the frontiers in the order as planned originally. To avoid such detours, the robot has to re-plan the global strategy once it detects that the planned path is obstructed:

1) Correct the user's graph according to the observations.

2) Complete the graph to a clique $G'$ as explained above.

3) Remove the nodes that have already been visited and do not have any frontiers associated with them, except for node $v_s$ to which the robot has to return after exploring (if applicable) and the robot's current node $v_r$.

4a) If the robot has to return to the original start node $v_s$: After exploring the last remaining node, the robot has to return to $v_s$ instead of the current node $v_r$. Hence, we replace the edge lengths $(v_i, v_r)$ for all $i \in \{1, \ldots, n\}$ by the shortest path costs between the positions of nodes $v_i$ and $v_s$, making the TSP asymmetric.



(a) Original exploration tour

(b) Unexpected obstacle

(c) Modified graph

(d) New plan

Figure 3.3: Re-planning in case of unexpected obstacles. Based on the graph given by the user, the algorithm computes an exploration tour, in this case without returning to the start location (a). While traveling, the robot encounters an unexpected obstacle blocking the planned path (b). The algorithm modifies the graph by removing the blocked edge (magenta) and already visited nodes without open frontiers (green) (c). Solving the TSP on the new graph then yields a global strategy for exploring the remaining parts of the environment (d).

4b) If the robot does not have to return to the start node: Set the edge lengths of $(v_i, v_r)$ to zero for all $i \in \{1, \ldots, n\}$ to allow the path to end at an arbitrary node as explained in Section 3.3.2.

5) Solve the TSP to get a tour $T'$ for the remaining nodes.

6) Replace the edges added by Johnson's algorithm by the shortest paths, passing previously visited nodes if necessary.

The resulting tour lets the robot explore the remaining areas on the shortest path assuming that the rest of the topology is correct. Figure 3.3 shows an example of a re-planning step.

## 3.4 EXPERIMENTAL EVALUATION

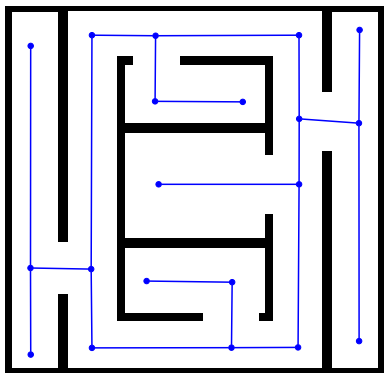
For testing our approach, we implemented our system in ROS [141] and simulated the exploration in Player/Stage [57]. We based our exploration implementation on the ROS exploration stack [42] that already implements frontier-based exploration. We keep the default cost function, except for setting the weight for the expected information gain to zero (see Section 3.4.6 for a discussion). This ROS greedy exploration strategy serves as a baseline for the evaluation of our approach.

### 3.4.1 *Environments*

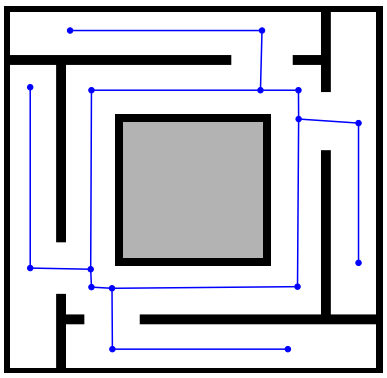
For our evaluation, we use a set of different maps, containing artificial maps, real maps recorded with mobile robots, and maps created from hand drawings of archaeological sites. See Figures 3.4 and 3.5 for an overview of the maps used in the experiments. Maps 1–4 are artificial maps for studying the behavior of our approach in environments with different levels of connectedness. We furthermore used a map of the Intel research lab in Seattle (Figure 3.4e). This map contains a high amount of clutter, which poses a challenge to exploration algorithms in general.

Finally, we evaluated our approach using the data from a real-world application for digitizing cultural heritage sites. Within the ROVINA project, we digitized the Catacomb of Priscilla in Rome, Italy [62]. During previous expeditions, archaeologists created hand-drawn maps of the catacombs. Based on these maps, we created a graph of the catacomb's topology (see Figure 3.5a).

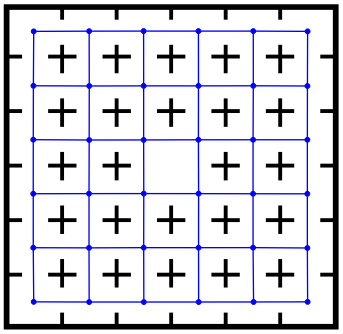
We then used this information as background knowledge for our exploration system. As the access to the catacomb is quite restrictive, we used a map (see Figure 3.5b) built in the Catacomb of Priscilla during a previous robotic mission with the SLAM system. In this previous mission, the robot was joysticked through the environment and we used the map from this run for the simulation, so that multiple



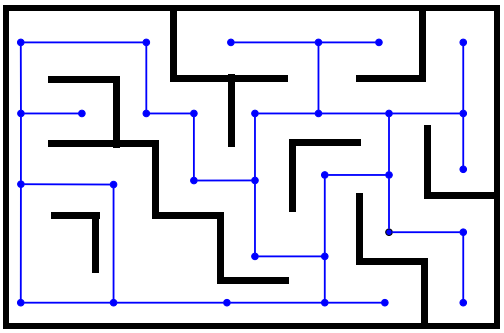
(a) Map 1



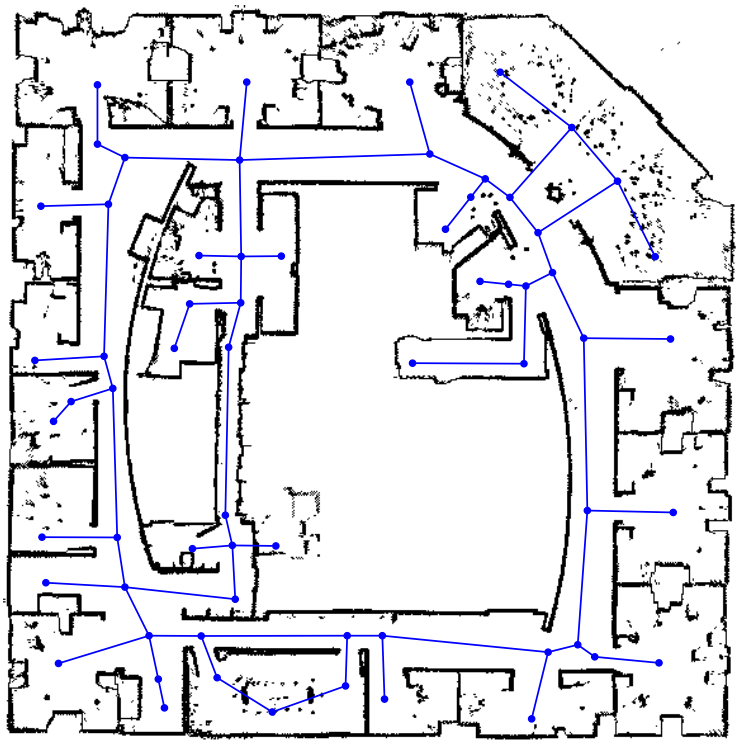
(b) Map 2



(c) Map 3

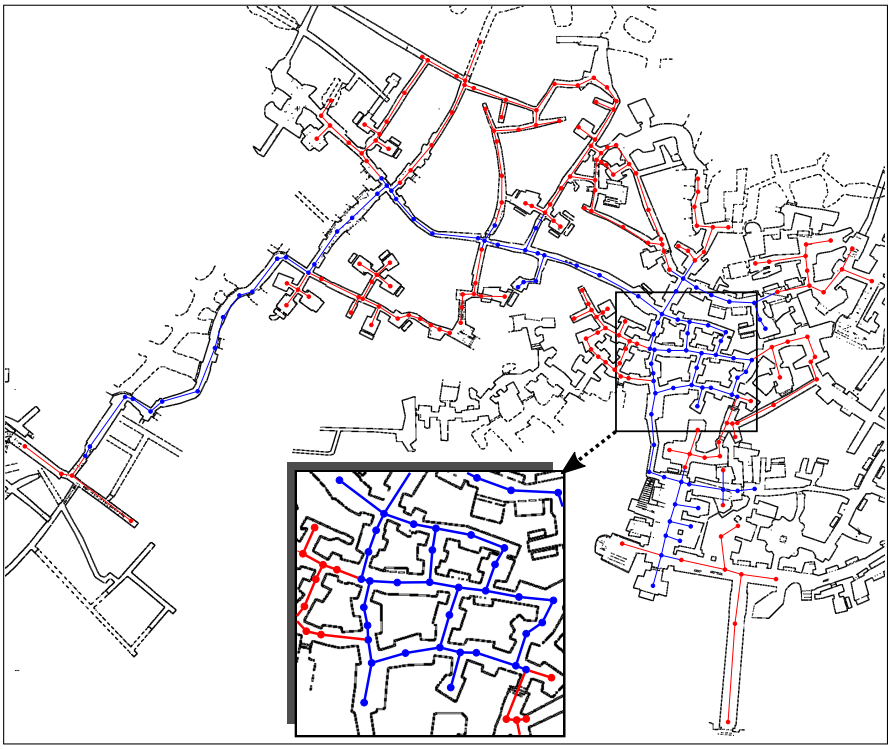


(d) Map 4

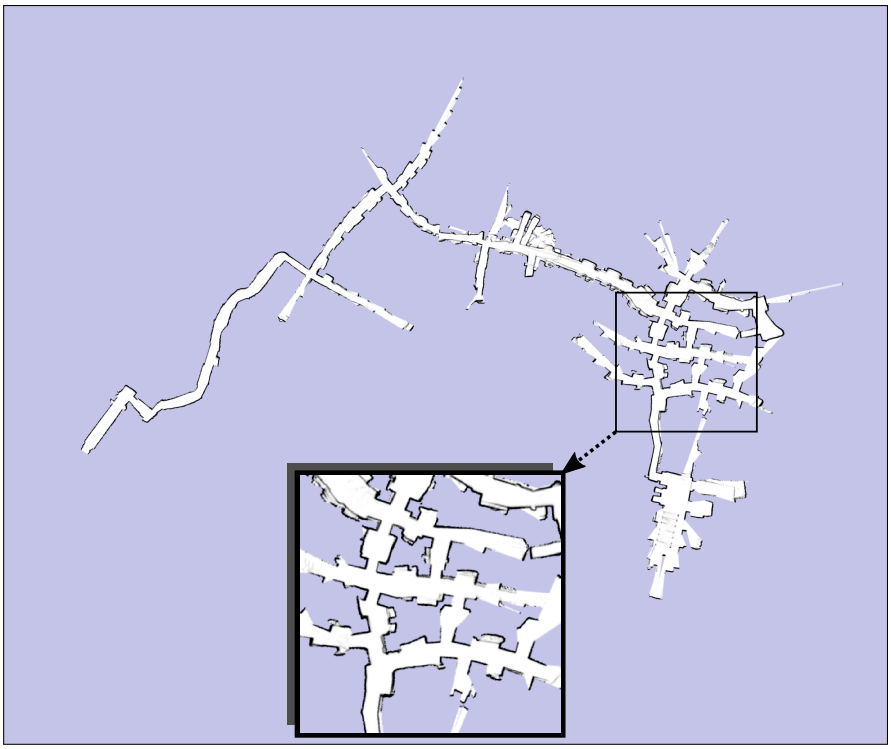


(e) Intel research lab Seattle

Figure 3.4: Maps and graphs used in the experiments: Part I. (a)–(d): Artificial maps. (e): Interior of the Intel research lab in Seattle, courtesy of Dieter Fox.



(a) Priscilla catacomb sketch map



(b) Priscilla catacomb SLAM map

Figure 3.5: Maps and graphs used in the experiments: Part II. (a): Map drawn by archaeologists of the Priscilla Catacomb in Rome, Italy. (b): Corresponding map built by a real robot.

| Map | Runs | Mean difference | Gain | p value |
|---|---|---|---|---|
| Map 1 | 10 | 40.6 m | 13.9 % | 0.019 |
| Map 2 | 10 | 50.1 m | 17.3 % | 0.002 |
| Map 3 | 10 | 9.0 m | 4.0 % | 0.051 |
| Map 4 | 10 | 30.6 m | 15.2 % | 0.004 |
| Intel lab | 10 | 164.5 m | 16.5 % | 0.001 |
| Priscilla with return | 5 | 49.3 m | 10.6 % | 0.020 |
| Priscilla without return | 5 | 101.8 m | 24.5 % | 0.004 |

Table 3.1: Comparison of the traveled distance.

exploration runs can be simulated and evaluated with statistical tests. Please note that the SLAM map deviates from the sketch map in some parts. For example, some corridors that are straight on the hand-drawn map appear curved on the SLAM map. Our experiments show that our approach compensates for such inaccuracies as long as the frontiers can be assigned correctly to graph nodes.

### 3.4.2 *Background information*

We created the graphs resembling the background information manually by placing one node per room and connecting the nodes according to the map topology. In general, one node per convex region is sufficient, as the greedy exploration then dominates the exploration strategy in the convex regions, for which it is well-suited. If the user places more than one node in a convex region, the global tour strategy gains influence, giving the user more control on the exploration strategy.

With our approach, the human operator can easily limit the exploration area by marking nodes that the robot should not explore. In the Priscilla catacomb shown in Figure 3.5a, for example, we marked the nodes that should explored in blue and the nodes to be skipped in red. The robot then deliberately leaves frontiers open that get assigned to the red parts of the graph, so the robot only explores the desired region.

Depending on the application scenario, the robot may be required to return to its starting position after completing the exploration. For the Priscilla map, we run experiments both with and without returning to the starting location.

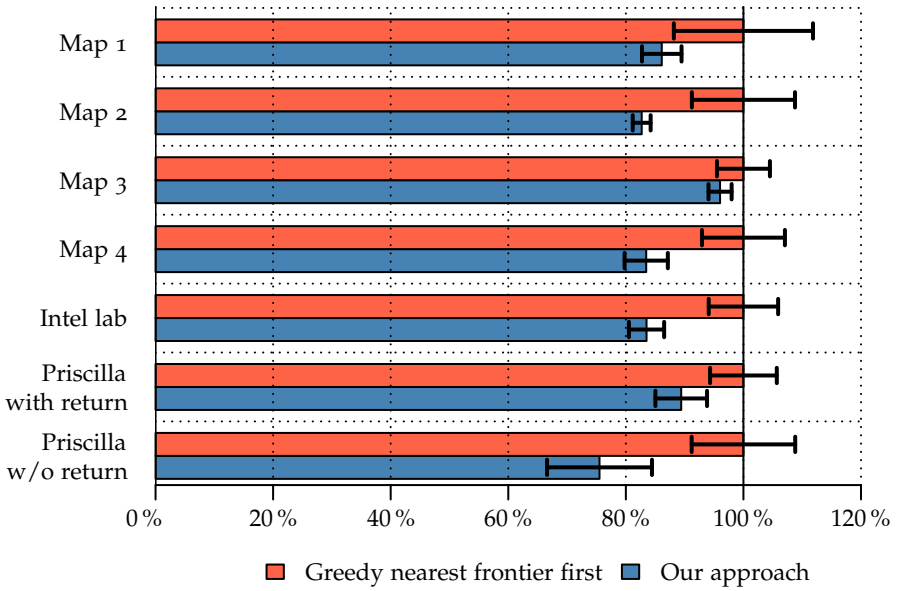



Figure 3.6: Mean and 95 % confidence interval of the traveled distance. The distances are normalized so that the greedy approach equals 100 %.

### 3.4.3 *Traveled distance*

The distance that a robot has to travel to explore the whole environment depends on the start location. Hence, we chose a set of start locations for each map and executed our approach and the baseline approach once for each start location.

Figure 3.6 visualizes the mean length of the exploration trajectory together with the corresponding 95 % confidence intervals. Table 3.1 shows the results of a paired t-test at the 0.05 level for the exploration tour length for 10 respectively 5 runs in the environments. The trajectories generated by our approach are significantly shorter than the trajectories generated by the baseline approach on all maps except Map 3. On Map 3, the greedy strategy is already near-optimal in many cases, as the map is similar to a large freespace area.

For the Priscilla map, we run experiments both with returning to the starting location and without. Our approach uses the knowledge about whether or not to return to the starting location to optimize the exploration tour as described in Section 3.3.2. In case the robot has to return to the starting location, the TSP tour generated by our approach is independent of the starting location of the robot. The robot explores the rooms always in the same sequence, hence the variance of the exploration tour length is small. In the baseline approach of greedy exploration, by contrast, the order in which the robot explores the rooms strongly depends on the starting location, leading to a higher variance in the tour length. In case the robot does not have to return to the start, the overall tour length depends on the start location, leading

| Map | Nodes to explore | Time required to solve TSP |
|---|---|---|
| Maps 1–4 | 16–36 | $(0.013 \pm 0.006)$ s |
| Intel lab | 65 | $(0.84 \pm 0.43)$ s |
| Priscilla catacomb with return | 94 | $(5.92 \pm 2.57)$ s |
| Priscilla catacomb without return | 94 | $(0.38 \pm 0.11)$ s |

Table 3.2: Time required for solving the TSP.

to a higher variance in the tour length for both approaches when averaged over different starting locations.

As can be seen in Table 3.1, our approach outperforms frontier-based exploration in all settings. The gain in exploration time is achieved by better planning the exploration. For example, on the Priscilla map when the robot does not have to return to the starting location, our approach always explores the corridor system in the left half of the map last (or first in case the robot starts already in the corridor) so that the robot does not have to traverse the corridor system for a second time. On a local scale, our approach enforces that the robot finishes exploring a room completely before moving on to the next room, such that the robot will not have to enter the room again.

### 3.4.4 *Run time*

For computing the global exploration strategy, our approach has to solve a traveling salesman problem once at the beginning of the exploration. In our experiments, we use the Concorde solver [9], which attempts to find the optimal solution of the TSP using branch-and-cut strategies. Table 3.2 shows the time required to solve the TSP for the individual maps. As the TSP has to be solved only once in the beginning (or when changing the graph leads to re-planning), the required time adds only marginally to the total exploration time.

During exploration, our algorithm needs to maintain a distance information for assigning each frontier to the nearest graph node (see the illustration in Figure 3.2). For this purpose, a low resolution map is sufficient. As this shortest path information changes whenever new walls appear during the exploration, we recompute it at a frequency of 1 Hz at a grid resolution of 15 cm, which can easily be done online during the exploration.

### 3.4.5 *Robustness*

To evaluate the robustness of our approach, we added Gaussian noise with increasing standard deviation to the node positions of the user-

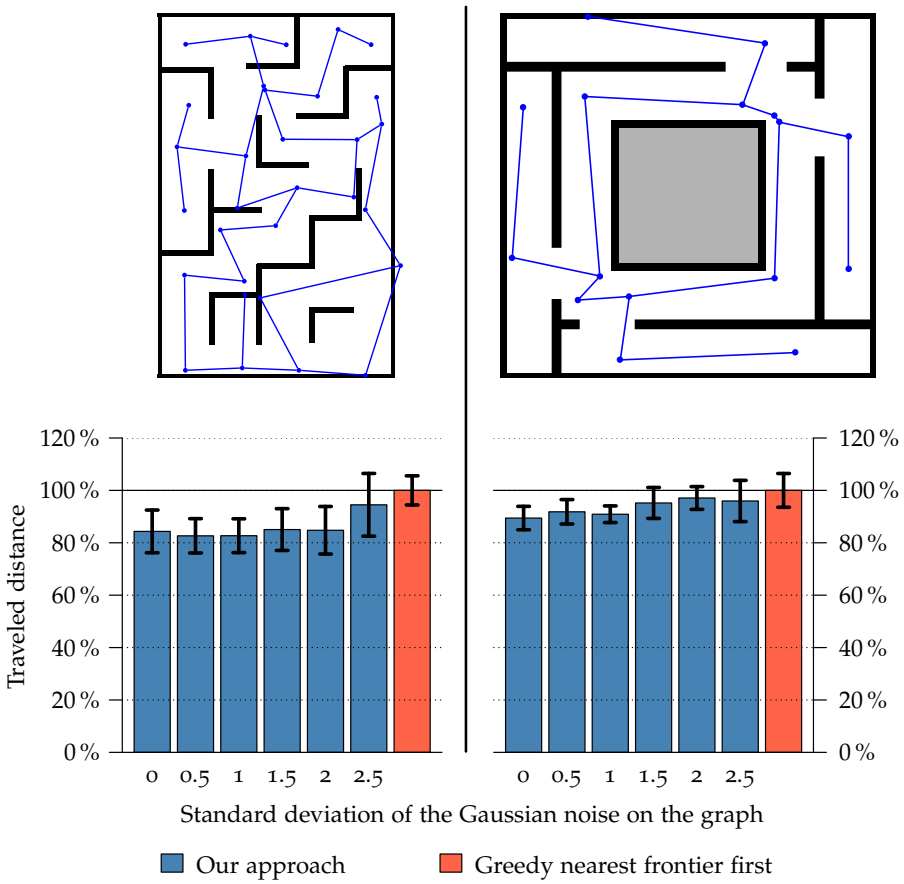



Figure 3.7: Robustness of the exploration strategy when Gaussian noise is added to the human-provided graph. The figure shows the experiments for Map 4 on the left and Map 2 on the right (see Figure 3.4 for the original graphs provided by the human user). Top: Gaussian noise of $\sigma = 1\,\text{m}$ added to the original graph. Bottom: Mean and 95 % confidence interval of the traveled distance for different amounts of Gaussian noise ($n = 10$ for each condition). The distances are normalized so that the greedy approach equals 100 % for the respective map.

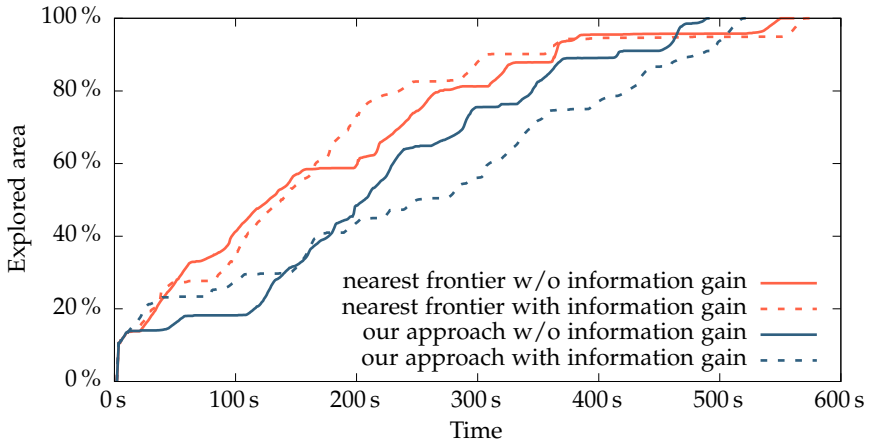



Figure 3.8: Comparison of the exploration progress for different strategies for a run on Map 4 shown in Figure 3.4d. While the nearest frontier approach makes faster progress at first, our approach explores more thoroughly and avoids having to backtrack, thus our approach finishes in a shorter total time.

provided graphs shown in Figure 3.4, which consequently also changes the edge lengths, and evaluated the effect on the overall tour length. Figure 3.7 shows that our approach still works even if the graph provided by the user is considerably inaccurate. For Gaussian noise with a standard deviation of $\sigma = 1\,\mathrm{m}$ as shown in the top of the figure, our approach still performs significantly better than the greedy approach according to a paired t-test at the 0.05 level. This robustness is mainly achieved by dynamically updating the assignment between frontiers and graph nodes using Dijkstra's algorithm as explained in Section 3.3.4.

### 3.4.6 *Influence of information gain*

Many exploration approaches use the estimated information gain in the cost function for selecting the next goal. The rationale of this approach is to greedily explore the largest frontiers first in order to cover the largest possible area within a given time limit. However, this strategy tends to leave smaller frontiers unexplored, which would be counterproductive in our scenario as the robot would have to come back later to complete the exploration.

Figure 3.8 illustrates how fast the presented approaches make progress during exploration. The nearest frontier strategy explores greedily, while our approach explores thoroughly. Consequently, the greedy approach covers more area per time frame than our approach in the beginning. After exploring the biggest frontiers, however, the greedy approach has to backtrack to explore the smaller areas and rooms that it left out, which is time-consuming and causes the total exploration time to be longer than with our approach. Considering the

information gain in the cost function increases this effect even further, leading to longer overall exploration times.

Our goal is to explore a map completely without having a fixed time limit after which the mission is stopped. Hence, our approach without considering the information gain is the best choice for covering the complete environment.

## 3.5 DISCUSSION

In this work, we focused on the question of *how to benefit* from background information made available by the user in form of a topo-metric graph. Metric information is available in many cases, for example when floor plans or in our case archeological drawings are available. When the user provides the background information as a sketch map, however, the proportions of the areas drawn in the sketch are likely to be incorrect or distorted. Hence, two questions remain: *How to interpret* information given by the user, and *how to align* the spatial representations of the robot with the given background information. Other research groups have presented interesting approaches for solving these questions. Skubic et al. have presented extensive research on how to interpret hand-drawn sketch maps for wayfinding, including the role of landmarks, spatial relations, and connections to linguistics [31, 155, 156]. Shah and Campbell [149, 150] also presented systems for interpreting sketches drawn by human users and showed in user studies that the interface is natural and easy to use even for novice users. While these approaches focus on spatial relations between the robot and landmarks, Boniardi et al. [25, 26] instead use floor plans sketched by a human user to localize and navigate in indoor environments. Based on a Monte Carlo localization algorithm, they additionally track two scaling variables in the robot's state space to account for distortions in the user's drawings, making the system robust against metric inaccuracies. Integrating such a sketch interpretation system with our system for generating efficient exploration strategies would lead to a natural interface for guiding a robot in exploration tasks. The sketch interpretation might be improved even further by including recent advances in sketch interpretation with neural networks [64] that allow the algorithm to recognize shapes and symbols within the drawing.

## 3.6 CONCLUSIONS

We presented a novel approach to autonomously learning a model of the environment under the assumption that the layout of the environment is known beforehand in form of a topo-metric graph. Such a graph can be provided by humans or automatically derived from existing floor plans or other maps. Our system exploits the given topo-

metric graph to generate shorter navigation trajectories that cover the terrain. We represent the problem as a traveling salesman problem and derive a global exploration strategy from its solution. Locally, a frontier-based approach is used that exploits the TSP solution and fully covers the environment with the robot's sensors. Accordingly, the overall trajectory length is reduced as the robot will move to dead ends, small loops, and similar structures first and thus does not need to return to these locations later on. We implemented and thoroughly tested our approach in different environments. The results show that our approach significantly reduces the time needed to cover the terrain with the robot's sensors requiring a negligible amount of additional computational resources. Our approach is valuable for several real-world exploration problems, for example, when exploring abandoned mines or archaeological sites, or for inspection tasks where the layout of the environment is not completely unknown.

In this chapter, we focused on an exploration task where background knowledge is available only on a coarse level as a topo-metric graph. In Chapters 5 and 6, we will have a look at the closely related problem of planning a coverage tour through a known environment for observing as much of the environment as possible with a robot's camera. In that case, we will assume that the robot already has access to a 3D model of the environment and we will present an approach for automatically creating a metric graph as an abstraction of the environment. We will again formulate the problem as a TSP and use its solution to guide the robot on a tour through the environment.

# 4

# SLAM FOR HUMANOIDS WITH A COMBINED RGB AND DEPTH DESCRIPTOR

*Simultaneous Localization and Mapping (SLAM) is a prerequisite for all autonomous mobile robots. For humanoid robots, this task is particularly challenging due to limited sensing and processing capabilities and additional constraints. In this chapter, we extend an existing appearance-based SLAM system to address these challenges. We introduce a new binary descriptor that combines color, depth, and intensity information to increase the reliability and robustness of the tracking and place recognition steps. We integrate the new descriptor into the state-of-the-art ORB-SLAM system, add additional filtering and geometrical verification steps for better data association, and replace the place recognition module with a modified version of FAB-MAP. In experiments under controlled conditions and in real-world experiments with a Nao humanoid equipped with an RGB-D camera, we show that our new descriptor outperforms established descriptors in precision and recall. Our new descriptor and our extensions to the SLAM system lead to more reliable tracking of features even in image sequences suffering from motion blur. Consequently, the trajectories generated by our SLAM system have lower absolute trajectory errors in comparison to the original ORB-SLAM and loop closures are detected more reliably.*

## 4.1 INTRODUCTION

In the previous chapters, we used 2D representations of the environment for planning our navigation tasks: In Chapter 2, we worked with graph structures representing road networks, and in Chapter 3, we combined topo-metric graphics with occupancy grid maps created by a SLAM approach. 2D representations, however, are not sufficient for truly autonomous robots working in home and office environments, in particular if we would like to leverage the full potential of humanoid robots that can climb stairs, step over obstacles, bend over boxes to inspect their contents, or use tools designed for humans. For these applications, we need approaches for building accurate 3D representations.

In recent years, there has been huge progress in the development of SLAM techniques, for example in the context of self-driving cars. For humanoid robots, however, SLAM is still particularly challenging due to the specific problems inherent with bipedal robots. One of the main challenges with humanoid robots is that they typically cannot carry

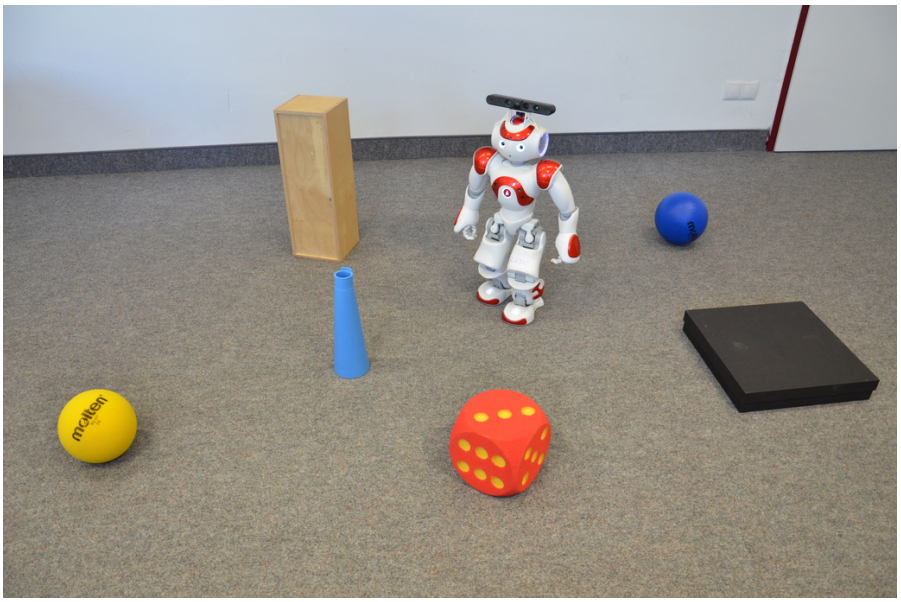



Figure 4.1: Nao robot with an RGB-D camera mounted on its head navigating through an environment cluttered with toys. We present a SLAM system with a feature descriptor that combines depth and color information, allowing the robot to distinguish both texture and shape of objects.

heavy payloads due to limited torque in their lower body motors, balance constraints, and limitations on energy consumption. While cars and wheeled robots are often equipped with multiple LIDAR systems providing 360° scans, humanoid robots have to resort to lightweight and energy-efficient sensors such as webcam-grade cameras or projected stereo RGB-D cameras. The Nao humanoid robots that we use for our experiments (Figure 4.1), for example, have two built-in monocular cameras with a resolution of $1280 \times 960$ pixels and a diagonal field of view of 72.6° (full specifications in Appendix A.1). As the cameras point in different directions, they cannot be used for stereo vision. Additionally, we mount an ASUS Xtion PRO LIVE RGB-D camera with a diagonal field of view of 70° on top of the robot's head (full specifications in Appendix A.2). The narrow opening angles of these cameras as well as the limited depth range of projected stereo cameras make the localization problem more difficult, as usually only a small portion of the environment is visible, leading to the aperture problem where localization is impossible if only parts of an edge or plane are in the field of view, but not their boundaries. Optical cameras are subject to motion blur, which is a frequent problem with humanoid robots as the walking cycle leads to hard impacts when the swing foot hits the ground at the beginning of a double support phase. Bad lighting conditions in indoor environments worsen this issue as longer shutter times are required. The swinging motion of a humanoid's upper body while walking is harder to predict than trajectories of wheeled robots,

in particular constant velocity assumptions are generally not justified on humanoid robots. Due to limited processing power and network bandwidth of embedded systems, it is usually not possible to process the incoming sensor data at high frame rates. In combination, these factors make localization and mapping hard problems for humanoid robots.

As humanoid robots have limited sensing capabilities, it is necessary to extract and combine as much information as possible from the sensor data to get good localization results. In our previous work, we investigated how geometrical features such as planes and edges can be used to detect staircases and align the robot accurately with the next step [128, 129]. While these geometrical approaches work well in cases where the robot interacts with an object of known shape such as a staircase, other approaches are better suited for entirely unknown environments. In this chapter, we use appearance-based methods that extract features from the environment and use these features to recognize previously seen portions of the environment and to stitch together the corresponding environment models in a SLAM approach.

In this context, *features* are interest points together with an abstracted description of their local neighborhood. An example for such a feature could be the corner of an object with the colors and angles of the adjacent patches. Our applications require the features to be:

- *Unique*: The features must be descriptive to avoid ambiguities. Regions with different appearances should produce different descriptors to avoid incorrect localizations.
- *Repeatable*: The same region should always produce similar feature descriptors so that the region can be recognized reliably.
- *Robust*: The feature descriptor should be robust against noise, illumination changes, and different viewing angles.
- *Fast to compute*: For mapping and localization applications, vast amounts of features need to be processed, hence the descriptors must be fast to compute.

Hence, the challenge of designing a good feature descriptor is to trade off between keeping local information that makes the descriptor unique and dropping redundant information to avoid overfitting to a particular observation to make the descriptor robust, repeatable, and fast. In the literature, several feature descriptors already exist. Scale-Invariant Feature Transform (SIFT) [101] and Speeded Up Robust Features (SURF) [14] are two well-established feature detectors and descriptors for textured images, and more recently ORB [146] was introduced as a computationally more efficient alternative. For depth images acquired with stereo cameras, a variety of descriptors has been developed, for a comparison see Hänsch et al. [66]. To achieve

higher uniqueness and robustness, however, information of all available channels should be combined. Color Signature of Histograms of Orientations (CSHOT) [172] and Binary Robust Appearance and Normals Descriptor (BRAND) [117], for example, use both texture and depth information in their descriptors.

In this chapter, we present a novel feature descriptor for RGB-D data that combines depth, intensity, and color information. In contrast to BRAND, it combines depth and color in separate fields instead of merging them in an "OR" operation, making the descriptor less ambiguous. We integrate our new descriptor into the ORB-SLAM system [114], a state-of-the-art visual SLAM system, and adapt the system to better address the particular challenges of humanoid robots. In summary, we add the following modifications:

1. We replace the ORB feature descriptor by our new descriptor that combines color, intensity, and depth information into a binary descriptor.

2. As constant velocity assumptions are not justified on humanoid robots due to the swinging motion of the upper body, predicting the motion of features between frames is not reliable. Hence, we use brute force matching between all extracted features of consecutive images instead and rely on a Random Sample Consensus (RANSAC) filter for geometric validation to remove outliers.

3. We estimate the current camera pose from a sequence of camera poses estimated from previous frames.

4. We replace the place recognizer DBoW2 [55] with FAB-MAP [58] because FAB-MAP produces more correct loop closures than DBoW2 in the experiments for our particular environment. Which one of the two frameworks performs better seems to depend on the dataset, as other research groups have also reported mixed results for the direct comparison of the two approaches, e.g. in [116]. As the original FAB-MAP implementation uses SURF, a continuously valued descriptor, we modified the FAB-MAP implementation to accept our new binary descriptor, which also results in faster computation.

To assess the performance of our approach, we first evaluate the precision and recall metrics of the descriptor under controlled conditions with manually applied image transformations such as translations and rotations of the image. The results show that our new descriptor yields comparable or higher precision while still keeping a high recall level. Afterwards, we evaluate the complete SLAM system with a Nao humanoid robot (see Figure 4.1) in real environments by comparing the trajectory estimated by our system to the ground truth trajectory recorded by an external motion capture system. The results show that our SLAM system is able to reliably track features in image sequences

even if they contain images affected by motion blur, which happens frequently due to the motion cycle of humanoids while walking. Additionally, our new place recognition module has a higher percentage of correctly identified matches. In combination, our system loses track less frequently and produces trajectories with lower absolute error than the original ORB-SLAM implementation.

## 4.2 RELATED WORK

### 4.2.1 *RGB-D SLAM*

Henry et al. [70] and Endres et al. [43] were among the first who developed a 3D mapping system for data acquired with an RGB-D camera. The general idea of these approaches is to combine the matching of features with pose optimization to reduce the error in the estimates after loop closures. Mur-Artal et al. [114, 115] proposed ORB-SLAM, which performs mapping, tracking, relocalization, and loop closure in real-time using ORB features [146]. Figueroa et al. [46] combined visual odometry and KinectFusion [118] to reconstruct indoor scenes using the BRAND descriptor [117].

In our work, we apply a modified version of ORB-SLAM, which tracks sparse features and therefore is not as computationally expensive as methods that run on GPUs [46, 178, 182]. We replaced the ORB descriptor with our new binary descriptor and enhanced the tracking behavior so it can also handle images with poor features, and can recover from cases where the current camera transformation cannot be determined.

Our new descriptor is based on BRAND [117] but we separate the appearance and depth information as mixing them causes ambiguity. Additionally, we change the way depth information is used. We do not use the point clouds and perform fast pixel tests on patches to speed-up the construction of the descriptor.

### 4.2.2 *Appearance-based loop closing*

Cummins et al. developed FAB-MAP [34], which is an appearance-based approach for mapping. It uses the bag-of-words model to decide whether a place is a new location or has been visited before. Hereby, the system uses the observation that some features are more likely to appear together rather than separately.

Gálvez-López and Tardos presented DBoW2 [55], a fast place recognizer that is also based on the bag-of-words approach. To speed up the feature extraction step, the authors initially used the binary feature descriptor BRIEF. Mur-Artal and Tardós then modified DBoW2 to use ORB features [116], which are rotation and scale invariant. ORB-SLAM [114] uses DBoW2 as its place recognition module. In the experiments with

our Nao robot, we experienced that a lot of the images it identifies as similar are in fact of different places. We therefore replaced DBoW2 with FAB-MAP [58] and modified it so that it works with our new descriptor. Since the new descriptor is binary, it is also faster compared to when using SURF, the descriptor originally used by FAB-MAP.

Sünderhauf et al. have presented a method that relies on CNNs for feature extraction, but it is computationally more expensive and requires processing on GPU [167].

### 4.2.3 *SLAM and visual odometry for humanoid robots*

Stasse et al. [162] presented a 3D SLAM system for humanoid robots. The authors combined data from the robot's walking pattern generator with odometry, Inertial Measurement Unit (IMU) data, and visual features from a monocular camera in an Extended Kalman Filter (EKF) framework. Since images taken by humanoid robots can suffer from blur due to the swaying motion during walking, Pretto et al. [139] investigated how to mitigate that effect by developing an approach that only selects highly distinctive features.

Oriolo et al. [123] used the head pose provided by the Parallel Tracking and Mapping (PTAM) algorithm and the torso orientation from the IMU measurements in the correction step of EKF to localize a humanoid robot.

In our work, we use the RGB and depth information to simultaneously localize the robot and map the environment. We match features of consecutive images and apply bundle adjustment to optimize the pose of the robot. We use distinctive frames, i.e., images with their estimated camera poses, to build a pool of locations for the place recognition module, which indicates whether a loop closure is likely to have appeared.

## 4.3 PROPOSED RGB-D FEATURE DESCRIPTOR

Feature descriptors are one of the core components of appearance-based place recognition and SLAM approaches, as the quality of the descriptors determines the quality of the overall result. As we already discussed in the introduction to this chapter, good feature descriptors need to be unique, repeatable, robust, and fast to compute. Hence, we designed a novel feature descriptor with these criteria in mind.

Regarding the time needed to process the descriptors, binary descriptors are generally faster than continuously valued descriptor vectors. SIFT and SURF, for example, produce descriptor vectors with 64 or 128 floating point components. To solve the data association problem of finding corresponding keypoints descriptors in subsequent images, these algorithms calculate the Euclidean distance between descriptor pairs to find the nearest neighbors, which is a time-consuming

operation. Binary descriptors, by contrast, can be compared using fast metrics such as the Hamming distance that counts the bits that differ between two descriptors. Computing the Hamming distance can be implemented very efficiently by applying an XOR operation to determine the bits that differ and by counting the bits with the `popcnt` (population count) instruction that is present in many CPUs, for example in Intel's SSE4.2 instruction set. Hence, we use binary descriptors in our approach.

Our new descriptor is inspired by the Binary Robust Appearance and Normals Descriptor (BRAND) [117] that also combines intensity and depth information. Given a point of interest, BRAND describes the local neighborhood of the point by a binary string of length 256. The value $f_i$ of bit $i$ is computed by comparing two pixels $\mathbf{x}_i, \mathbf{y}_i$ that are chosen in a pattern around the interest point. The pattern is arbitrary but fixed and is pre-computed by drawing locations from a Gaussian distribution centered at the interest point. The value of the bit is computed according to

$$f_i = \begin{cases} 1 & \text{if } (I(\mathbf{x}_i) < I(\mathbf{y}_i)) \vee \tau(\mathbf{x}_i, \mathbf{y}_i) \\ 0 & \text{otherwise,} \end{cases} \tag{4.1}$$

where $I(\cdot)$ is the intensity of the pixel and $\tau(\cdot)$ is a function comparing the normal displacement and surface complexity in the local vicinity of the two pixels. For more details, see the original publication [117].

The main problem with this descriptor is that it combines intensity and depth in an "OR" logical disjunction operation that creates ambiguity as regions may lead to the same descriptor even though only their texture matches, but not the shape, or vice versa. We propose to resolve this ambiguity by partitioning the descriptor and storing depth and intensity cues in separate bits, eliminating the OR operation.

The BRAND descriptor only considers the intensity of each pixel. The color information, however, is also valuable for distinguishing objects based on their texture. As we already use the intensity information, we represent the color information as the $a^*$ and $b^*$ components of the CIE $L^*a^*b^*$ colorspace. The $a^*$ coordinate indicates the position of the color between the complementary colors magenta and green and the $b^*$ coordinate indicates the position between the colors yellow and blue. We use the $L^*a^*b^*$ colorspace because the values are well-defined for desaturated colors in contrast to the hue-saturation-value (HSV) colorspace where the hue is undefined for black, white, and shades of gray.

Combining the discussed components, we define our descriptor as a 256 bit binary descriptor consisting of four parts with 64 bits each representing the depth value $D$, intensity value $L^*$, and the two color components $a^*$ and $b^*$ of the $L^*a^*b^*$ colorspace. Each of the 64 bits again represents the result of a comparison of two patches drawn from an arbitrary, but fixed pattern around the interest point.

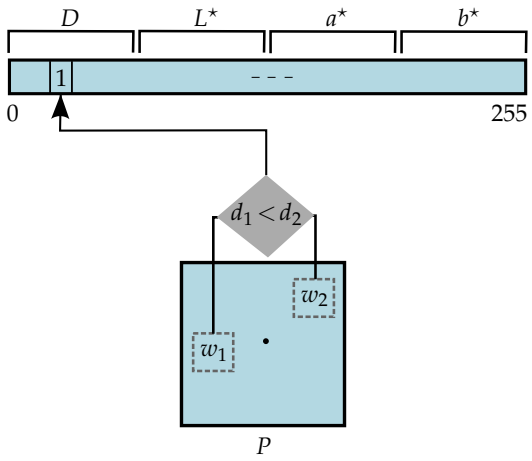



Figure 4.2: Layout of our descriptor. Each field in the binary descriptor is filled out by comparing one pair of windows in a patch $P$. The descriptor is 256 bits long and has equal sections for the depth, intensity, and color channels.

The descriptor needs to be invariant to different viewing angles and distances so that an interest point can be recognized from different poses while the robot walks through the environment. Hence, we apply the same mechanisms that the original BRAND method proposes: Before computing the descriptor, we rotate and scale the patch that we consider for calculating the descriptor to a canonical position. In contrast to descriptors for 2D images where the scale of the patch has to be estimated, we can directly use the measured depth to scale the patch so that it encompasses the same object region irrespective of the distance to the camera. Following the original approach, we scale the patch size between $9 \times 9$ and $48 \times 48$ pixels linearly with the viewing distance. To compute a canonical rotation, we use the Haar wavelet responses to find the dominant orientation as both the BRIEF and SURF descriptors do.

Computing the descriptor for a given interest point is done in the following steps: We first convert the RGB camera image to the $L^*a^*b^*$ colorspace. We then scale and rotate the predefined pattern of pixel pairs according to the measured distance and dominant orientation. For each pixel pair in the pattern, we compare the depth, intensity, $a^*$ and $b^*$ values and fill the corresponding four bits with the test result. To reduce the impact of noise in the data, we do not compare the values of the two pixels directly, but compute the sum of the $9 \times 9$ neighborhood of the pixels and compare the sums. To speed up the

computation of this averaging step, we pre-compute an integral image $S$ once for the whole image according to

$$S(x,y) = \begin{cases} 0 & \text{if } x,y \text{ outside image} \\ v(x,y) + S(x-1,y) & \\ \quad + S(x,y-1) - S(x-1,y-1) & \text{otherwise,} \end{cases} \tag{4.2}$$

where $v(x,y)$ is the value of the image at pixel coordinate $(x,y)$. Summing up 81 values in the $9 \times 9$ sliding window centered at $(x,y)$ then reduces to summing up four components:

$$\sum_{\Delta x=-4}^{4} \sum_{\Delta y=-4}^{4} v(x+\Delta x, y+\Delta y) \tag{4.3}$$

$$= S(x-4,y-4) + S(x+4,y+4) \tag{4.4}$$
$$\quad - S(x+4,y-4) - S(x-4,y+4).$$

Figure 4.2 illustrates the process of constructing the descriptor for a patch of pixels $P$ centered at an interest point. The integral sum value of the depth values in window $w_1$ denoted as $d_1$ is compared to that of the second window $w_2$ denoted as $d_2$. If $d_1$ is smaller than $d_2$, the associated bit in the binary vector is set to 1. In the same way, we also compute the intensity and color components of the descriptor.

## 4.4 APPEARANCE-BASED SLAM

In this thesis, we focus on mobile robots acting in human environments, for example service robots performing tasks in home and office environments, or robots that can autonomously go for grocery shopping in a nearby store. The environments that we have in mind have in common that they are highly dynamic as robots and humans share the same environment, they constantly change during the course of the day and during seasons, and they are large as we expect the robots to work in a whole building or even in an urban neighborhood. Hence, we need SLAM solutions that support robust long-term navigation. Appearance-based SLAM is a class of SLAM systems that meet these requirements well. The core component of appearance-based systems is a database that stores the "appearance" of places. During navigation, the robot queries this database to retrieve a set of places with appearances that are similar to what the robot is currently perceiving, giving the robot a set of hypotheses about its current location. The appearance of a place consists of global features, for example geometrical objects, or of a constellation of smaller features detected in the environment. If the environment changes, for example when objects are removed from the scene, the algorithm can still recognize the place if the overall feature constellation is still intact. This property

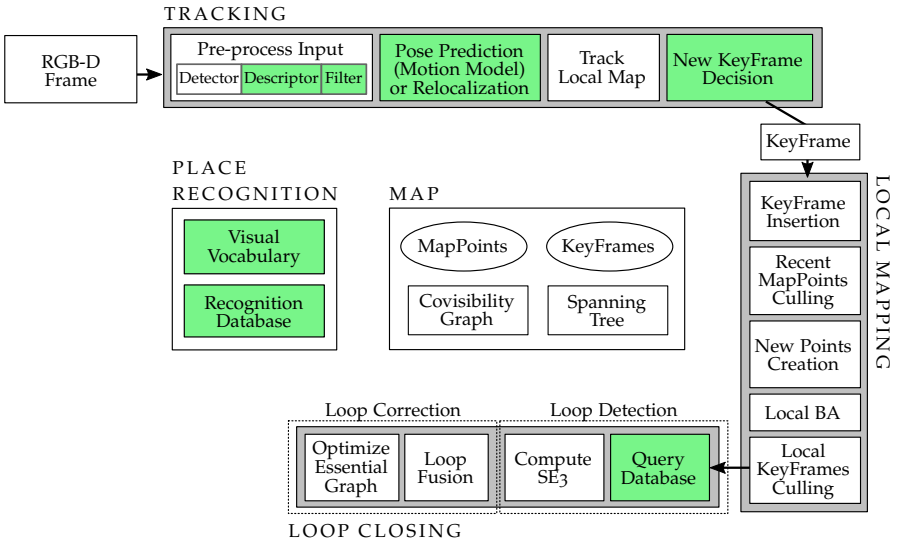



Figure 4.3: Overview of the ORB-SLAM system, adapted from [115]. We modified the parts marked in green to the better match the specific requirements of humanoid robots.

makes appearance-based SLAM robust against dynamic changes in the environment. In contrast to SLAM approaches that create dense maps, appearance-based approaches can be applied on large scales. The authors of FAB-MAP 2.0 [35], for example, demonstrated that their system is capable of finding correct loop closures in a dataset consisting of omnidirectional images recorded along a road network of 1000 km. Several appearance-based SLAM systems have been presented in the literature, see Bacca et al. [11] for a survey.

In this chapter, we analyze ORB-SLAM2, a state-of-the-art SLAM system for monocular, stereo, and RGB-D cameras presented in 2016 by Mur-Artal et al. [114], and extend the system to better suit the requirements of humanoid robots. Figure 4.3 shows an overview of the system with the components that we modified highlighted in green. In the following sections, we will provide more detail on the changes and design decisions for the individual steps of the algorithm.

### 4.4.1 *Pose tracking*

The first step of the algorithm is to pre-process the raw input data received from the RGB-D camera. To detect interest points, we keep the extended FAST algorithm that the original ORB implementation uses, but we exchange the feature descriptor for our new binary descriptor introduced in Section 4.3.

The original ORB-SLAM approach then tries to track the extracted features over a sequence of camera frames by predicting their position with a motion model and searching for matching feature descriptors

in a window around the predicted position. While constant velocity or constant acceleration models yield useful predictions for wheeled platforms, aerial vehicles, or (to a certain extent) hand-held cameras, these predictions are often not accurate enough when applied to humanoid robots. The walking cycle of bipedal robots leads to swinging motions of the upper body while moving forward and irregular motion patterns when turning on the spot. Due to the limited processing power and bandwidth, the camera frame rate is typically low, so that subsequent images are often captured during different phases of the walking cycle. Hence, the motion between frames is difficult to predict. Additionally, the impact of the swing foot on the ground also affects the camera, especially in robots such as the Nao robots that do not have mechanisms for damping the impact. This impact results in motion blur and jittering of the camera. As a result, the displacement of features in subsequent images is hard to predict. In the original approach, this often leads to the case where the data association fails, forcing ORB-SLAM to restart with a new trajectory segment. The system can only recover if it finds a loop closure between the two segments later. Hence, we do not use a motion model for tracking. Instead, we try to match all pairs of feature descriptors to find data association candidates. As the Hamming distance that we use to compare descriptors is fast to compute, this process is still reasonably fast.

To improve the quality of the data association, we introduce filters to reduce the features to the ones that are the most distinctive, hoping that these features will lead to correct data associations. We only accept features that are mutually the best match, i.e., a pair of features $(q, a)$ with $q$ from the current image and $a$ from the previous image is considered a match if $a$ is the best match for $q$ among all features in the previous image, and $q$ is the best match for $a$ among all features in the current image. If this is the case, we add another filter step: We only accept $(q, a)$ as a valid feature pair if the distance to the second best match $b$ in the previous image is substantially bigger than the distance to $a$, i.e.

$$d_{qa} < r \cdot d_{qb}, \tag{4.5}$$

where $r$ is a fixed ratio and $d_{qa}, d_{qb}$ are the corresponding Hamming distances. Feature pairs that pass both filters are probably unique and descriptive.

The two filters added so far work on pairs of features from subsequent frames without considering the other features. As a third filter step, we hence use a Random Sample Consensus (RANSAC) method to verify that the feature pairs follow a consistent geometric transformation, similarly to the approach used by Endres et al. [43]. We project the features back from 2D image space to 3D space using the depth information provided by the camera. We randomly select a subset of feature pairs and compute the transformation between the

selected feature locations in the previous and current frame using a Singular Value Decomposition (SVD). We then count the number of feature pairs that support this transformation hypothesis, called *inliers*. A feature pair is considered an inlier if the Euclidean distance between the projection of the old feature according to the computed transformation and the new feature detected in the current frame is below a threshold. After running multiple sampling iterations, we keep the transformation hypothesis that is supported by the most inliers.

The largest set of inliers is then used to estimate the motion of the camera between the two images by optimizing the pose with g²o [93] using a variant of local bundle adjustment where the points are fixed and the pose is optimized.

In some cases, no valid transformation between subsequent frames can be found, for example if the image suffers from motion blur that makes recognizing features impossible, if the data association fails due to self-similarities in the environment, or if no features are available, for example if only a plain wall is in the robot's field of view. To recover from short-term failures, we keep a history of the last five frames for which a valid transformation was found. If the algorithm is unable to find a transformation between the previous frame and the current frame, we try to compute the transformation using earlier frames instead, going back in time. If the algorithm still does not find a transformation, we skip the image.

### 4.4.2 *Mapping*

After tracking the robot's pose locally, the second component of the SLAM system is the mapping component. As keeping the information of all frames is infeasible, the algorithm has to decide which frames and their corresponding transformations to keep. These frames are called *keyframes*. The mapping algorithm represents these keyframes as nodes of a graph and the corresponding transformations between the keyframes as edges between the nodes. From this graph representation, the mapper reconstructs a sparse representation of the environment. The keyframes are selected based on the estimated error of the transformation between the images and the number of usable features.

While dropping frames is necessary to keep the algorithm efficient, selecting appropriate keyframes is difficult. When the robot returns to the same location later, it may fail to detect the loop closure if the corresponding frames were dropped and the nearest keyframes are too far away. Hence, we first keep all frames with a valid transformation as keyframes to increase the chances that loop closures get detected. As the robot explores the environment, the mapper accumulates the 3D points from the frames. At regular intervals, we perform bundle

adjustment on the local point cloud, merge identical points, and drop redundant keyframes. Hence, we only drop keyframes that overlap in large parts with other keyframes so that loop closures can still be reliably detected. In contrast to the original implementation, we run the mapping component synchronized with the pose tracking component every 10 keyframes to keep the size of the local maps approximately equal, leading to more consistent coverage of the map.

### 4.4.3 *Place recognition for loop closing*

The third major component of SLAM systems is detecting loop closures. Appearance-based methods use a database for storing places within the map together with the overall "appearance" of the place. The appearance of a place is a constellation of local features, often representing the geometric arrangement of multiple objects present in the place. These constellations can be represented in a Bag of Words (BoW) approach where a *codebook* for a particular type of environment is created in an offline step. The *words* in this vocabulary consist of cluster centers representing clusters of visual features detected in the training images. This learned vocabulary can then be used to describe places in new environments by choosing a subset of words that describe a place.

The original ORB-SLAM approach that we base our system on uses Dynamic Bag of Words (DBoW2) [116] as the place recognition module. In preliminary tests with datasets recorded in our experimental environment, however, we found that DBoW2 produces a substantial amount of false positive loop detections. Such incorrect loop closures are detrimental for SLAM as they can irrecoverably destroy the layout of the map. FAB-MAP [58], by contrast, produced less false positives. Hence, we replaced DBoW2 by FAB-MAP as the place recognition module.

As the original FAB-MAP implementation uses SURF which produces continuously valued feature vectors, we modified the approach to work with our new binary descriptor. Hence, we replaced the Euclidean similarity metric for assigning features to clusters in the training phase by the Hamming distance which counts the number of bits in which two descriptors differ.

After assigning feature descriptors to clusters for creating the vocabulary, the original approach computes the mean of the assigned descriptors to determine the cluster center. For binary strings, however, there is no meaningful definition of an "average". Hence, we follow the approach from [55] to replace the computation of the cluster center by a majority voting scheme. For each bit of the descriptor, each member descriptor of the cluster votes for either one or zero in that bit position. After voting, the algorithm assigns ones to the bit positions

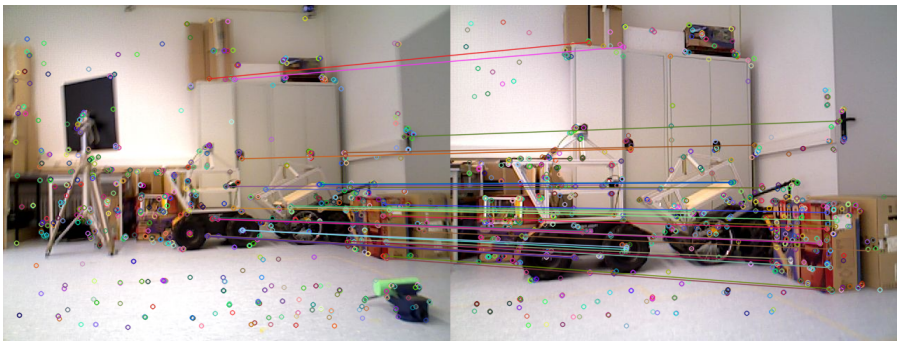


Figure 4.4: The robot revisits an old place and FAB-MAP with our descriptor returns a possible candidate for loop closure. If there are enough matched features after RANSAC as in this example, a loop closure takes place.

where the majority of member descriptors voted for one, and zeros to the other bit positions.

Whenever the pose tracker creates a new keyframe, we compute the BoW descriptor of the keyframe and query the database for places with similar appearance. If a candidate for a matching place is found, we verify whether there is a consistent geometric transformation between the feature pairs of the corresponding frames. We again use the RANSAC algorithm described in Section 4.4.1 to check whether a transformation exists and enough feature pairs support the transformation as inliers. If the detected loop closure passes the geometric validation, we optimize the transformation between the frames by minimizing the reprojection error as we did in the tracking phase and add the loop closure edge to the graph representing the map.

Afterwards, we add the BoW descriptor of the new keyframe to the database if at least 10 frames have passed since the last added keyframe to avoid adding redundant key frames with negligible visual change.

The algorithm then applies a g²o [93] optimization run to optimize the graph representing the map where each node represents a camera pose and the edges represent transformations between subsequent frames or loop closure frames. The optimization step distributes the error along the loops, thus decreasing the local errors.

## 4.5 EXPERIMENTAL EVALUATION

We evaluate our approach on real-world datasets in three steps: First, we evaluate the robustness of our new binary descriptor to changes in the camera perspective in a controlled environment. We transform RGB-D images of real-world scenes and show that the descriptor is invariant to translations and rotations in the image. We show that

our descriptor performs better than existing descriptors in terms of precision and recall.

Second, we run our modified SLAM system including our new descriptor on real-world datasets recorded with a Nao humanoid robot and compare the resulting trajectories to the ground truth trajectories recorded with a motion capture system. The results show that our approach leads to lower absolute trajectory errors.

Third, we provide a comparison of the run times of the different descriptors that shows that our novel descriptor is substantially faster than SIFT, SURF, and BRAND and almost as fast as ORB.

### 4.5.1 *Evaluation of descriptors*

For tracking features in a sequence of frames, the pose tracker of our SLAM system has to solve a data association problem: Which features in the previous and current frame represent the same location in the physical world? The problem of finding corresponding features can be interpreted as an instance of the information retrieval problem. Given a database of feature descriptors from the previous frame, the algorithm queries the database for records matching the feature descriptors of the current frame. The database then returns a set of descriptor pairs. In the optimal case, the system is *precise*, meaning that all returned descriptor pairs actually originate from the same location in the real world, and *efficient*, meaning that it returns all such pairs that exist in the database. In reality, however, the retrieval is usually not perfect so that the retrieval returns *false positives*, i.e., descriptor pairs that originate from different locations in the environment, or it omits descriptor pairs that actually match (a *false negative*). The quality of an information retrieval procedure is commonly evaluated with the two measures *precision* and *recall*:

$$\text{precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}} \tag{4.6}$$

$$= \frac{\text{correctly returned matches}}{\text{all returned matches}} \tag{4.7}$$

$$\text{recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}} \tag{4.8}$$

$$= \frac{\text{correctly returned matches}}{\text{all correct matches in the database}} \tag{4.9}$$

$$\tag{4.10}$$

Hence, *precision* is the percentage of returned pairs that are actually correct matches, and *recall* is the percentage of correct matches retrieved from the database.

In our application, we order all pairs of feature descriptors by their Hamming distance, remove matches with distances above a threshold,

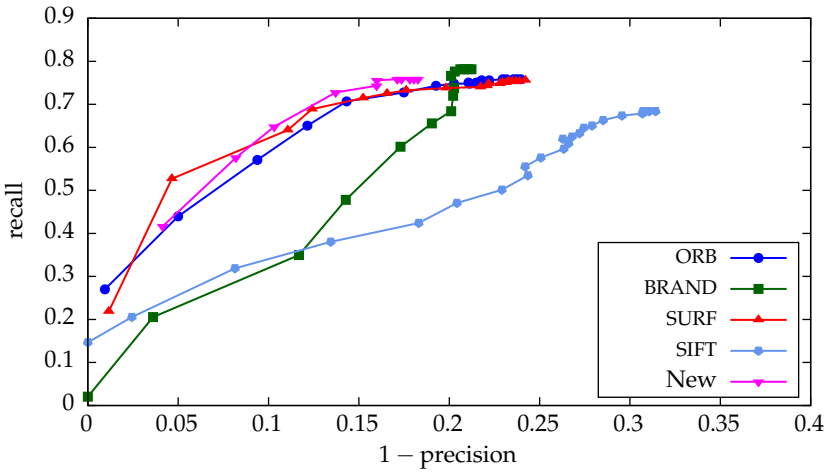



Figure 4.5: Precision-recall diagram for evaluating the invariance of descriptors when transforming images with a translation of 10 pixels. The points along a curve represent different thresholds on the similarity metric. Our descriptor performs best since it produces the highest recall and precision (note that the horizontal axis shows 1 − precision).

and filter out unlikely or uncertain matches in three filter steps as described in Section 4.4.1 before passing the remaining descriptor pairs to the algorithm for estimating the camera transform based on these matches. By changing the thresholds and filter parameters such as the distance ratio $r$ in Equation 4.5, we can influence the balance between precision and recall. Returning more pairs increases the recall up to the trivial solution of returning all pairs, leading to a recall of 1 by definition. But returning more pairs generally also lowers precision, as returning less likely matches increases the risk of including incorrect matches. Hence, we have to find a trade-off between precision and recall. In SLAM applications, precision is usually preferred over recall as incorrect loop closures are detrimental to the map quality and may lead to situations that the robot cannot recover from. If the recall is too low, however, the algorithm may fail to compute transformations between camera frames, leading to disconnected local maps.

For the experimental procedure for determining precision and recall, we follow the approaches used by Nascimento et al. [117] and Rublee et al. [146]. For a set of images captured with an RGB-D camera in a real world scene, we first detect interest points and calculate the corresponding descriptors. We then apply geometrical transformations such as translations and rotations to the images to create a new set of images. As we transform the images manually, we know the ground truth for the expected feature locations, data associations, and maximum number of potential correct feature matches inside the image boundaries. For the experiments, we used 1000 randomly selected images from the benchmark dataset provided by Sturm et al. [165]. We

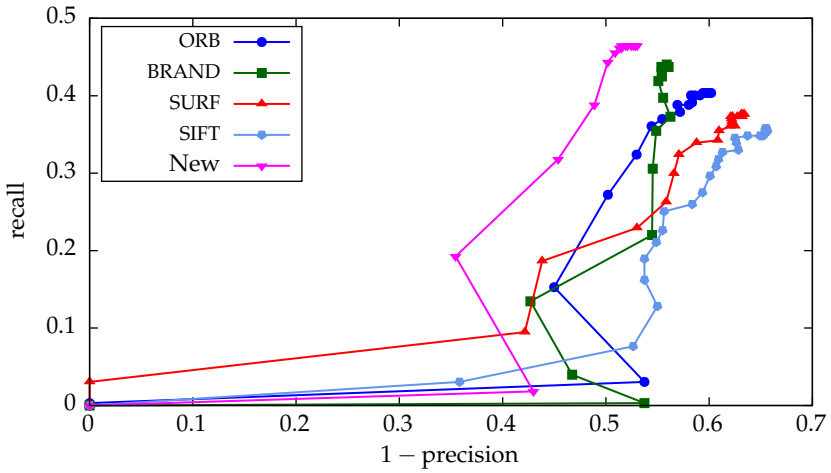



Figure 4.6: Precision-recall diagram when transforming the images with a rotation of 30°. The precision is lower than for translation-only transforms due to discretization into a pixel raster. Again, our new descriptor has the highest precision among the descriptors.

set the parameters of the detector to choose 500 interest points per image and generated the corresponding feature descriptors with our new descriptor and four existing descriptors for comparison. We then transformed the images, re-run the detector and descriptors, and searched for matching features with the procedure from Section 4.4.1 including the three filter stages. We validated the returned matches against the projection of the features of the original frame with the ground truth transformation to determine the number of correct matches for calculating the precision and recall metrics.

Figure 4.5 shows the result for a translation of 10 pixels and Figure 4.6 shows the result for a rotation of 30°. Note that the horizontal axis shows $1 - \text{precision}$ for better readability and consistency with the results presented in [117], so a system with optimal precision and recall would produce a data point in the top left corner. In both cases, we varied the threshold on the Hamming distance of acceptable pairs to shift the balance between precision and recall. Each point in the resulting curve represents the precision and recall at a certain threshold. The range for the threshold is 0– 255 in steps of 5 for the binary descriptors ORB, BRAND, and our new descriptor. For the continuously valued descriptors with Euclidean similarity metric, we chose a range of 0– 10000 in steps of 100 for SIFT and 0–10 in steps of 0.1 for SURF. As expected, increasing the threshold increases the recall and decreases precision. As can be seen, our descriptor performs best in comparison to the other descriptors under both translation and rotation transformations. The results show that the depth cue is valuable for differentiating features and that resolving the ambiguity between intensity and depth cues leads to better performance in comparison to BRAND.

| | Frames | Candidates | Correct |
|---|---|---|---|
| | Dataset 1 | | |
| FAB-MAP-New | 2534 | 1805 | 1805 |
| FAB-MAP-ORB | 2534 | 1876 | 1862 |
| DBoW2 | 2534 | 2038 | 799 |
| | Dataset 2 | | |
| FAB-MAP-New | 3160 | 2124 | 2124 |
| FAB-MAP-ORB | 3160 | 2280 | 2266 |
| DBoW2 | 3160 | 2360 | 430 |

Table 4.1: Place Recognition Results.

Note that our experimental setup for these results also includes the detector stage. We run the detector separately on the original image and the transformed image with a fixed number of interest points to detect. As the image transformation leads to some parts of the image being clipped, the detected interest points are inevitably different. The detector might also detect interest points at slightly different locations as a result of the discretization of the image. This effect also lowers precision and recall.

### 4.5.2 *Place recognition*

Recognizing places that have been visited before is the core component of all SLAM systems as loop closures allow the system to correct accumulated errors and to generate more accurate maps. In appearance-based SLAM, place recognition is done by extracting features in the current camera image, computing a bag of words descriptor according to a pre-trained vocabulary, and querying a database for places with similar combination of words. In this work, we compare two appearance-based place recognition systems: Dynamic Bag of Words (DBoW2) as used in the original ORB-SLAM implementation and our modified version of FAB-MAP working with our new binary descriptor as explained in Section 4.4.3. To evaluate these approaches, we recorded datasets with a Nao humanoid robot walking in an indoor environment and queried the two place recognition modules for places that the robot has seen previously along the trajectory. Table 4.1 shows the results for two datasets. The first column is the total number of frames recorded along the trajectory. The second column shows the number of frames that the system has recognized as frames from previously seen locations, and the last column indicates how many of those candidates are correct according to manual labeling. As FAB-MAP

returns images that potentially match the query image together with confidence values, we follow [34] and only consider frames with a confidence value of 0.98 or higher as candidates.

The results in Table 4.1 show that although using FAB-MAP with ORB leads to higher detection rates than when using the new descriptor, the percentage of correct detections is lower. As discussed above, incorrect loop closures are harmful for SLAM as they might destroy the layout of the map to the point where the robot cannot localize anymore and thus cannot recover. On the other hand, missing a loop closure is less critical, as usually several subsequent frames show the same place and a low number of matching images is enough to correct the map with loop closures.

Manual inspection of the matches returned erroneously by FAB-MAP with ORB reveals that the main cause is the lack of distinctive features. Due to the small opening angle of the cameras, it frequently happens that the portion of the scene that is in the robot's field of view does not provide enough diversity for recognizing the place. Other causes of incorrectly returned matches are occlusions by people moving in front of the robot and images affected by motion blur.

The original implementation of ORB-SLAM uses DBoW2 as its place recognition component. The last column in Table 4.1 shows that DBoW2 generates a large number of false positives, hence this approach is less suitable for SLAM.

### 4.5.3 *Simultaneous localization and mapping*

To evaluate the performance of the overall SLAM system, we reconstructed the robot's trajectory with the SLAM trajectory and compared the trajectory to the ground truth provided by a motion capture system. As the error metric, we compute the Absolute Trajectory Error (ATE):

$$\mathrm{ATE} = \sqrt{\frac{\sum_N \|X_t - G_t\|^2}{N}}, \tag{4.11}$$

where $X_t$ and $G_t$ are the estimated and ground truth poses respectively at time $t$, and $N$ is the number of pose estimates. We computed the ATE using the benchmark evaluation utility from [165]. As the transformation between the world frames of the motion capture system and the frame of the SLAM system is unknown, the utility aligns the trajectories in a least-squares optimization. The resulting ATE is a measure for the global consistency of the estimated trajectory.

Table 4.2 reports the results for four different trajectories recorded with a Nao robot with an ASUS Xtion camera mounted on top, as well as the results for a dataset recorded with a wheeled Pioneer platform that is provided by the RGB-D benchmark dataset from [165].

Figure 4.7 shows the corresponding estimated and ground truth trajectories for these datasets. The top two rows show that our system

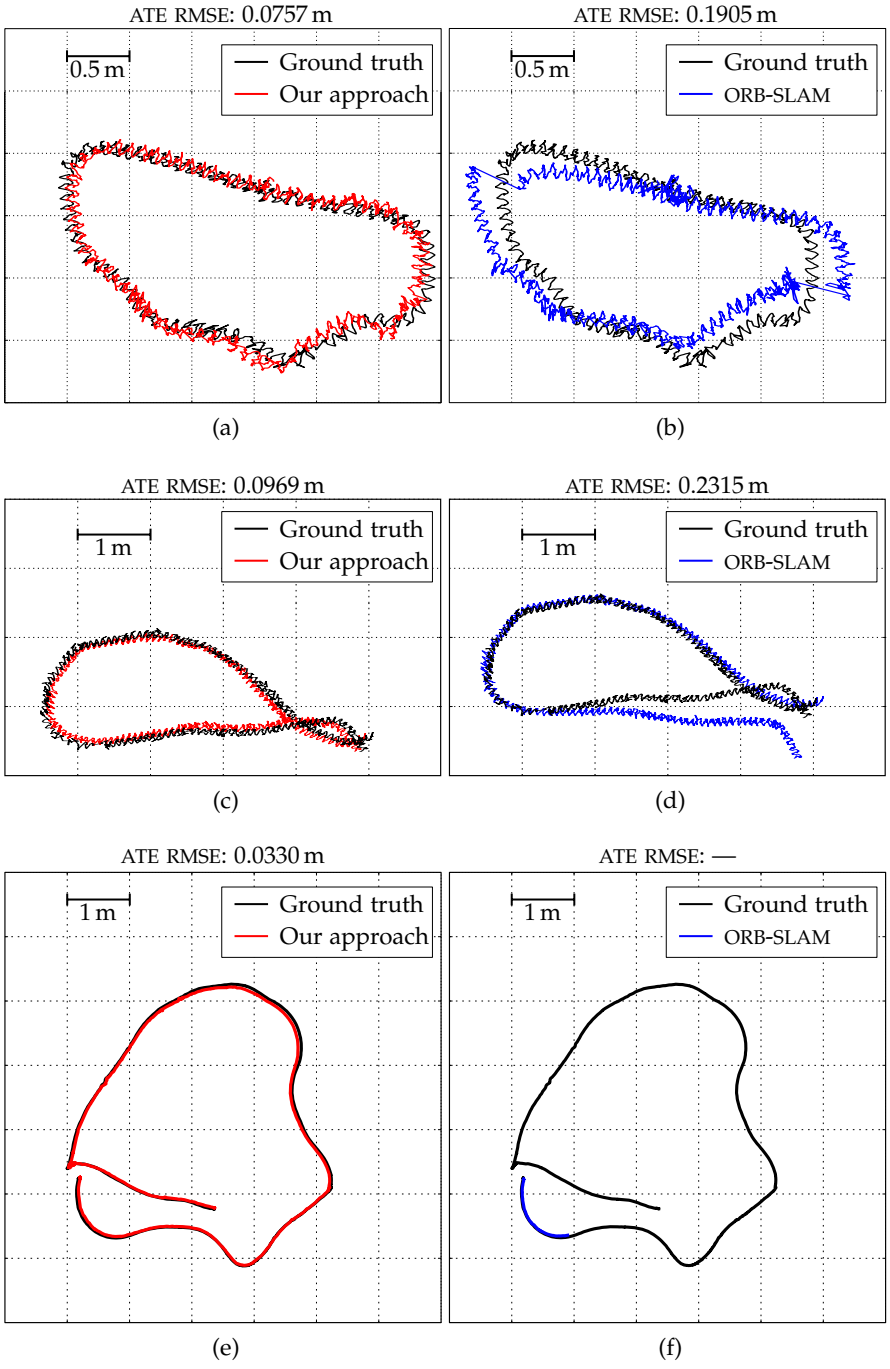



Figure 4.7: SLAM results on several datasets. Our system (left column) tracks the robot accurately and the ATE is lower than when using ORB-SLAM (right column). The top two rows show the results for data recorded with the Nao robot (datasets Nao1 and Nao2 in Table 4.2), whereas the last row shows the results for a benchmark dataset of a Pioneer wheeled robot.

| Datasets | Nao1 | Nao2 | Nao3 | Nao4 | Pioneer |
|---|---|---|---|---|---|
| Our system | 0.08 m | 0.1 m | 0.04 m | 0.05 m | 0.03 m |
| ORB-SLAM | 0.19 m | 0.23 m | 0.04 m | 0.09 m | — |

Table 4.2: Absolute Trajectory Error.

| Descriptor | ORB | Ours | SURF | BRAND | SIFT |
|---|---|---|---|---|---|
| Time | 4 ms | 14 ms | 94 ms | 125 ms | 132 ms |

Table 4.3: Time to compute 100 descriptors of each type.

performs well with data recorded with the Nao robot and outperforms ORB-SLAM. Pose tracking with legged robots such as the Nao is more challenging than with wheeled robots since the movement of the head can result in blurry images that may not have well-defined features. The last row of Figure 4.7 shows the estimated and the ground truth trajectory for the Pioneer dataset. In contrast to ORB-SLAM which is not able to track the robots pose, our system accurately tracks the trajectory and performs loop closing.

As can be seen from Figure 4.7 (f), ORB-SLAM cannot track the pose as the robot makes a turn. ORB-SLAM first uses the motion model from the previous frames to initialize the new pose. This new pose, however, is not supported by enough matches so it then tries to compute the transformation from the keyframe of the local map. Since there was a rotation in the movement of the robot, the reference keyframe no longer shares enough points with the current frame and matching fails.

### 4.5.4 *Computational cost*

Computing the descriptors has a major share in the overall computation time of the SLAM system as it has to compute a large number of descriptors for every input frame. Table 4.3 shows a comparison of the computation times for the different descriptors. The table shows the time needed to create 100 descriptors of each type on a single Intel Core Pentium 987 CPU. While ORB descriptors are the fastest to compute, they contain less information as they do not consider depth. Our new descriptor is significantly faster than BRAND which also considers depth information. The main speed-up is attributed to how the depth information is used. We compare depth values directly, whereas BRAND computes normals from the local neighborhood of the interest points and uses them in the geometric tests. Our descriptor, however, is both faster and better in terms of precision and recall as shown in Section 4.5.1.

| Step | Time |
|---|---|
| $L^*a^*b^*$ colorspace conversion | 0.012 ms |
| Integral image computation | 0.011 ms |
| Canonical orientation computation | 0.073 ms |
| Color, depth, and intensity tests | 0.048 ms |
| Total | 0.144 ms |

Table 4.4: Breakdown of processing time needed to create one instance of our descriptor.

The more detailed breakdown of the run time of our descriptor in Table 4.4 shows that the most expensive step is the computation of the patch orientation. It uses the same method as in SURF where the Haar wavelet responses are computed and summed in the $x$ and $y$ directions.

## 4.6 DISCUSSION

Our work presented in this chapter is a step towards robust SLAM systems for humanoid robots with their limited sensing capabilities and particular challenges arising from the complex motion patterns. We showed that our improvements to existing appearance-based SLAM systems leads to higher precision and recall in the feature matching step and thus more robust and accurate tracking of the robot. There are, however, still situations where tracking fails, in particular when there are not enough descriptive features in the robot's field of view or when bad lighting conditions disturb the recognition of features. To improve the results in these situations, it might be worth to integrate more information available to the robot such as the kinematic walking odometry, IMU data, and sonar measurements. Integration of these measurements, however, need well-calibrated sensor models as these measurements are typically rather inaccurate and subject to drift. To avoid blurry images, capturing images could be synchronized with the walking cycle so that the robot takes the images in phases with less motion and jitter.

Even when integrating all available information, there might still be situations where there are not enough features to track. In these situations, the robot could actively perform actions for recovering, for example by standing still to avoid motion blur or turning its head to search for features outside its regular field of view. In our previous work [126], we presented a reinforcement learning approach for learning when to perform such actions by trading off between

localization uncertainty and time required for the navigation task, and which actions to perform for successfully arriving at the goal location. Integrating such a learned active recovery policy into a SLAM system would lead to a more robust system.

The experiments in this chapter were performed on a single core CPU. There is a lot of potential for parallelization both by multi-threading and by exploiting Single Instruction Multiple Data (SIMD) architectures. In Chapter 6, we show that many navigation tasks can be formulated as rendering problems that can be solved highly parallelized on dedicated graphics cards. It may be worth to investigate which parts of the computation and evaluation of feature descriptors could benefit from offloading to a GPU.

Feature descriptors are not only used in appearance-based SLAM, but also in other robotics applications. Most prominently, robust features can be used for object detection and classification in images, for example for semantic scene classification or when searching for objects in an environment.

## 4.7 CONCLUSIONS

In this chapter, we adapted and improved an existing appearance-based SLAM system to improve its accuracy and robustness on humanoid robots. SLAM on humanoid robots is particularly challenging due to irregular motion patterns originating from the walking cycle that are hard to predict and due to the limited sensing capabilities. We developed a novel binary descriptor that extracts more information from the input RGB-D data by combining intensity, depth, and color information from patches around interest points in an unambiguous way. We presented modifications to the existing ORB-SLAM system to make it more robust by adding filtering, improving geometric validation, and replacing the place recognition module with a modified version of FAB-MAP that uses our binary descriptor.

We thoroughly evaluated our approach on real-world datasets. First, we compared the precision and recall metrics of our new descriptor to four existing descriptors and showed that our novel descriptor has a higher precision with comparable recall. As avoiding incorrect loop closures is crucial in SLAM to prevent irrecoverable damage to the constructed map, the high precision of our descriptor makes it especially useful for SLAM.

Second, we compared the accuracy of the loop closure step of our modified system with existing approaches. The results showed that the percentage of correct loop closure candidates generated by the place recognition module is higher with our proposed system than with existing approaches using DBoW2 or FAB-MAP with ORB descriptors.

Finally, we evaluated the overall performance of our SLAM system on real-world data sets recorded with an RGB-D camera mounted on a

Nao humanoid robot and on benchmark datasets for a wheeled robot. The results demonstrate that our system outperforms ORB-SLAM as it follows the ground truth trajectory more closely and has a lower absolute trajectory error. Our modifications make the SLAM system robust even in sequences with blurred images, which frequently occur with walking humanoids.

# 5

# EFFICIENT COVERAGE OF 3D ENVIRONMENTS WITH HUMANOID ROBOTS USING INVERSE REACHABILITY MAPS

*Covering a known 3D environment with a robot's camera is a commonly required task, for example in inspection and surveillance, mapping, or object search applications. In addition to the problem of finding a complete and efficient set of view points for covering the whole environment, humanoid robots also need to observe balance, energy, and kinematic constraints for reaching the desired view poses. In this chapter, we approach this high-dimensional planning problem by introducing a novel inverse reachability map representation that can be used for fast pose generation and combine it with a next-best-view algorithm. We implemented our approach in ROS and tested it with a Nao robot on both simulated and real-world scenes. The experiments show that our approach enables the humanoid to efficiently cover room-sized environments with its camera.*

## 5.1 INTRODUCTION

In Chapter 3, we investigated the problem of how a mobile robot can explore an unknown environment as efficiently as possible if the user provides background knowledge about the topological layout of the environment. We presented an approach that solves a traveling salesman problem to generate a tour for visiting all areas so that the robot can build a full model of the environment.

In this chapter, we will turn to the closely related problem of *covering a known environment* with the robot's sensors. The goal is again to generate a tour for observing as much as possible of the environment. This time, however, the robot has access to a full 3D model of the environment instead of just an abstract topo-metric graph. We assume that either the robot has captured this 3D model previously with a SLAM approach, or that the human has given the model to the robot beforehand as a CAD model. In either case, the robot's task is to use the given map to calculate a tour through the environment from where it can cover the relevant areas with its camera as efficiently as possible. This problem is relevant for several real-world applications such as searching a lost object in an apartment, taking stock in a grocery store, inspecting technical installations for damage, or re-mapping an environment to incorporate changes into the map.

Figure 5.1: Nao inspecting an environment. The robot bends over to peek into a box for completing the task of completely covering the environment with its camera, e.g., for finding objects.

Coverage planning is a problem that has been investigated before, for example in the context of RoboCup Rescue where the robot has to find victims in a disaster scene, in the context of cleaning robots that have to plan a tour for vacuuming a floor or wiping a table, or in the context of patrol robots that should guard a building. What sets our approach apart from previous approaches is that we focus on *humanoid robots* operating in *human environments*. Humanoid robots are particularly well suited for executing inspection tasks in human environments, as they can bend over to look into boxes like the robot in Figure 5.1, bend down to peek below tables and chairs, and open drawers and cupboards to inspect their contents. As humanoid robots have a similar body plan as humans, they can typically reach regions that are interesting to humans and operate facilities designed for humans, making them ideal for collaborating with human users.

Before we will turn to articulated environments where the robot has to actively move and manipulate objects in the next chapter, we first focus on static scenes. In this chapter, we present an approach for covering a known environment with the camera of a humanoid robot by integrating view point planning with whole-body motion planning.

Our approach adapts the next-best-view algorithm for 3D coverage originally presented by Dornhege et al. [41] to meet the requirements of humanoid robots. The original approach has been used in RoboCup Rescue disaster scenes on tracked vehicles with cameras mounted on a robotic arm with six degrees of freedom. This setup allows

the robot to position its camera freely within a spherical working volume with radius 1 m around its base position. Humanoids with head-mounted cameras, by contrast, have a much smaller reachable volume to place the camera due to the kinematic limitations of the robot. To move the camera, the robot can turn and tilt its head, bend over with its whole upper body to peek downwards, or arch its back to look upwards. As a result, a humanoid robot can place its camera only within a small, irregularly shaped reachable volume (see Figure 5.2 on page 86 for an illustration). While the original approach uses sampling methods to determine camera positions within the reachable volume for computing valid pairs of endeffector and base poses, the smaller reachability volume renders these sampling methods inefficient for humanoid robots.

Additionally, using such whole-body movements to position the camera is difficult as the robot has to maintain its balance at all times to avoid falling over. Bending the upper body forward is particularly strenuous for the hip, leg, and ankle joints, increasing both the energy consumption and the risk of overheating these joints. Hence, the robot has to consider walking and pose stability, self-collisions and collisions with the environment, energy consumption, and overheating when planning view poses for observing the scene. Due to the high degree of freedom, planning whole-body postures while searching next-best-view poses is computationally highly expensive.

To cope with these additional constraints to the planning problem introduced by humanoids, we propose to use pre-computed Inverse Reachability Maps (IRMs) that can be queried efficiently while planning the view points of the coverage tour. In a first step, our approach generates promising view poses by casting rays from surfaces and sampling candidate camera poses for free-space voxels where many of these rays pass through. Afterwards, we use the IRM to evaluate possible whole-body configurations to reach the views. As a result, we get a set of camera poses that cover the environment, including suitable foot poses and whole-body configurations for reaching these camera poses. By computing the shortest paths for navigating between all pairs of robot poses, we compute a graph and use a Traveling Salesman Problem (TSP) solver to find the shortest connecting tour. The formulation as a TSP instance is similar to the approach for exploration with background knowledge that we presented in Section 3.3.2. In the current chapter, however, we derive the node positions and the graph automatically from a 3D map instead of relying on a user-provided graph.

To validate our approach, we performed experiments in both simulation and real-world scenarios with a Nao robot. Our results show that our system enables the humanoid to successfully and efficiently inspect home-like environments covering all interesting surfaces.

## 5.2 RELATED WORK

Finding view points from where a whole known or unknown scene can be observed is a well-known, challenging problem in both robotics and computer graphics. A large number of applications need to solve this problem, including autonomous exploration, autonomous scanning and reconstruction of 3D objects, and coverage and surveillance tasks.

The optimization problem of finding the minimum number of viewing points required for observing a known environment has been formulated as the *art gallery problem*, which asks for the positions where guards or CCTV cameras have to be placed for monitoring an art gallery with a polygonal floor plan in 2D or a polyhedral model in 3D. The art gallery problem is known to be NP-hard and APX-hard even in 2D environments [124].

Stasse et al. [163] and Foissotte et al. [48, 49] presented a two-step approach for exploration and coverage with a humanoid robot. In the first step, a next-best-view algorithm is used for finding suitable view poses. In the second step, a posture generator tries to find postures to reach the desired view poses. The two steps are alternated in a greedy iterative scheme. Separating view pose planning and pose planning has the disadvantage that collision checks, stability constraints, and energy optimization cannot be considered while optimizing the view points. Generating a pose for every candidate view point is infeasible. We overcome this problem by pre-computing an inverse reachability map that can be queried fast enough to be used in the view point planning stage.

In the past, several variations of next-best-view algorithms for finding a good sequences of view points to observe a scene have been proposed. Bissmarck et al. [21] published a run-time comparison of some existing solutions. Next-best-view algorithms have also been successfully applied to find view points for 3D reconstruction using in-hand manipulation [89], using cameras mounted on robotic arms with a fixed base [90], and using unmanned aerial vehicles [19]. Our scenario, however, requires a humanoid robot to walk around in the environment, which makes planning more difficult as balancing constraints and pose optimization have to be considered. In contrast to approaches that reduce the problem complexity by limiting view point candidates to convex hulls [133] or bounding spheres [175] surrounding the objects of interest, we sample candidate view poses in the whole volume that the robot can reach.

While we focus on the task of covering a known environment completely, the general framework can also be used for autonomous exploration as in the work of Dornhege and Kleiner [40]. In the autonomous exploration task, covering the known surfaces of the environment is replaced by covering the *frontiers* between known and unknown regions, as we discussed in our exploration with background knowledge

approach in Section 3.3.4. The information gain of a view point is estimated based on the size of the unknown voids in the field of view. Daudelin and Campbell [38] propose a probabilistic extension of the work by Isler et al. [79], which consider the information gain for each cell in the vicinity of frontier surfaces for computing the next best view.

In Chapter 3, we introduced an approach for speeding up exploration tasks by exploiting background knowledge. Based on a topological graph provided by the user, the robot computes a global exploration strategy using a traveling salesman problem solver. This global strategy can be combined with a local exploration strategy determined with the approach we present in this chapter.

Burget and Bennewitz [28] applied inverse reachability maps for selecting suitable stance poses of a humanoid for grasping tasks. This application requires high maneuverability of the endeffector, and hence the authors use a manipulability measure based on the Jacobian matrix of the kinematic chain. For our coverage task, high maneuverability is not needed and we evaluate poses based on stability, energy consumption, and required time for reaching the whole-body pose instead.

## 5.3 PROBLEM DESCRIPTION AND FRAMEWORK

Our goal is to completely cover a known environment with the camera of a humanoid robot. We assume that a complete 3D model of the environment is already given, for example generated during a previous SLAM run or provided by the user. The robot then has to determine a sequence of lookout poses for the camera so that all relevant regions can be covered, e.g., for the purpose of executing a search or inspection task. Thus, the goal is to find a preferably small set of viewing poses that respect the robot's kinematic limits and stability constraints and from where the whole scene can be observed. As discussed in Section 5.2, the problem of finding the minimum set of view poses that cover the whole environment is known to be a hard problem on its own according to complexity theory. Solving this problem in the context of humanoid robots introduces several constraints and long planning times due to the many degrees of freedom, which increase the complexity of the problem even further.

We approach this challenge by implementing a sampling-based next-best-view algorithm that has already been successfully used on tracked vehicles [41]. For efficient planning for humanoid robots, we extend this approach by pre-computing possible robot poses in an inverse reachability map that can be queried efficiently while searching for good view poses. In the following sections, we will introduce our efficient implementation of the inverse reachability map and present its applicability within a next-best-view planning algorithm.

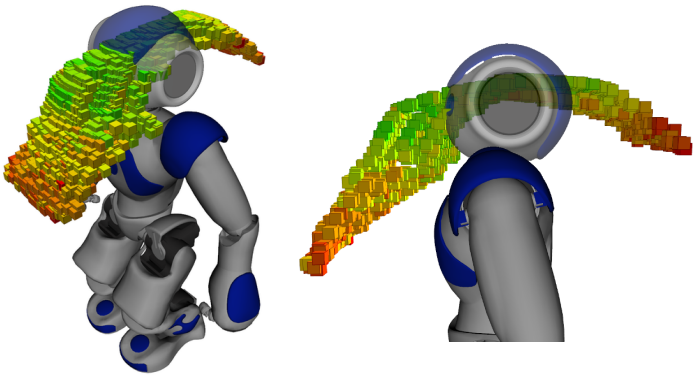


Figure 5.2: Reachability map of a Nao robot. The colored boxes represent poses that the top camera mounted in the robot's head can reach given the robot's current feet positions. Green poses have low costs according to the cost function (Equation 5.4), whereas red poses with high costs should be avoided.

## 5.4 REACHABILITY MAP AND POSE EVALUATION

Whole-body planning for humanoid robots is a challenging problem due to the high-dimensional configuration space and due to computationally expensive constraints such as posture stability and self-collision avoidance. Planning times can be significantly reduced by pre-computing valid postures and storing them as a *Reachability Map* (RM). A reachability map is a volumetric representation of the poses that an endeffector can reach given that the robot's base frame (i.e., the center pose between the feet poses on the ground) is located at the origin of the RM. The RM is typically computed by sampling in the configuration space. Each cell of the RM that is marked as reachable can be annotated with one or more joint configurations for reaching the desired pose together with a cost value associated with that joint configuration. During motion planning, the planner uses the RM as a lookup table for finding a set of suitable robot configurations without having to perform expensive kinematic computations or stability and self-collision checks. The planner then only has to perform location-dependent checks such as collision checks with the environment and optimize a cost criterion.

Reachability maps have already been successfully used for grasp planning [174] and stance pose planning [28] with humanoid robots where a 6D grasp pose is given in world coordinates and the robot has to find suitable, collision-free stance poses for reaching the desired grasp pose. Transferring this concept to our application of full coverage planning, however, needs two modifications as the requirements are different.

First, the size of the reachable volume is different. In the grasp planning application, the reachable volume of a robotic arm with multiple degrees of freedom is large as it covers the robot's whole

workspace and the endeffector can be translated and rotated in all six dimensions. In our application, by contract, the endeffector is a head-mounted camera, which typically can only move within a small and thin volume, shaped like a spherical segment (see Figure 5.2 for an example). Nao robots do not have a roll joint in the neck and the two hip joints are mechanically connected, hence the robot can only rotate the camera around the roll axis within a very limited range of a few degrees by tilting the whole upper body when shifting the knee positions. As the reachability map is sparse in our case, it would be inefficient to represent it as a full 6D grid map and to sample pose candidates from a full grid structure. Hence, we will instead propose an indexed database in the next section.

Second, the optimization criterion of grasp planning application is not useful in our application. The optimization criterion in the grasp planning is based on a manipulability measure. Using a tool with the robot's hand requires the ability to move the endeffector to nearby poses, e.g., for turning a screw with a screwdriver. Hence, Yoshikawa [188] proposed the manipulability measure

$$w = \sqrt{\det(JJ^T)}, \tag{5.1}$$

where $J$ is the Jacobian of the joint configuration with respect to endeffector pose. $w$ indicates how easily the robot can move the endeffector from the current pose in any of the six dimensions by moving its joints. If the endeffector position is a singularity from where it cannot be moved in a certain direction, then $JJ^T$ does not have full rank and $w$ drops to zero. Vahrenkamp et al. [173] extended this measure by adding penalties for configurations near joint limits and endeffector positions nearby obstacles.

In our application, however, manipulability of the endeffector is not important. The robot only has to reach a given camera pose for observing the environment, but it does not necessarily have to move the camera around in that position. Hence, it is acceptable if the kinematic chain reaches a singularity. Penalties for endeffector poses near obstacles would be redundant in our application, as the next-best-view algorithm selecting the camera poses already prefers panoramic viewpoints that are far away from obstacles so that large parts of the environment are visible in the robot's field of view.

Hence, we propose a new cost function tailored to coverage planning that considers the constraints that are relevant for humanoid robots. The three most important constraints are pose stability, time needed to reach the pose, and energy consumption.

Pose stability is a critical issue because the robot has to go to the limits of what is physically possible to reach all desired camera poses. To peek down into a box on the ground, for example, the robot has to bend over with its torso and tilt its head down, which moves both the center of gravity and the zero moment point to the boundary

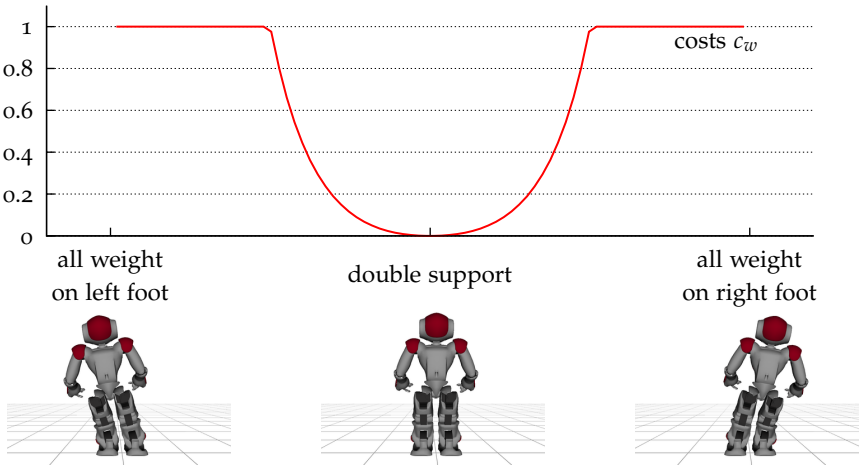



Figure 5.3: Cost function for measuring pose stability based on the weight distribution between the feet. Postures in double support are preferred as they lead to bigger support polygons and are considered to be more stable than single support postures.

of the support polygon. While the robot should be able to move to less stable poses when necessary, it should generally prefer stable poses. Hence, the first component our optimization criterion is a pose stability measure. One possibility for defining such a measure is to calculate the zero moment point and to determine the location of the point with respect to the support polygon, or alternatively to measure the center of pressure on both feet directly with sensors. When the center of pressure approaches the boundary of the support polygon, the robot posture gets unstable and small perturbations put the robot at risk of tipping over. For the Nao robots that we use during our experiments, however, the center of pressure cannot be measured reliably enough, hence, we use an approximate stability measure that compares the weight distribution on the two feet. Postures where the robot is in double support are more stable than poses where all the weight rests on one foot only. Let $w_l$, $w_r$ be the weight on the left and right foot, respectively, as measured by the foot pressure sensors located in the soles of the feet. Then, we define a weight ratio

$$r = \begin{cases} \min\left(\frac{\max(w_l,w_r)}{\min(w_l,w_r)}, m\right) & \text{if } \min(w_l, w_r) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

that is the ratio of the weight distribution between the two feet clamped to the range $[1, m]$ where $m$ is a user-defined constant, meaning that a pose is considered unstable if the weight on one foot is more than $m$ times higher than the weight on the other foot. The ratio is symmetric with respect to both feet. In practice, a value of $m = 3$ yields reasonable results. We then define a cost function

$$c_w = \frac{(r-1)^2}{(m-1)^2} \tag{5.3}$$

that increases quadratically with increasing weight ratio and is normalized to the range [0,1]. See Figure 5.3 for an illustration of the cost function.

A low run time for reaching a camera pose is important for efficiency of the whole coverage process. To estimate the time $\Delta t$ to reach the desired robot posture, we measure the time to get from a standard walking pose that the robot uses for navigation to the desired pose. Alternatively, the run time can be estimated from the joint displacements and the maximum joint velocities.

As the motors of humanoid robots easily overheat when remaining in a stressful pose for a longer period of time, we minimize the energy consumption of the motors in our optimization criterion. We continuously measure the electric current for each joint using the built-in shunt resistors while moving from the standard walking pose to the desired pose and back to the walking pose. Given the technical properties of the motors given in the data sheet of the robot, we compute the consumed power $P$ integrated over the whole movement.

We then combine the three cost components to a cost function

$$c \quad = \quad k_w \cdot c_w + k_t \cdot \Delta t + k_p \cdot P \tag{5.4}$$

with the components defined above and linear coefficients $k_w, k_t, k_p$ that we determined experimentally. Our view-point planning algorithm tries to find poses that cover the environment while minimizing this cost function.

For generating the reachability map, we sample a large number $n$ of robot postures $q$ in the configuration space, execute the configuration on a real robot, measure the time, power, and stability data, and compute the cost function. The result is a set

$$R = \{(q, c(q)) \mid i = 1 \dots n\} \tag{5.5}$$

of postures with associated costs that can be visualized as a reachability map (see Figure 5.2). The individual cells of this reachability map reflect the reachable workspace of the robot and the color of the boxes in the figure shows the associated cost. If more than one configuration leads to the same pose, the costs of the lowest cost pose is shown.

## 5.5 EFFICIENT REPRESENTATION OF THE IRM

In both the grasping application and our view-point planning application, the desired endeffector pose is implied by the task, whereas the robot's base position can be chosen freely. Hence, it is more efficient to invert the reachability map to create an *Inverse Reachability Map* (IRM) where the endeffector is located in the origin and the cells reflect the potential locations of the base frame from where the robot can reach the desired endeffector pose. The IRM can be represented as a 6D voxel structure where the voxel coordinates correspond to the 6D pose of

the robot's base frame and each voxel contains a list of one or more joint configurations. Given a desired endeffector pose, the algorithm transforms the IRM into the world coordinate system so that the origin of the IRM matches the desired endeffector pose. The intersection of the transformed IRM with the ground plane yields a list of potential stance foot positions, which then have to be checked for collisions and optimized for the cost value.

Intersecting the two volumetric representations of the IRM and the environment model as Burget and Bennewitz suggest [28], however, is time-consuming and storing the IRM as a sparse 6D structure is not memory efficient. Making the assumption that the robot can only stand on horizontal planes allows for a more efficient implementation. If the robot's feet rest flat on the ground, then the roll and pitch angle of the feet relative to the endeffector frame as well as the distance between the endeffector and the feet on the vertical axis can be directly derived from the endeffector coordinates in the world frame. Hence, we propose to represent the IRM as a database instead of a volumetric representation. As the schematic layout of the database in Figure 5.4 shows, each entry of the database consists of a list of base frame poses relative to the endeffector and a whole-body joint configuration to reach the endeffector pose from the given base pose with a corresponding cost value. We index the database on the roll ($\phi$), pitch ($\theta$), and $z$ component of the base frame pose discretized into equally spaced bins $(\overline{\phi}, \overline{\theta}, \overline{z})$. As these coordinates can be computed directly from the desired endeffector pose under the assumption that the feet rest flat on the ground, we can access the candidate base frame poses without having to perform geometrical intersections of volumes by finding the nearest neighbor in the database. In practice, the IRM can be implemented as a $k$-d tree or octree that allows quick nearest-neighbor searches. For a given query, the database returns a list of candidate feet poses to reach the desired camera view pose. The algorithm then selects the configuration with the lowest costs that does not collide with the environment.

| Index | List of entries |
|---|---|
| $(\overline{\phi}, \overline{\theta}, \overline{z})$ | configuration 1: $(x, y, z, \phi, \theta, \psi)$, $(q_1, \ldots, q_N)$, $c$ |
| | $\vdots$ |
| | configuration $k$: $\underbrace{(x, y, z, \phi, \theta, \psi)}_{\substack{\text{base position} \\ \text{wrt. endeffector}}}$, $\underbrace{(q_1, \ldots, q_N)}_{\substack{\text{whole-body joint} \\ \text{configuration}}}$, $\underbrace{c}_{\substack{\text{costs} \\ \text{(Eq. 5.4)}}}$ |

Figure 5.4: Schematic layout of the database for storing the IRM. Each bin indexed by $(\overline{\phi}, \overline{\theta}, \overline{z})$ contains a list of $k$ whole-body configurations with associated costs.

## 5.6 PLANNING A TOUR OF VIEWING POSES

For generating view points and planning a sequence of poses to cover the environment, we adopt ideas of the approach by Dornhege et al. [41]. Please refer to the original publication for a more detailed description including mathematical formulations of the problem and algorithm.

### 5.6.1 *Sampling candidate viewing poses*

The known model of the environment is represented efficiently as an OctoMap [75]. Our algorithm first determines which occupied voxels belong to surfaces that should be covered. For our application, the ground plane does not have to be observed, so we filter it out and limit the region of interest to a user-defined bounding volume. However, our system still keeps the full OctoMap for collision checks and navigation planning.

For each occupied voxel to be observed, the algorithm casts rays starting from the occupied voxel into free space. The ray is clipped at a minimum and maximum distance from the occupied start voxel corresponding to the distance range where the robot can well observe the surface. For each free-space voxel, a counter is created that counts the number of rays traversing the voxel. If many rays pass through one voxel, this voxel is assumed to be a good lookout point as many interesting surfaces can be observed. Contrary to Dornhege's original approach, we do not sample random linear rays, but subdivide the unit sphere around the occupied voxel into 512 equally shaped conic rays. We then iterate through the cells in each cone from the tip outwards and increment the voxel counters of the traversed cells. When a collision occurs, we continue with the next cone. This approach is more systematic and a better representation of the utility of the view pose candidates, as it eliminates the systematic bias of cells near walls that get traversed more often if randomly sampled linear rays are used. We filter the traversed voxels by the height above the ground and do not consider voxels that are above or below the range where the robot's camera can be placed.

We then sort the remaining free-space voxels by decreasing utility according to the ray count. For each voxel at position $(x, y, z)$, we sample $n$ random 3D orientations $(\phi_i, \theta_i, \psi_i)$ to get a set of 6D camera poses $\{(x, y, z, \phi_i, \theta_i, \psi_i) \mid i = 1, \ldots, n\}$. The yaw angles $\psi_i$ are sampled from the full range $[0, 2\pi]$. As we represent the inverse reachability map as a database indexed by bins $(\overline{\phi}, \overline{\theta}, \overline{z})$ (see Section 5.5), we can directly sample roll angles $\phi_i$ and pitch angles $\theta_i$ for a given $z$ from the inverse reachability map, guaranteeing that all sampled poses are within the feasible kinematic range of the robot. For each of the $n$ camera poses, we determine the number of surface voxels that are

visible in the viewing frustum, which yields a utility value for that camera pose.

### 5.6.2 *Determining whole-body configurations*

If the utility of a camera pose is above a threshold, our system determines whether there is a collision-free robot pose for reaching that view. The methods used in Dornhege's original approach are not efficient and capable enough for humanoid robots with very limited reachability ranges and constraints on stability and energy consumption. To cope with these challenges, our algorithm queries the inverse reachability map (see Section 5.5) to retrieve a list of candidate whole-body configurations for reaching the given camera pose. Each configuration consists of a list of joint angles, the poses of the robot's feet relative the camera frame, and a cost term. For each of the configurations, we transform the feet poses into the world coordinate system and perform a sequence of checks to determine whether the whole-body pose is reachable:

1. Check that the desired robot location is reachable from the robot's start location. As computing a full plan from the start location to the desired location is too computationally expensive to be executed for a large number of candidate views, we instead pre-compute a 2D reachability map once at the beginning and update the reachability map in case the environment changes. In our current implementation, we generate a 2D occupancy grid map by down-projecting the 3D model onto the map. In the resulting map, we use a region-growing algorithm to mark all free-space cells that are definitely reachable from the robot's starting position.

2. Check that the feet poses of the configuration (including stepping safety margins) do not collide with obstacles by comparing the feet polygons to the 2D grid map of the environment.

3. Check that the full body of the robot does not collide with the environment. In our experiments, we use the Flexible Collision Library [134] for fast collision checks.

The checks are ordered by ascending complexity for lazy evaluation, as the more complex checks do not have to be performed if a simpler check already failed. If a given camera pose can be reached by multiple robot configurations, which is usually the case, we select the configuration with the lowest costs according to the cost function (see Equation 5.4). If no whole-body configuration is found, then the camera pose is unreachable and is not considered further.

### 5.6.3 *Formulation as a traveling salesman problem*

The user can trade off the run time of the view pose search versus the thoroughness of the coverage by setting the number of view pose samples and the utility thresholds, i.e., the ray count for the voxels and the number of visible surface voxels for the sampled camera poses. After sampling and evaluating camera poses for all high-utility voxels, we get a set of camera poses that cover large parts of the environment. Following Dornhege's approach [41], we partition the observed voxels by the viewing poses to determine the smallest set of viewing poses that still covers all observed voxels. This step reduces the number of poses that the robot has to navigate to and thus reduces the size of the planning problem to make it tractable.

The final step before the robot can start executing its task is to compute a tour for visiting all viewing poses, starting at the robot's current location. Dornhege [41] provides a comparison of different planning algorithms including utility-based and cost-based greedy algorithms, set cover and traveling salesman planners, and an exhaustive search. The best choice of the planning algorithm depends on the application: For object search tasks, it makes sense to start with panoramic view points where large parts of the environment are visible, as it is likely that the object can be seen from these view points and the search can be completed early. Hence, a utility-based greedy approach should be used in this scenario. If, by contrast, the task requires that all view points must be visited, for example in an inspection or mapping task, then a cost-minimizing planner is preferred. For our experiments, we choose to formulate the problem as a traveling salesman problem on a graph and use a Lin-Kernighan heuristic solver [69] to find the shortest-path solution that visits all viewing points necessary for completely covering the environment. As Dornhege showed, the additional time required for solving a TSP in comparison to greedy approaches is outweighed by the gain in the time and energy required to execute the plan, which is especially true for humanoid robots.

To plan the coverage tour, we follow the same approach as for exploration with background knowledge in Section 3.3.2. The nodes of the TSP graph consist of the viewing poses and the robot's start location. For each pair of nodes, we add an edge annotated by the length of the shortest path between the corresponding poses as computed by an A* planner on the 2D occupancy grid map. As we don't require the robot to return to the start location, we make the graph asymmetric by replacing the costs for traveling from any view point back to the start node by 0, meaning that the robot can "teleport" at no costs from the last visited view pose back to the start. In the resulting tour, the last edge is removed, leading to the shortest open-end path starting at the robot's current location and visiting all viewing poses. For

a mathematical formulation of this process and a discussion of the consequences for the problem complexity see Section 3.3.2.

## 5.7 EXPERIMENTS

To evaluate our approach, we conducted a series of experiments in both simulated environments and real-world settings with a Nao humanoid robot (see Appendix A.1 for technical specifications of the robot).

### 5.7.1 *Generating the inverse reachability map*

As a first step, we need to record an IRM once in the beginning for the given robot. Generating the IRM in simulation is not accurate enough, especially as the simulators that support Nao robots cannot simulate critical factors for stability such as joint backlash and inertia effects. Power consumption is also hard to estimate in closed-loop reactive systems. Hence, we propose to record the IRM in real world in an automated process where the robot moves to sampled poses while measuring the required quantities for computing the cost function.

Any posture generator can be used to create the set of samples to be stored in the IRM, including kinesthetic teaching by a human or random sampling in a physics simulator. In our case, we used the whole-body controller provided by the manufacturer as a black-box posture generator for recording the IRM. The whole-body controller formulates the generalized inverse kinematics problem as a quadratic problem including joint limit constraints and stability criteria that constrain the center of mass to the support polygon. The controller then solves the quadratic problem in a fixed cycle of 20 ms. More details on the whole-body controller are given in the manufacturer's documentation [4]. We systematically sampled head orientations in the feasible range and torso heights between 24 cm and 32 cm and let the whole-body controller find suitable joint configurations. The configurations are evaluated and stored in the reachability map as described in Section 5.4. In our experiments, the IRM contained 1514 robot configurations in 277 database records. For more complex robots with more degrees of freedom, the pose sampling density can be reduced to keep the algorithm efficient while still covering the full configuration workspace.

### 5.7.2 *Coverage of simulated environments*

We tested our approach first in simulation experiments with a Nao humanoid simulated using the Choregraphe framework provided by Aldebaran Robotics. To generate footstep plans, we used the anytime search-based footstep planner by Hornung et al. [74]. To study our

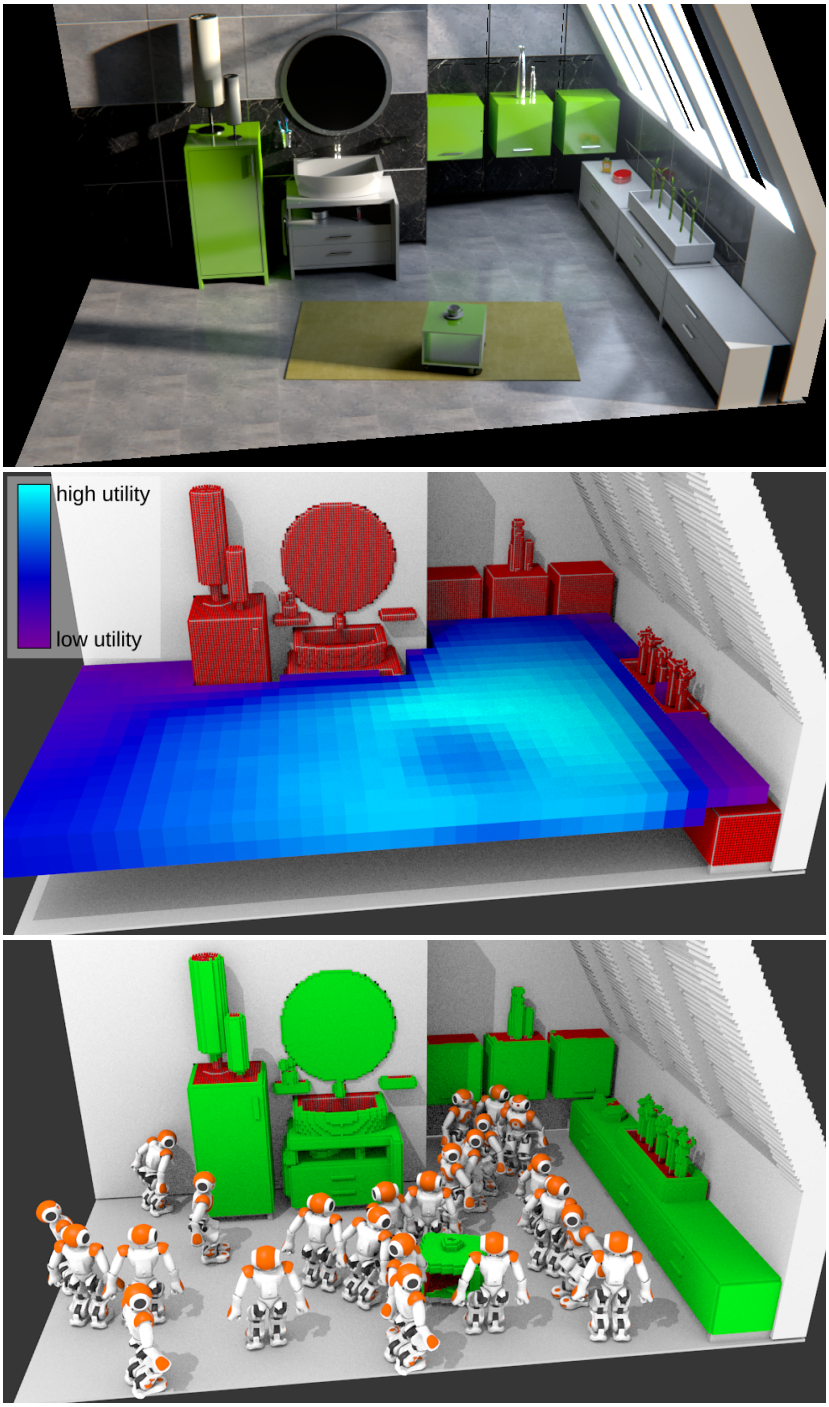



Figure 5.5: Bathroom scene, model based on [22]. *Top*: Overview of the scene. *Middle*: Utility of candidate viewing poses (see Section 5.6.1). Light blue voxels indicate panoramic view points from where large parts of the scene are visible. The robot's task is to observe the red parts of the environment, whereas the gray objects are only considered for collision checking and path planning, but do not have to be observed. *Bottom*: Resulting set of poses and areas covered by the robot's camera (green). The maximum viewing range of the camera is 3 m.

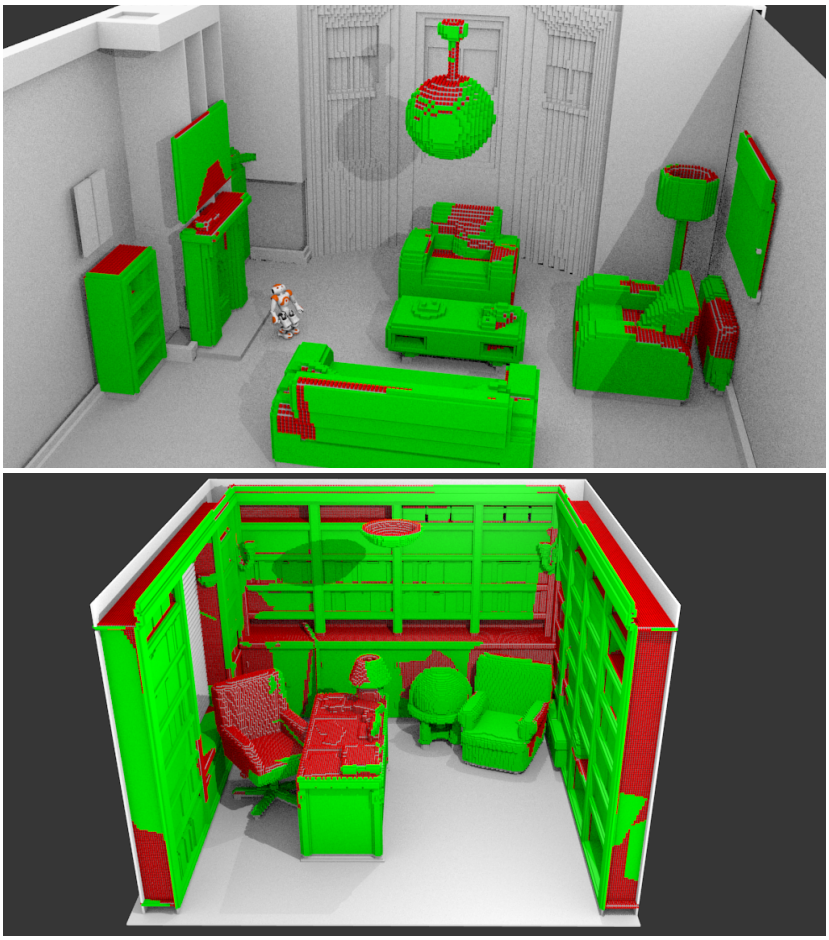


Figure 5.6: Coverage results of further simulation experiments. *Top*: "The White Room" model based on [68]. Note that the robot inspects the ceiling lamp by leaning backwards and looking up. *Bottom*: "Library-Home Office" model based on [23].

algorithm's behavior in realistic settings, we downloaded openly available models of indoor rooms and apartments provided by the Blender community and rendered the models as OctoMaps [75]. Figure 5.5 shows an example scene in a bathroom based on a model from [22]. The red surfaces are the relevant environment regions that the user selected for the robot to inspect. The figure in the middle visualizes the utility map generated with the algorithm described in Section 5.6.1. Light blue voxels are traversed by many conic rays emitted from the user-selected surfaces, thus these voxels are panoramic view points from where large parts of the environment are visible. The algorithm considers these voxels first when sampling view poses. The bottom figure of Figure 5.5 shows the resulting robot poses and the areas covered by the robot's camera in green. Some surfaces are unobservable for the robot due to its body height, e.g., the top face of shelves.

Figure 5.6 shows additional experiments in other environments. The top figure visualizes a living room scene and highlights that the robot

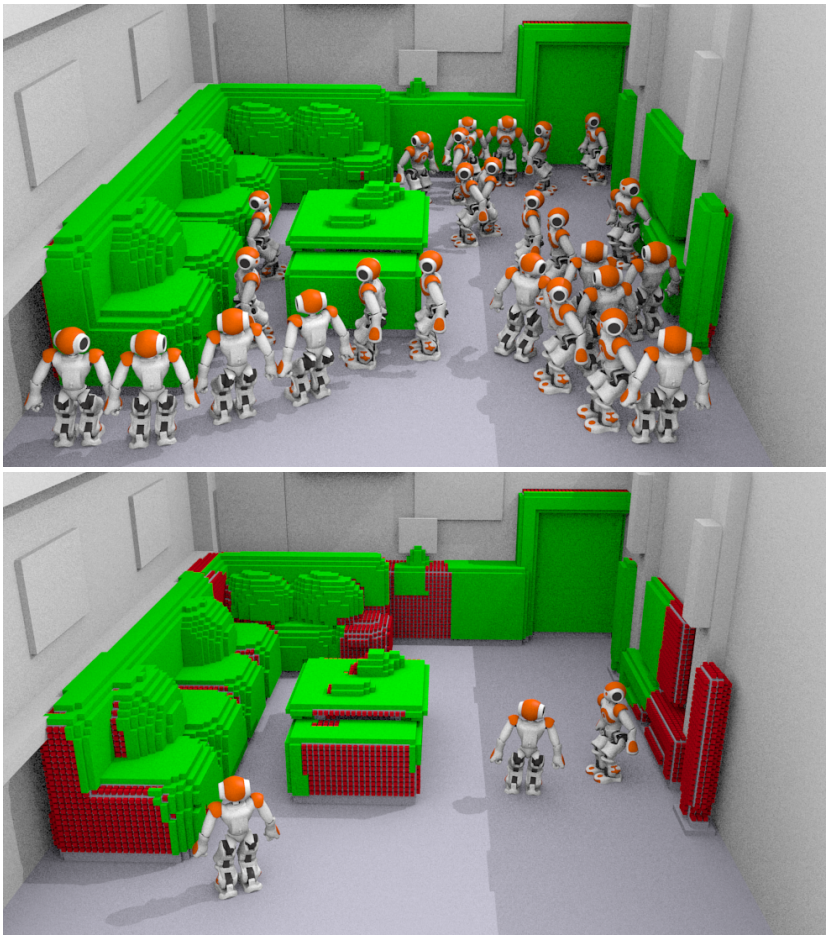


Figure 5.7: Trading off between completeness of coverage and time required to execute the plan. Both images show the "Living Room" model based on [120]. By adapting the utility threshold, the user can trade off between completeness of coverage versus the number of poses and thus the time for task completion. *Top*: The environment is fully covered by 26 robot poses. *Bottom*: The three robot poses with the highest utility already cover large parts of the environment.

also has to lean back and look up to inspect the ceiling lamp. The bottom figure shows a library scene. The remaining areas in red are either inaccessible to the robot, for example the table top, or are left out by the robot as the information gain is too small compared to the costs of navigating to a suitable view point.

Figure 5.7 shows the same living room scene covered with different numbers of view poses. By choosing the threshold for the candidate view utility appropriately, the user can trade off between completeness of coverage versus run time. A low utility threshold (Figure 5.7 top image) leads to 26 view poses that cover all observable details of the scene. A higher threshold on the utility, in contrast, leads to a small set of high-utility poses that already cover large parts of the environment. The bottom image of Figure 5.7 shows the coverage of the three highest utility poses that already cover most of the environment. In object search tasks, a planner should be used that executes these highest utility poses first, thus the robot searches for the object from panoramic view points first and proceeds with lower utility, close-up view points of smaller details until the object has been found.

### 5.7.3 *Coverage of a real scene*

We conducted experiments with a Nao humanoid in a real-world environment with children's toys and furniture that matches the size of the robot. The 3D map was recorded with a SLAM approach running on RGB-D data from an ASUS Xtion Pro Live camera (technical specifications in Appendix A.2). Figure 5.8 shows an example scenario of such an environment where the robot successfully covers all objects. The results show that our approach can handle environment models with measurement noise.

## 5.8 DISCUSSION

In this work, we used a sampling-based raycasting approach to find panoramic viewpoints from where large portions of the environment are visible. The advantage of this approach is that it can be applied in unstructured environments without relying on planar surfaces, polygonal representations, or similar assumptions. For this reason, this approach was chosen in the original application of finding persons in disaster scenarios in [41]. This flexibility, however, comes at the expense of longer planning times as raycasting is generally a time-consuming operation. Adapting the number of rays and the resolution of the map representation allows trading off between accuracy and efficiency. The optimal parameters depend on the task, the robot, and the environment. In search tasks, for example, the map resolution should be chosen in relation to the object to be searched, as searching for smaller objects needs finer resolutions than searching for bigger

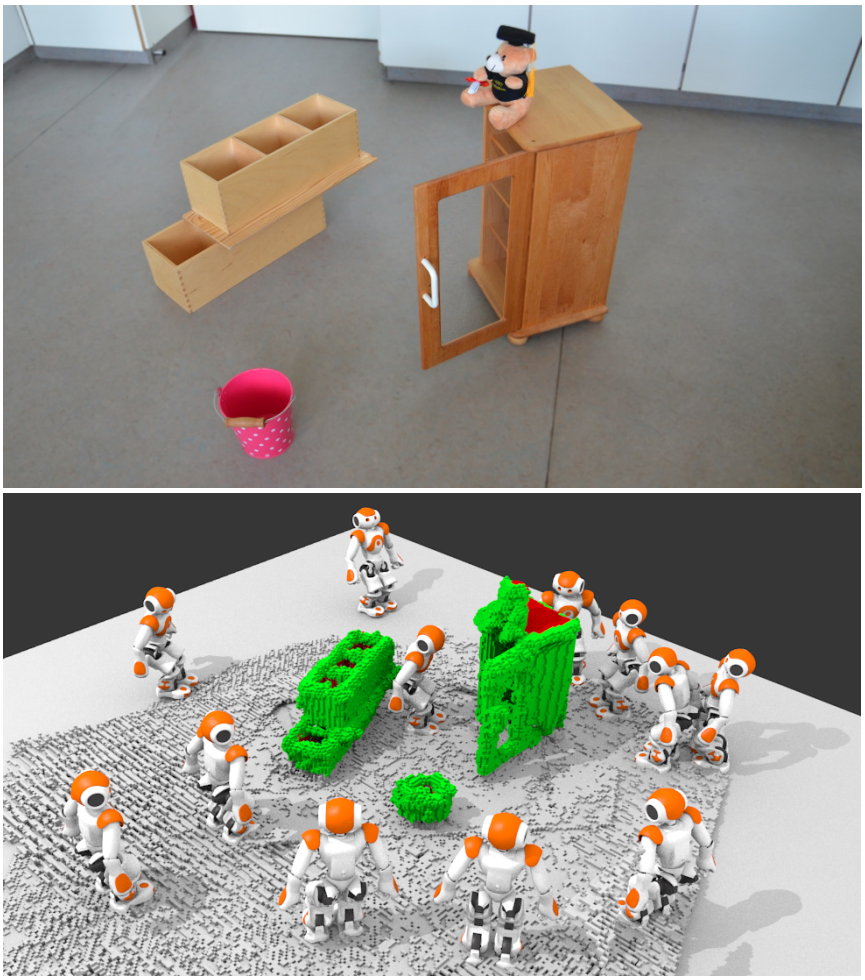


Figure 5.8: Real-world experiment. *Top*: Overview of the scene. *Bottom*: Resulting set of poses and covered surfaces (green). The computed strategy contains both panoramic view points where large parts of the scene are visible and close-up view poses for peeking into the cupboard. The maximum viewing range of the camera is 1.5 m.

objects. For the experiments shown in this chapter, the run times for planning the coverage tours are in the range of 30 s to a few minutes as we have chosen relatively fine resolutions. As the run times strongly dependency on the parameters and the environment, however, the exact run times for the experiments are only of limited relevance. In the following chapter, we will present a different approach for the coverage problem that uses a rendering pipeline instead of raycasting. The new approach will be orders of magnitude faster, but requires environment models that can be efficiently represented as triangulated meshes, which is generally more likely the case in structured environments such as indoor home and office environments.

The utility measure in this work considers the information gain from observing the environment as a positive factor and the costs for energy consumption, execution time, and potential pose instability as negative factors. Depending on the task of the robot, this basic utility function can be extended. If, for example, the task is to search an object in a home environment, the likelihood of finding the object in a particular location should be included in the utility function, as finding a coffee cup in the kitchen is more likely to finding it in the bathroom. Several approaches for estimating such likelihoods have been presented in the literature, for example semantic mapping approaches [140], object co-occurrence estimation, or heuristics for finding objects in structured environments that leverage knowledge about typical object arrangements [82].

We implemented tools for recording and visualizing inverse reachability maps and implemented the whole-body planner from Section 5.6.2 as a kinematics plugin compatible with the *MoveIt!* motion planning framework. Hence, our planning module can be re-used in other projects for computing whole-body poses for given camera poses in real time.

## 5.9 CONCLUSIONS

In this chapter, we presented a framework for planning view points and full-body postures for covering a known environment with the camera of a humanoid robot. We introduced a novel representation for inverse reachability maps that supports fast sampling and validation of robot poses. Integrating our inverse reachability map representation with a sampling-based next-best-view algorithm allows us to interleave view point planning with full-body pose planning for a humanoid robot. Our algorithm produces a set of full-body postures that are feasible and energy efficient and allow the robot to cover the whole observable 3D environment with its camera. In combination with a traveling salesman problem solver, our algorithm generates an efficient plan that can be used in a wide range of applications including inspection, surveillance, mapping, and search tasks.

# 6

# GPU-ACCELERATED NEXT-BEST-VIEW EXPLORATION OF ARTICULATED SCENES

*In this chapter, we extend the coverage problem discussed in the previous chapter to environments that also contain articulated environments where the robot also has to manipulate objects to inspect obstructed areas. This problem is particularly challenging due to the additional degrees of freedom resulting from the articulation. We propose to exploit graphics processing units that are present in many embedded devices to parallelize the computations of a greedy next-best-view approach. We implemented algorithms for cost map computation, path planning, as well as simulation and evaluation of viewpoint candidates in OpenGL for Embedded Systems and benchmarked the implementations on multiple device classes ranging from smartphones to multi-GPU servers. We introduce a heuristic for estimating a utility map from images rendered with strategically placed spherical cameras and show in simulation experiments that robots can successfully explore complex articulated scenes with our system.*

## 6.1 INTRODUCTION

In the previous chapter, we presented an approach for planning a tour for covering a known environment with a robot's camera, for example for the purpose of searching for an object. Real-world environments, however, are often more complex. If we would like a service robot to search for an object in home and office environments, the robot will likely have to manipulate objects to inspect their contents, for example to open cupboards, drawers, and dishwashers, or to move furniture aside to observe occluded areas. In this case, the robot has to plan a whole action sequence of manipulating objects and moving to suitable camera poses to investigate both the environment and the inside of containers. As discussed in the previous chapter, the coverage problem is hard even for static scenes, as well-known NP-hard problems such as the art gallery problem and the set coverage problem can be reduced to coverage of known scenes. Considering articulated objects adds even more degrees of freedom, making it infeasible to solve the planning problem by exhaustive search. While several approaches for covering known scenes exist in the literature, including sampling-based and probabilistic techniques, these methods mostly consider only static scenes due to the high complexity of the planning problem. In this chapter, we propose to make the complex problem of planning a

Figure 6.1: Nao inspecting a kitchen environment. The robot's task is to search for an object in the user-defined regions of interest marked in red. Kitchen model based on [60].

coverage tour in articulated scenes more tractable by applying two approaches: First, we identify subproblems that are parallelizable and propose algorithms for solving these problems efficiently on the computer's Graphics Processing Unit (GPU), and second, we propose a novel heuristic for estimating utility maps to speed up the process of finding suitable camera poses.

As in the previous chapter, we again assume that the robot already has access to a 3D model of the environment, as well as user-defined regions of interest that the robot shall cover. Additionally, we now also assume that the robot has knowledge about how to manipulate the articulated objects in the scene, for example how to open the door of a cupboard. We propose a greedy next-best-view approach that selects the next viewpoint by maximizing the expected utility, which trades off the costs for the robot navigating to the viewpoint and the expected information gain from observing the environment. Estimating this utility function contains several subproblems. We define the information gain as the size of the newly observed portion of the region of interest in the camera image, hence our algorithm needs to render a virtual camera view from each potential view pose. In the navigation cost function, we need to consider that the robot has to keep a safety clearance to obstacles, so we follow the popular approach of computing an inflation cost map. Determining the costs for navigating to viewpoint candidates includes solving a single-source shortest path problem. All these problems are highly parallelizable. Hence, we propose to exploit the computer's GPU to parallelize the workload for solving the subproblems, which allows us to compute utility maps for a large number of articulation configurations in a reasonable amount of time, making the planning problem more tractable.

Fast GPUs are widespread and due to the popularity of deep learning algorithms their power and availability will likely increase even more in the future. In the literature, algorithms for some of the mentioned subproblems have already been successfully formulated and solved using General-Purpose Computation on Graphics Processing Units (GPGPU) on devices that support high-level frameworks such as CUDA, OpenCL, or ROCm. Variations of Dijkstra's shortest path algorithm, for example, have been implemented on CUDA [108]. GPGPU approaches, however, need high-end graphics cards. Embedded systems such as robots, smartphones, and single-board computers often do have a GPU, but only provide a subset of functions needed for their original purpose of graphics rendering. To leverage the computing power of embedded systems GPUs, we formulate all algorithms as rendering problems and implement them in OpenGL for Embedded Systems (OpenGL ES), which is an open standard and widely supported across platforms.

The three main contributions of the approach presented in this chapter are: First, we adapt the jump flood algorithm for computing cost inflation maps, the Bellman-Ford algorithm for shortest path planning, and a view simulation algorithm for estimating the information gain for being solved with an OpenGL ES graphics pipeline. We show optimizations for increasing the throughput of the algorithms and make use of modern OpenGL features such as transform buffer feedback and random image access. We benchmark the algorithms on multiple device classes ranging from smartphones to multi-GPU servers.

Second, we introduce a heuristic for estimating a utility map based on rendering the scene with virtual spherical panorama cameras placed strategically at the edge of articulated objects and compare the result with a ground-truth utility map obtained by exhaustive sampling.

Third, we integrate the algorithms and the heuristic into a system for exploring an articulated scene and show in simulation experiments how a humanoid robot successfully explores home environments such as the kitchen environment in Figure 6.1.

## 6.2 RELATED WORK

In Section 5.2, we already discussed problems related to coverage of known scenes such as the art gallery problem and the traveling salesman problem and we referenced other next-best-view approaches for exploration and coverage planning. These algorithms, however, are designed for Central Processing Units (CPUs). In this chapter, we propose to leverage the GPU to accelerate the search for the next-best-view candidates.

Graphics processing units have originally been developed to speed up typical rendering tasks such as texture mapping and interpolation,

polygon rendering, coordinate system transformations, and raytracing. These tasks are characterized by two properties: First, they are memory intensive, hence modern GPUs include specialized caches to reduce the time needed to fetch data. Second, these tasks belong to a class of problems coined "embarrassingly parallel problems" [71], i.e., problems that can be split into concurrent tasks with little or no effort, promising high potentials for speed up through parallelization according to Amdahl's law [8]. To exploit this data parallelism, GPUs use Single Instruction Multiple Data (SIMD) processor architectures that execute the same sequence of operations on multiple pieces of data in parallel.

As these characteristics also apply to many problems in robotics, parallelization techniques have been investigated before. Lee and Lin [97] presented a design for a SIMD machine in 1991 and showed how this architecture can be exploited to speed up the computation of kinematics, dynamics, and Jacobians including their inverses. More recently, Taylor and Kleeman [168] reported a 4– 8 times reduction in execution time when exploiting the multimedia and streaming extensions of modern regular CPUs for processing images in a hand-eye coordination scenario on a robot platform. Martínez et al. [109] provide a direct comparison of CPUs with SIMD extensions and GPUs in the context of emotion-based decision making in service robots. In contrast to these approaches that focus on the low-level structure of the problem and its parallelization, we formulate navigation tasks as high-level rendering problems and delegate the parallelization to the driver of the graphics card.

As the parallelization architecture of GPUs is also beneficial for many problems beyond computer graphics, GPUs have evolved in recent years to general-purpose computing boards that can be freely programmed in high-level languages. This concept is known as General-Purpose Computation on Graphics Processing Units (GPGPU). In the domain of navigation planning in robotics, several problems have been solved with GPGPU approaches, for example path planning with Dijkstra's algorithm [108], the Bellman-Ford algorithm with bucketing [39], parallel breadth-first search [111], or multi-agent path planning with potential field methods [47]. These algorithms, however, need GPGPU frameworks such as CUDA. In our work, by contrast, we limit GPU usage to the embedded systems subset of OpenGL that is more widespread, cross-platform, and does not require dedicated high-end computation boards. Camporesi and Kallmann [29] also use OpenGL GPU shaders to compute shortest paths based on shortest path trees. In contrast to their approach, our approach is not restricted to 2D polygonal input data, but can use any renderable scene.

OpenGL is a popular tool in robotics for the simulation and visualization of 3D scenes. It is used, for example, in the Gazebo [87] and Webots [112] simulators and in simulations of robot kinematics [179].

In our work, by contrast, we do not use OpenGL for visualization or simulation purposes, but for accelerating the computation of navigation subproblems by exploiting the parallel computation power of GPUs.

To the best of our knowledge, there are no existing algorithms for calculating exploration or coverage tours in environments with articulated objects that the robot has to actively manipulate.

## 6.3 PROBLEM FORMULATION

In this chapter, we consider the problem of planning a tour for covering user-defined regions of interest with a robot's sensor. We assume that the robot already has a renderable 3D model of the environment, either provided by the user or previously acquired using a SLAM approach and that the model contains a mechanism for moving the articulated objects. In our implementation, we represent the scene as a Collada model and the articulation mechanisms as bone animations. Each possible pose of an articulated object is mapped to a position variable $p \in [0, 1]$. For a sliding door, for example, $p = 0$ means that the door is closed and $p = 1$ means that the door is fully open. For movable furniture, $p$ indicates the position of the piece of furniture along a defined path.

Let $A$ be the set of articulation objects contained in the scene. Each articulation object $a \in A$ is represented as $a = (p_0, c)$, where $p_0 \in [0, 1]$ is the initial state of the articulated object and $c\,(p_{i-1}, p_i)$ is a cost function returning the costs for moving the articulated object from position $p_{i-1}$ to $p_i$.

We assume that the user defines the robot's task by specifying one or more regions of interest $R$ that the robot should cover. In our implementation, we represent the regions of interest as textured objects with a designated color, marked in red in the figures throughout this chapter. The algorithm then keeps track of observed portions of $R$ by marking them in a different designated color (green throughout this chapter).

We also assume that the scene model contains one or more surfaces $C$ where the robot's camera can be placed. For mobile robots with cameras mounted on the base these surfaces will typically be planes parallel to the ground, whereas for arm-like robots sphere surfaces can be used.

The goal of our approach is to find a sequence $s_1, s_2, \ldots$ of viewpoints and articulation positions $s_i = \left(v_i, p_i^1, \ldots, p_i^{|A|}\right)$ with low costs from where the robot can see as much as possible of the defined regions of interest. We define the costs as

$$cost(s_i) = \mathit{infl}(v_i) + \mathit{dist}(v_{i-1}, v_i) + \sum_{a=1}^{|A|} c(p_{i-1}^a, p_i^a), \tag{6.1}$$

where $infl(v_i)$ are costs exponentially decaying with the distance to the nearest obstacle following the commonly used inflation cost map approach, $dist(v_{i-1}, v_i)$ is the shortest path distance from the robot's current position $v_{i-1}$ to the new viewpoint $v_i$, with $v_0$ designating the robot's initial position, and $c$ are the costs for manipulating the articulated objects as defined above. In a single time step, multiple articulation objects can be moved, but typically only few articulation objects have to be changed to uncover a region of interest.

Finally, we define the information gain $IG(v)$ of visiting a camera viewpoint $v$ as the number of texels of the region of interest $R$ that are visible from $v$ and have not been observed before. The goal of our approach is then to maximize the utility function

$$U(s_i) = IG(v_i) - cost(s_i) \tag{6.2}$$

to iteratively determine the next best view.

## 6.4 GPU ALGORITHMS

We tailor our algorithms to GPUs of embedded systems, hence we implement our approach in OpenGL for Embedded Systems version 3.2, which is an open standard and widely supported on many platforms.

### 6.4.1 *Properties of the graphics pipeline*

OpenGL defines a pipeline of *shaders*. Shaders are simple programs that the GPU executes either for each vertex of the input mesh or for each pixel of an output image. Originally, these programs were used to compute light and shadow on rendered surfaces, hence the name *shader*. Nowadays, shaders are used for a wide variety of post-processing and special effects in computer graphics. As shaders are freely programmable, they can also be used for other purposes as long as the problem to be solved can be formulated to fit into the pipeline stages defined in the standard. The pipeline stages relevant for this work are as follows:

1. The *vertex shader* takes one vertex of the scene mesh at a time in world coordinates and transforms the coordinates to screen coordinates by multiplying with camera and projection matrices.

2. The *geometry shader* takes one primitive of the scene mesh at a time (i.e., a point, line, or triangle) and outputs an arbitrary number of primitives of the same type. This mechanism can be used to manipulate or duplicate the geometry of scene parts.

3. The *rasterizer* converts geometric primitives to a set of pixels and interpolates vertex attributes such as color and texture coordinates. The combined data for generating an output pixel is

called *fragment*. This stage can be influenced by parameters, but in contrast to the other stages it is not freely programmable.

4. The *fragment shader* takes the fragment data for one pixel at a time and computes the final output color of the pixel. The fragment shader can neither modify the pixel position, nor consider neighboring pixels.

All shaders can perform additional work, e.g., reading and writing to textures or passing variables to other pipeline stages. The performance gain that GPUs offer is mainly achieved by running the shader programs in parallel for multiple vertices or pixels using Single Instruction Multiple Data (SIMD) architectures. Considering the special architecture of GPUs, we followed these design guidelines:

- SIMD instructions by definition require that the same instruction is executed for all data. Branching into `if/else` constructs reduces the possibilities for parallelization, hence branching should be avoided where possible. While lazy evaluation paradigms speed up sequential computations, the overhead of computing unneeded data with SIMD might be smaller than branching based on whether or not the data is needed.
- The GPU processes the pipeline asynchronously. The graphics driver keeps the pipeline filled with about three next instructions for the GPU. Reading back data from the GPU to the CPU may introduce synchronization points that flush the pipeline. Following best practices [20, 110, 176], our GPU algorithms write to ring buffers that the CPU reads back with a delay of three frames, allowing the GPU to continue writing the next frame to a different buffer while data is transferred from an older buffer.
- In our implementation, we reduce data transfer between CPU and GPU to the minimum. All intermediate results are held on the GPU and only the final results are transferred to the CPU through a ring buffer.

### 6.4.2 *Cost map computation*

For safe navigation, robots need to keep a safety clearance from obstacles. A commonly used approach (e.g., implemented in the ROS navigation stack [102]) is to create an occupancy grid map indicating the obstacles and free space and to "inflate" the size of the obstacles by the safety margin. In a cost map, the costs for all cells inside the inflation radius around obstacles are set to infinity or a high constant. To get smoother paths around obstacles, the cost function is often modeled as exponentially decaying from inflation edges towards free space so that the robot can get close to objects if necessary, but gets an incentive to stay further away.

**Algorithm 6.1** Cost map computation of an $n \times n$ grid map

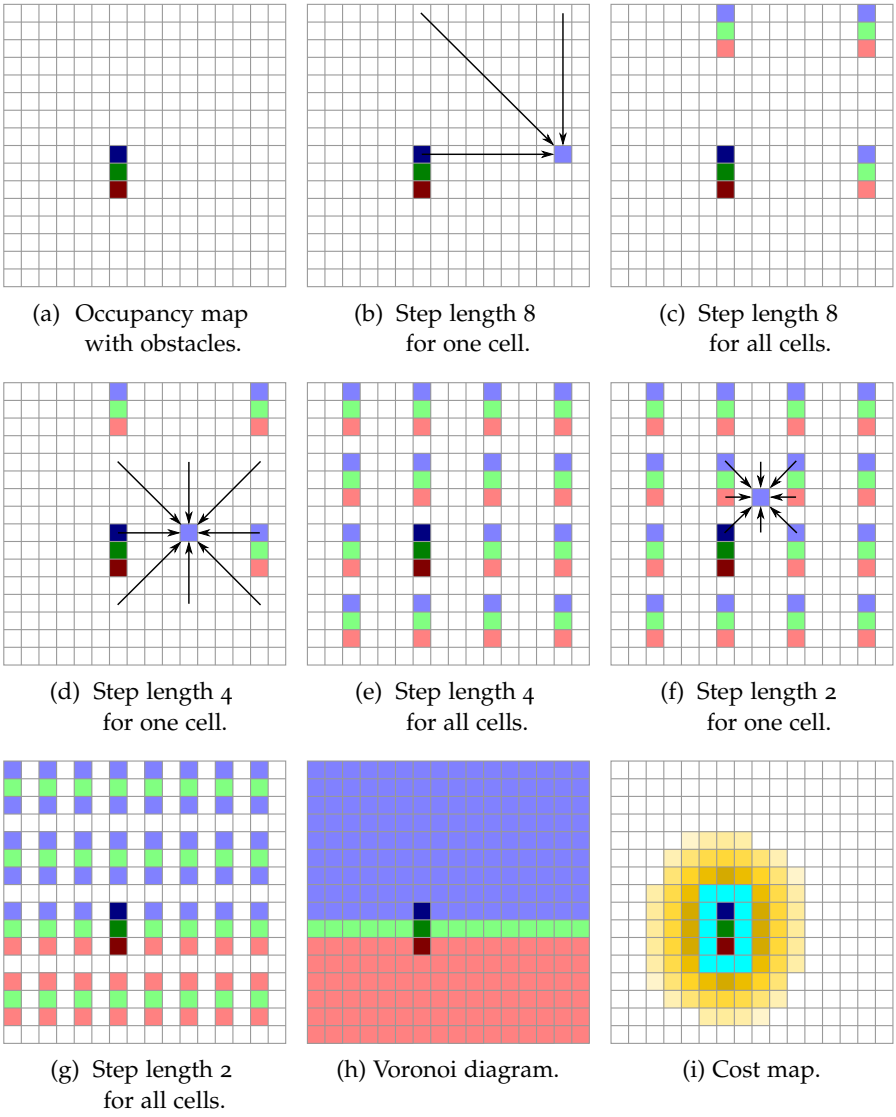
```
1:  // Initialization
2:  render obstacles
3:  for all obstacle fragments p do in parallel:
4:      color_p ← encode(coordinates of p)
5:  // Jump Flood Algorithm
6:  for all i ∈ {1, ..., log(n)} do:
7:      s ← 2^(log(n)−i)
8:      for all map pixels p do in parallel:
9:          for q ∈ {p + (x, y) | x, y ∈ {−s, 0, s}} do:
10:             if decode(q) < decode(p) then
11:                 color_p ← color_q
12: // Compute cost map
13: for all map pixels do:
14:     compute distance to color-coded cell coordinate
15:     compute cost value and write to output
```

For computing both the inflation cells and the decay function, the distance of each cell to the nearest obstacle is required. We use the Jump Flood Algorithm (JFA) [63] to compute a Voronoi diagram where each cell contains the coordinates of the nearest occupied cell, which can directly be converted to a distance transform map. For a map of size $n \times n$, the algorithm needs one shader pass for initialization, $\log n$ shader passes for propagating the coordinates of the nearest obstacles, and one shader pass to compute the Euclidean distance and the cost function. The resulting distance transform is not exact, as the error discussion in [63] shows, but we found that the errors are negligible in our application, especially as the cost map is only an approximation of the actual navigation cost as the true motion of a humanoid robot is much more complex.

Algorithm 6.1 gives a high-level overview of the algorithm. The basic idea of the JFA is to encode the coordinate of the nearest obstacle found so far as a color value in each cell and to propagate the information on the nearest obstacle in exponentially decreasing "jumps" across the map. Figure 6.2 illustrates the process in a toy example with a grid map containing three occupied cells. Rong Guodong analyzed the JFA in detail including its variants, potential failure cases, and applications in his PhD thesis [63] that we refer to for more details.

### 6.4.3 *Single-source shortest path*

Planning the shortest path from the robot's current location to one or more destinations is a basic component for many navigation tasks. While there exist efficient algorithms for solving the Single-Source Shortest Path (SSSP) problem, these algorithms rely on special data



(a) Occupancy map with obstacles. (b) Step length 8 for one cell. (c) Step length 8 for all cells.

(d) Step length 4 for one cell. (e) Step length 4 for all cells. (f) Step length 2 for one cell.

(g) Step length 2 for all cells. (h) Voronoi diagram. (i) Cost map.

Figure 6.2: Illustration of the jump flood algorithm computing a Voronoi diagram and cost map. (a) shows an occupancy map with three cells marked as obstacles. The light colors in (b)-(h) indicate the closest obstacle found so far. In each step, each cell is updated with the closest obstacle registered in the 8 horizontal, vertical, and diagonal neighbor cells "jumping" over steps of 8, 4, 2, and 1 cells. In (f), the Euclidean distance of the cell to the blue obstacle is smaller than the distance to the red obstacle, hence the cell is updated by registering the blue obstacle. The resulting Voronoi diagram (h) contains a reference to the nearest obstacle in each cell, which is used to compute the distance transform and cost map (i) (cyan: safety margin around the obstacles that the robot must not enter, yellow: costs decaying with distance to the nearest obstacle)

**Algorithm 6.2** Bellman-Ford algorithm for single-source shortest path computation

1: *// Initialization*
2: for all map pixels $p$ do in parallel:
3: $\quad dist_p \leftarrow \begin{cases} 0 & \text{if pixel is robot's current location} \\ \infty & \text{otherwise} \end{cases}$
4: $\quad changed_p \leftarrow \begin{cases} true & \text{if pixel is robot's location} \\ false & \text{otherwise} \end{cases}$
5: *// Wavefront propagation*
6: $\#changed \leftarrow 1$
7: while $\#changed > 0$ do:
8: $\quad \#changed \leftarrow 0$
9: $\quad$ for all map pixels $p$ do in parallel:
10: $\quad\quad changed_p \leftarrow false$
11: $\quad\quad$ for all neighbor pixels $q$ of $p$ do:
12: $\quad\quad\quad$ if $changed_q$ then
13: $\quad\quad\quad\quad d \leftarrow dist_q + \|p - q\| + cost_p$
14: $\quad\quad\quad\quad$ if $d < dist_p$ then
15: $\quad\quad\quad\quad\quad dist_p \leftarrow d$
16: $\quad\quad\quad\quad\quad changed_p \leftarrow true$
17: $\quad\quad$ if $changed_p$ then
18: $\quad\quad\quad$ atomically increment $\#changed$

structures such as Fibonacci heaps (e.g., Dijkstra's algorithm [53]) or bucketing structures (e.g., Thorup's algorithm [170]) that are not suitable for being implemented in a graphics pipeline. The core problem with these algorithms is that they rely on processing nodes sequentially in a particular order, whereas GPUs are most efficient at processing nodes in parallel.

The Bellman-Ford algorithm [17, 51], by contrast, can be parallelized. In a grid map with the same size as the occupancy grid map of the environment, we store the shortest-path distance from the robot's location. We initialize the robot's current grid cell with distance 0 and all other cells with $\infty$. For each grid cell, we then check which of the eight neighbor cells have changed and decrease the cell's distance in case a shorter path via a changed neighbor has been found. In this way, a wavefront of cells with decreasing distance emerges, and we repeat the process as long as cells change. The speed-up of using the GPU stems from processing all wavefront cells in parallel.

We present two implementations. In the first implementation in Algorithm 6.2, we use a "changed" flag to mark changed cells similar to the *EBellflaging* algorithm in [85]. We encode the "changed" flag as the sign and the current distance as the absolute value of a single-channel

**Algorithm 6.3** Bellman-Ford algorithm variant with transform feedback buffer (XFB)

1: *// Initialization*
2: for all map pixels $p$ do in parallel:
3: $dist_p \leftarrow \begin{cases} 0 & \text{if pixel is robot's current location} \\ \infty & \text{otherwise} \end{cases}$
4: $changed_p \leftarrow \begin{cases} true & \text{if pixel is robot's location} \\ false & \text{otherwise} \end{cases}$
5: *// Wavefront propagation*
6: emit robot's neighbor cell coordinates to XFB
7: while XFB is not empty do:
8: for all cells $p$ in XFB do in parallel:
9: $changed_p \leftarrow false$
10: for all neighbor pixels $q$ of $p$ do:
11: if $changed_q$ then
12: $d \leftarrow dist_q + \|p - q\| + cost_p$
13: if $d < dist_p$ then
14: $dist_p \leftarrow d$
15: $changed_p \leftarrow true$
16: if $changed_p$ then
17: emit neighbor cells of $p$ to XFB

integer texture. As we are only interested in the length of the shortest path, we do not store a pointer to the predecessor of each cell. OpenGL enforces protection against memory access collisions when processing pixels in parallel by denying fragment shaders to write to other pixels and by requiring separate buffers for reading and writing. Hence, we read from and write to two separate buffers and swap the buffers after each pass. We implemented the counter for counting changed pixels as an atomic counter and read back the counter value with a delay of three frames to prevent CPU-GPU synchronization points with pipeline flushes as explained in Section 6.4.1. The overhead of processing two extra shader passes after the wavefront has stopped is negligible in comparison to the loss of computation power that would occur with synchronous readbacks requiring to flush the graphics pipeline.

The disadvantage of the first implementation is that in each shader pass, the fragment shader has to check for each neighbor whether it has changed in the previous iteration. Reading neighbor cells require costly texture fetches. Modern OpenGL ES implementations support a *Transform Feedback Buffer* (XFB) that captures the output of the geometry shader in a buffer that can be used as input for the vertex shader in subsequent passes. In our second implementation in Algorithm 6.3, we enqueue the neighbor cells of a changed cell by emitting the neighbor

**Algorithm 6.4** Computing the information gain of a viewpoint candidate

```
1: enable depth test
2: render obstacles to depth buffer
3: render region of interest surfaces
4: for all front facing region of interest fragments p do in parallel:
5:     compute texture coordinate t corresponding to p
6:     store "observed" flag to region of interest texel t
7: #covered ← 0
8: for all region of interest texels t do in parallel:
9:     if t has "observed" flag then
10:        atomically increment #covered
```

cell's coordinate from the geometry shader. The next pass processes in parallel only the queued cells, reducing overhead. However, this technique requires both XFB and texture access in the geometry shader, which is not supported on some of the tested devices.

After computing the shortest distance map with either of the two Bellman-Ford implementations, we combine the distance map including the inflation costs from Algorithm 6.1 with the articulation costs, which are constant for all cells, to a combined cost map according to Equation 6.1.

### 6.4.4 *Information gain computation*

The information gain of a viewpoint is based on counting texels of the region of interest that are observable from the viewpoint and have not been seen before. When rendering a region of interest object, the fragment shader accesses the texel at the texture coordinates given in the fragment data to determine the fragment's color. Using the `imageStore` command present since OpenGL ES 3.1, the fragment shader can modify the texture at the same time. We use this mechanism to mark the texel as observed by writing a flag to the texture. By rendering the region of interest objects last with early depth tests and backface culling enabled, we make sure that the fragment shader only marks texels visible from the camera as observed. Counting the newly marked texels then yields the information gain value as Algorithm 6.4 shows.

### 6.4.5 *Estimating the utility map*

Algorithm 6.4 can be used to estimate the information gain for a single camera viewpoint. For choosing the next best view optimizing the utility function in Equation 6.2, we would need to compute the information gain for every possible camera location. Alternatively, the

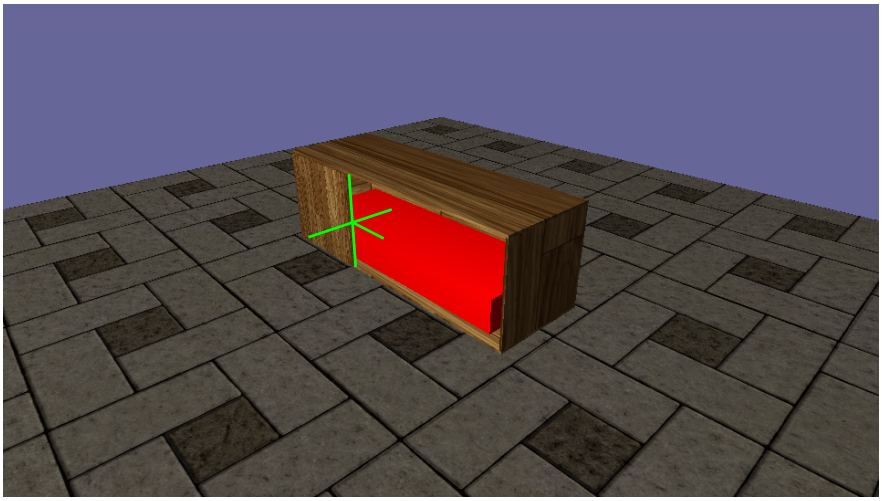
**Algorithm 6.5** Rendering a spherical panorama with semantic information.

```
1: enable depth test
2: render to 6-side cube map:
3:     obstacles (depth only)
4:     region of interest surfaces with texture from Algorithm 6.4
5:     reachable surfaces with index texture
6: for all color cube map pixels do
7:     project pixel to equirectangular coordinates
```

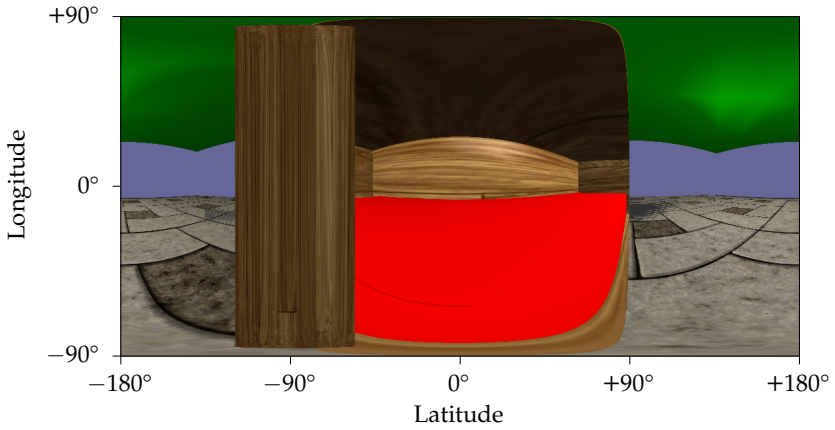
algorithm could render an image with virtual cameras placed at every possible region of interest surface location to determine the freespace volume from where the given region of interest texel can be observed. Both variants would require rendering a large number of images from different camera locations, causing high computational loads. Hence, we propose a novel heuristic for estimating an information gain map by rendering a spherical panorama image with a virtual camera placed at the edge of articulated objects or other obstacles. The core idea is that a spherical camera image placed in the opening of a drawer, cupboard, or other container shows the inside of the container on one side and the freespace volume from where the contents can be observed on the opposite side of the panorama image. Figure 6.3b shows an example panorama rendered by a virtual camera placed at the crosshair in Figure 6.3a.

Algorithm 6.5 describes the standard method of rendering a spherical panorama by first rendering all objects to the six sides of a cube map and then projecting the cube map texture to equirectangular coordinates. We first render all obstacles to a depth cube map only, as we do not need the color information of the obstacles. With depth testing enabled, we then render the region of interest surfaces with the texture from Algorithm 6.4 that indicates for each texel whether it has been observed before or not. Finally, we render the surface of reachable camera poses with a special texture that encodes the texture coordinate of each texel as its color value. This technique allows us to convert back from spherical coordinates to coordinates of the reachable surface later for projecting information gain values onto the cost map. Figure 6.3b shows the result for an example scene. If a region of interest texel $t$ and a camera viewpoint $v$ on the surface of reachable poses are on opposite sides of the sphere, that means that $t$ just becomes visible when the camera moves to $v$.

To estimate the information gain of moving the camera to $v$, we calculate an integral image by accumulating the number of region of interest pixels while pivoting a ray around the spherical camera's center. Figure 6.4 illustrates the concept of computing the integral images for two spherical panorama cameras $c_1$ and $c_2$. At the panorama image cell $A$, the container edge blocks the line of sight through the



(a) Scene with a cupboard. The region of interest is marked in red.




(b) Same scene rendered with a spherical panorama camera located at the crosshair in (a). The green surface indicates the reachable camera poses.

Figure 6.3: Example scene illustrating the process of estimating an information gain map. The scene is rendered as a spherical panorama image from virtual cameras placed at the edges of articulated objects. If a region of interest (red) is opposite the surface of reachable camera poses (green), then there is a direct line of sight. We use the panorama image to determine which region of interest parts are getting into the field of view when the camera moves on the reachable surface.

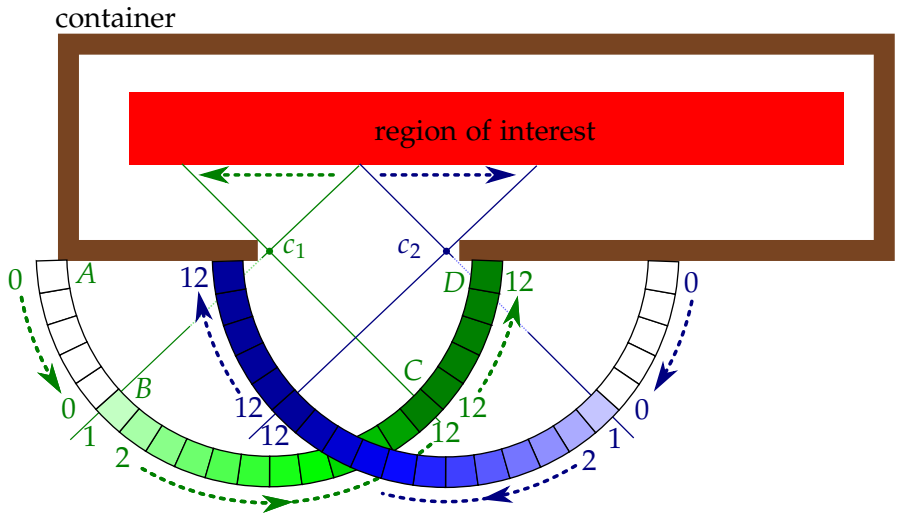



Figure 6.4: Estimating the information gain map by calculating the integral images of two spherical cameras $c_1$ and $c_2$. The numbers indicate the estimated information gain accumulated by pivoting rays through the camera center in the direction indicated by arrows.

camera origin to the region of interest. Pivoting counterclockwise around the camera center, the region of interest first becomes visible at $B$. Moving towards $C$, we increment the information gain estimate as long as new region of interest texels come into view. Between $C$ and $D$, the container edge again blocks the line of sight to new region of interest texels, hence the information gain stays constant. Adding the information gain estimates for the two cameras $c_1$ and $c_2$ gives a good approximation of the true information gain map without having to render panoramas at all camera positions between $c_1$ and $c_2$. Computing the integral image is done in the first part of Algorithm 6.6.

The remainder of the algorithm estimates the utility map. As we render the surface of reachable camera poses with a special index texture that encodes the texture coordinates on the surface in Algorithm 6.5, we can use these coordinates to compute the corresponding position on the cost map. If a pixel $p = (lat, lon)$ on the sphere image belongs to the reachable surface, the algorithm looks up the associated cost value from the cost map and subtracts it from the estimated information gain value encoded at the pixel $(\overline{lat}, \overline{lon})$ on the opposite side of the sphere, writing the output to a utility map with the same coordinate system as the cost map.

### 6.4.6 *Covering environments with a robot's camera*

We use the utility map estimated with Algorithm 6.6 to determine the next best view pose in a greedy iterative scheme. For each articulation object, the robot uses Algorithm 6.6 to estimate how the utility

**Algorithm 6.6** Estimating utility map

1: *// calculate integral image I*
2: for all $lon \in [-90°, 90°]$ do in parallel:
3: $\quad c\,(lon) \leftarrow 0$
4: for $lat \in \{-180°, \ldots, 180°\}$ do in sequence:
5: $\quad$ for all $lon \in [-90°, 90°]$ do in parallel:
6: $\quad\quad$ if pixel at $(lat, lon)$ is region of interest then
7: $\quad\quad\quad c\,(lon) \leftarrow c\,(lon) + 1$
8: $\quad\quad I\,(lat, lon) \leftarrow c\,(lon)$
9: *// calculate utility map U*
10: for all pixel $p$ at $lon \in [-90°, 90°]$, $lat \in [-180°, 180°]$ do in parallel:
11: $\quad coord \leftarrow$ color-decode$(p)$
12: $\quad$ if $p$ is marked as reachable surface then
13: $\quad\quad \overline{lat} \leftarrow lat + 180°$
14: $\quad\quad \overline{lon} \leftarrow -lon$
15: $\quad\quad U\,(lat, lon) \leftarrow I\left(\overline{lat}, \overline{lon}\right) - \text{cost}(coord)$
16: $\quad$ else
17: $\quad\quad U\,(lat, lon) \leftarrow -\text{cost}(coord)$

map changes when manipulating the object. Our algorithm currently actuates objects separately to avoid collisions with other articulated objects, as for example doors and drawers of cupboards can block each other. A high-level planner could be used to resolve this issue and generate feasible configurations of multiple objects. The algorithm then determines the articulation object and the robot position on the utility map with the highest estimated utility. To determine the best camera orientation, we sample orientations on a unit sphere and use Algorithm 6.4 to determine the orientation with the highest information gain.

## 6.5 EXPERIMENTAL EVALUATION

We implemented our approach in OpenGL ES 3.2 and evaluated the approach with models of home and office environments created with Computer-Aided Design (CAD) software. We first present the results of benchmarking the individual algorithms and then show experiments where a robot successfully explores an environment with our novel utility map heuristic.

### 6.5.1 *Benchmarking the GPU algorithms*

We designed our approach to be compatible with a wide range of devices ranging from embedded devices to multi-GPU servers, hence we evaluated the performance on different device classes. Table 6.1 lists

| Class | Test device |
|---|---|
| Smartphone | Device: Google Pixel<br>CPU: Qualcomm Snapdragon 821 (4 cores, 2.4 GHz)<br>GPU: Qualcomm Adreno 530<br>OpenGL profile: 3.2 ES |
| Notebook | CPU: Intel i7-4710MQ (4 cores, 8 threads, 3.5 GHz)<br>GPU: Intel Haswell Mobile HD 4600<br>OpenGL profile: 3.3 core |
| Desktop | CPU: Intel Core i7-3770 (4 cores, 8 threads, 3.4 GHz)<br>GPU: NVIDIA GeForce GTX 660 Ti<br>OpenGL profile: 4.3 core |
| Server | CPU: Intel Xeon E5-1630 (4 cores, 8 threads, 3.7 GHz)<br>GPU: 4×NVIDIA GeForce GTX 1080<br>OpenGL profile: 4.6 core |

Table 6.1: Devices used for benchmarking.

the technical specifications of the tested devices. We chose an Android smartphone as a representative of embedded devices and implemented the CPU-side code using the Android Native Development Toolkit (NDK) in C++. Apart from device-specific initialization, we use the same code base on all systems.

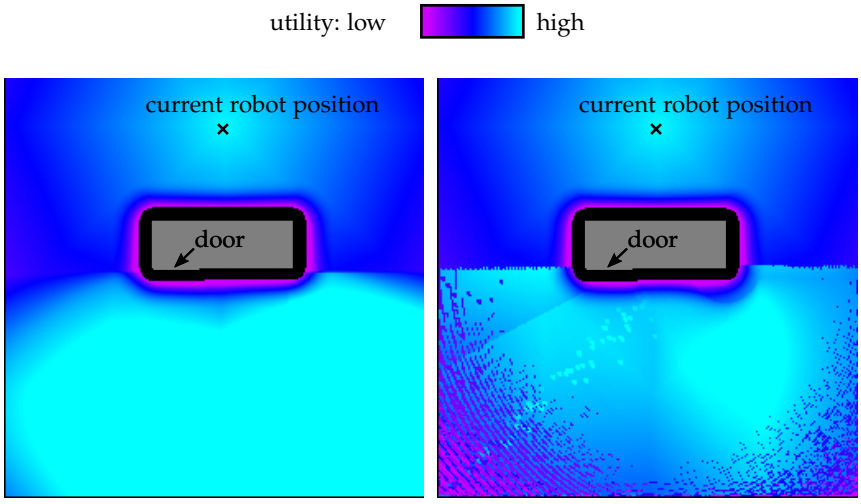
Unfortunately, the required OpenGL ES profiles are yet not available on the version 4 and 5 Nao robots that we use for our experiments. The upcoming version 6 of the Nao robots [5], however, will be equipped with an Intel Atom E3845 processor that supports the OpenGL 4.2 and OpenGL ES 3.0 profiles on Linux [78], hence we expect that our algorithms will be able to run on-board on the next Nao generation.

Table 6.2 shows the benchmarking results for tested each device. Algorithm 6.3 could not be tested on the notebook device due to missing support for image access in geometry shaders. While the OpenGL API is standardized, the GPU architectures and driver implementations differ significantly between manufacturers and device generations, hence the benchmarking results also vary strongly between devices. The algorithms for cost map computation, information gain estimation, and panorama rendering run in less than 3 ms on all devices, showing the potential of our approach. The standard deviation of computation times is very small in many cases, which is due to the design principle of reducing branching, making the run time independent of the input data.

| Algorithm | Smartphone | Notebook |
|---|---|---|
| Algorithm 6.1: cost map | $2.572 \pm 2.492$ ms | $0.142 \pm 0.010$ ms |
| Algorithm 6.2: Bellman-Ford | $272.9 \pm 21.05$ ms | $52.29 \pm 0.987$ ms |
| Algorithm 6.3: Bellman-Ford XFB | $251.4 \pm 26.87$ ms | N/A |
| Algorithm 6.4: information gain | $2.116 \pm 7.228$ ms | $0.092 \pm 0.006$ ms |
| Algorithm 6.5: panorama rendering | $0.667 \pm 0.541$ ms | $0.103 \pm 0.069$ ms |
| Algorithm 6.6: utility map | $575.2 \pm 32.17$ ms | $245.4 \pm 2.284$ ms |

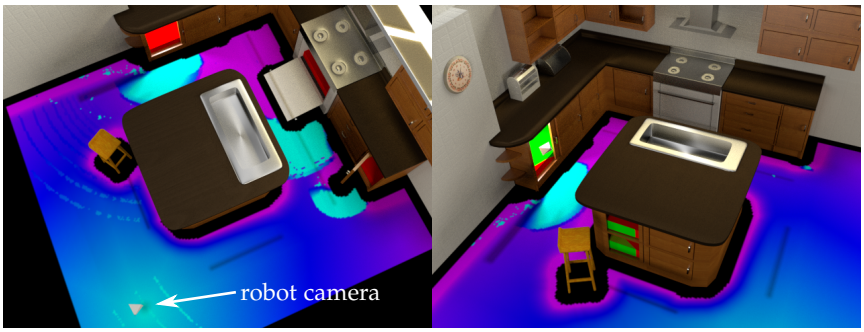
| Algorithm | Desktop | Server |
|---|---|---|
| Algorithm 6.1: cost map | $0.097 \pm 0.013$ ms | $0.071 \pm 0.002$ ms |
| Algorithm 6.2: Bellman-Ford | $10.92 \pm 0.103$ ms | $9.432 \pm 2.191$ ms |
| Algorithm 6.3: Bellman-Ford XFB | $11.08 \pm 0.274$ ms | $8.694 \pm 0.040$ ms |
| Algorithm 6.4: information gain | $0.099 \pm 0.016$ ms | $0.084 \pm 0.002$ ms |
| Algorithm 6.5: panorama rendering | $0.083 \pm 0.009$ ms | $0.060 \pm 0.024$ ms |
| Algorithm 6.6: utility map | $30.92 \pm 0.745$ ms | $12.34 \pm 0.723$ ms |

Table 6.2: Benchmark results: time per frame with standard deviation. See Table 6.1 for device specifications.

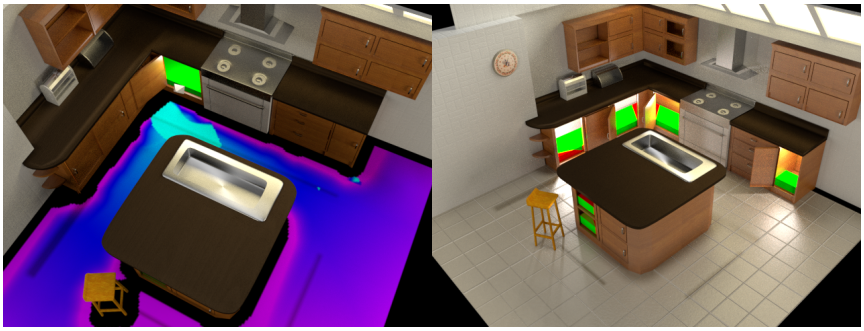



(a) Real utility map (b) Estimated utility map

Figure 6.5: Comparison of the real utility map obtained by exhaustive sampling and the estimated utility map generated by Algorithm 6.6 for the scene with a cupboard shown in Figure 6.3.




(a) Example utility map at start (b) First exploration step



(c) Second exploration step (d) Intermediate result after 7 steps

Figure 6.6: Progress of exploring a kitchen scene. The robot first estimates utility maps for multiple articulation configurations. The robot then chooses the articulation object and view point with the highest utility (b). Afterwards, the algorithm computes utility maps from the new location taking into account already observed regions. Regions of interest are marked in red, observed regions in green, and the robot's camera is symbolized as a white pyramid.

### 6.5.2 *Information gain map estimation*

Figure 6.5 shows a comparison of the estimated utility map and the real utility map generated by exhaustive sampling for the cupboard scene in Figure 6.3a. As can be seen, the estimated utility map is similar to the real utility map. Our heuristic, however, focuses on covering hard-to-reach corners due to the placement of the virtual spherical cameras at the edges of articulated objects. The information gain of center portions of regions of interest tend to be underestimated, but these regions are usually covered along the way when inspecting the rest of the region. Coverage of the center regions can be improved by adding additional panorama cameras inside the openings of articulated objects.

### 6.5.3 *Covering environments with a robot's camera*

Figure 6.6 shows the progress of a robot covering a kitchen environment. Starting from the initial pose in Figure 6.6a, the robot estimates a utility map for each articulated object in turn. The floor texture indicates the combined estimated utility map for three articulated objects partly opened. The robot then chooses the articulated object with the highest estimated utility, articulates the object, and investigates the scene from the location and camera orientation with the highest utility (Figure 6.6b). The observed regions of interest are marked in green. The average time required to compute the next-best-view position is $4.85\,\mathrm{s} \pm 0.29\,\mathrm{s}$ for this scene on the desktop computer specified in Table 6.1.

## 6.6 DISCUSSION

In this chapter, we presented algorithms for solving common navigation problems efficiently on GPUs and we showed that these building blocks can be integrated into a system for planning coverage tours through articulated, known environments. Despite these promising results, there is further space for improvement towards a system that can find the optimal solution in environments with many articulated objects.

In our approach, we used a greedy utility-based approach. As discussed in Section 5.6.3, this approach is most suitable for object search tasks where the most promising locations should be searched first. In applications where the whole environment has to be covered in any case, offline planning strategies such as the TSP approach used in the previous chapter are better suited as they minimize the tour length.

Articulated objects often block each other or have to be moved conjointly. The drawers of a dresser, for example, should not be opened at the same time, as the upper drawers would then obstruct access to the lower drawers. A cupboard with multiple sliding doors moving behind and in front of each other is another example of articulated objects with inherent constraints on their movement. To generate useful plans for operating these articulated objects with constraints, a symbolic planner could be integrated to determine the order in which to open and close the doors and drawers to get access to all compartments.

Another open question is how the robot can recognize which objects can be manipulated and how to manipulate them. Other research groups have presented solutions to this problem of determining "object affordances" including haptic exploration [32], detection of visual cues such as handles [24, 83], and learning kinematic models from

observations or kinesthetic teaching [164] that could potentially be integrated with our system.

In this chapter, we represented the reachable volume of the robot's camera with one or more planes parallel to the ground. To consider the additional constraints of a humanoid robot such as kinematic constraints, balance, and energy consumption, the inverse reachability map approach from Chapter 5 could be integrated into the system.

In future work, it might also be interesting to investigate whether our approach can be extended to other navigation problems, for example to path planning techniques or components of SLAM systems. To benefit from the full speed-up of hardware acceleration, however, it is necessary that *all* components of the system run on the GPU, as large data transfers between GPU and CPU generally slow down the system. The system presented in this chapter follows this principle by uploading all geometry and shaders to the GPU once in the beginning and minimizing all subsequent data transfer. In particular, our system does not download maps and object textures indicating the region of interest and its observed portions from the GPU, except for visualization if requested. Hence, combining components running on both GPU and CPU may be detrimental to performance due to expensive data transfers, which limits the possibilities for applying our approach to other problems as not all problems can be parallelized.

## 6.7 CONCLUSIONS

In this chapter, we proposed a novel approach for covering known scenes containing articulated objects that the robot has to manipulate for inspecting user-defined regions of interest. We presented algorithms for cost map estimation, path planning, and information gain estimation that run on GPUs. In this way, our system can parallelize the necessary computations to determine the next best view. We introduced a heuristic for estimating information gain maps based on spherical panoramic images and showed in simulation experiments that our approach enables a robot to successfully inspect home environments. Our algorithms run on GPUs for embedded systems and we showed benchmark results for multiple device classes.

# 7 CONCLUSIONS

In this thesis, we investigated how humans and robots can successfully collaborate in navigation tasks as a step towards universally programmable, intuitively usable service robots in everyday environments. Our focus was on how humans and robots can communicate about navigation tasks in a natural and intuitive way, and on how we can leverage the body plan of humanoid robots for executing tasks in environments designed for humans. The presented novel approaches for four different applications as well as a novel descriptor for RGB-D data and optimizations to an appearance-based SLAM system as a building block for robot navigation.

In the first two applications, we focused on the communication aspect. In the first application, the robot is the one with access to background information in form of a map and the robot's task is to describe a route for wayfinding to a human user. In the second application, the roles are reversed, as it is now the human who conveys information about the topological layout of an environment to a robot for making the robot's exploration task more efficient. In both cases, we presented communication methods that are intuitive and natural to use for the human user.

In the first application, we used natural language to communicate. We collected a corpus of human-written route descriptions in a user study and applied inverse reinforcement learning to learn a policy describing what amount and type of information the human teachers prefer in which context. This policy allowed the robot to generate descriptions for new routes that imitate the description style and content of human-written descriptions. By training on descriptions written for particular user groups such as visually impaired persons or children, the algorithm could adapt to the user group and tailor the information content to the needs of the recipients. In a user study, the participants rated the descriptions generated by our system as similarly natural compared to human-written descriptions and as significantly more natural than descriptions provided by existing commercial routing web services.

In the second application, we investigated how a robot can make use of the human user's knowledge about the topology of the environment during an exploration task. We assumed that the human user provides a topo-metric graph that the robot then exploits to generate an efficient global exploration strategy based on the solution of a Traveling Salesman Problem (TSP). The background knowledge about the coarse layout of the environment allowed the robot to make informed

decisions about which areas to explore first so that it will not have to return there later, thus avoiding unnecessary detours. Combined with a local nearest-frontier-first exploration scheme, the robot could efficiently explore the environment. Our experiments with artificial and real-world environments showed that our approach leads to significantly shorter exploration tours and that our approach is robust against inaccuracies and noise. We also presented recovery strategies in case the topological graph is incorrect, for example due to blocked passages.

In the remaining two applications, we investigated the closely related problem of coverage planning in a known environment. We assumed that the robot has access to a 3D model of the environment, either captured in a previous SLAM run or provided by the user. The robot's task was to cover all surfaces that the user marked in the model with its camera as efficiently as possible, for example for searching an object or inspecting facilities for damage. In Chapter 5, we extended an existing approach based on sampled raycasting by integrating an Inverse Reachability Map (IRM). Querying a pre-recorded IRM allowed the planner to consider balance, energy, and full-body collision constraints of humanoid robots early in the planning process when searching for camera view poses from where large portions of the environment are visible. Once the algorithm had found a suitable set of full-body poses for viewing as much as possible of the environment, a TSP planner provided the shortest tour for visiting these camera view points.

In Chapter 6, we extended our coverage planning approach from static scenes to articulated environments where the robot has to actively move objects and furniture, for example to open doors and drawers to inspect cupboards and dressers. Due to the increased complexity of the planning problem, this application required fast approaches for cost map computing, path planning, and information gain estimation. Hence, we introduced implementations for these subproblems that leverage the parallelization power of graphics processing units. In benchmark experiments, we showed that our algorithms run efficiently on multiple devices classes including embedded systems. We presented a novel heuristic for estimating utility maps based on spherical panoramic images and integrated these components into a greedy next-best-view planning system. Our experiments showed that our approach enables a robot to successfully inspect a home environment.

As accurate localization is a prerequisite for all robot navigation tasks, we presented improvements to an appearance-based SLAM system in Chapter 4. We introduced a novel feature descriptor for RGB-D data and showed that the descriptor outperforms the precision of existing descriptors while keeping recall levels high. We integrated the descriptor in an existing SLAM system and proposed modifications to

adapt the system to the particular challenges of humanoid robots. Our experiments on benchmark datasets and real-world datasets recorded with a Nao robot, we showed that our modifications make the SLAM system more accurate and robust.

In summary, this thesis presented techniques and solutions to the following questions:

- How can automated systems learn from human experts how to generate route descriptions that better fit the needs of humans?
- How can robots leverage background knowledge provided by human users to explore unknown environments more efficiently?
- How can humanoid robots plan view points and tours to cover a known environment efficiently and use their whole body to inspect hidden areas?
- How can robots plan coverage tours in environments with articulated objects that they have to manipulate to get access to occluded regions?
- How can the parallelization power of GPUs on embedded systems be used to accelerate navigation tasks?

We believe that the approaches presented in this thesis advance the state of the art towards robots that are universally programmable, can be operated by non-expert users, and can accomplish tasks in environments designed for humans.

## OUTLOOK

The work presented in this thesis offers a starting point for further research in the areas of robot navigation in human environments and communication with humans about navigation concepts. We already discussed limitations and potential extensions of our approaches in the individual chapters. In this section, we will describe ideas how the approaches presented in this thesis can be extended and integrated into a bigger picture.

In this thesis, we investigated how humans and robots can communicate about navigation tasks through verbal descriptions and graphs drawn by the user. As we have shown, inverse reinforcement learning is a promising technique to make human-robot communication more natural and intuitive for the human users. This approach could be transferred to other tasks beyond navigation. The recently announced Google Duplex system [98], for example, can automatically place phone calls to accomplish tasks such as scheduling an appointment with a hair salon or book a table at a restaurant with a human answering the phone. While the full technical details have not been published at the time of writing this thesis, the article introducing the system [98]

mentions that it uses a Recurrent Neural Network (RNN) trained on a corpus of recorded phone conversations and that "real-time supervised training" by a human expert is used to improve the quality of the system. It would be worth to investigate whether the system can benefit from inverse reinforcement learning for selecting the amount and type of information to convey in such automated phone calls.

Beyond natural language, we should also consider additional communication modes to improve the communication and to make robots more accessible to non-expert users. Most prominently, all applications discussed in this thesis would benefit from non-verbal communication through gestures. Research in spatial cognition has shown that gestures are an intuitive and expressive device for giving wayfinding instructions [7, 157] and Okuno et al. [122] have shown the importance of gestures also in human-robot wayfinding interaction. Pointing in the direction where the user has to walk first is an intuitive way to align the reference frames of the recipient and the person giving the route directions, and pointing left or right helps to reduce left-right confusion. In a scenario where a human user guides a robot through an environment unknown to the robot, the user could use gestures to give background information, for example information on adjacent rooms, similar to the approach that Bastianelli et al. propose in [13]. In the coverage tour planning application, a humanoid robot could report its findings to the human user using gestures, for example by pointing at the object that the human asked the robot to search for. Hence, approaches on recognizing and producing gestures with humanoid robots could be integrated to improve the communication.

Besides gestures, other communication devices could be integrated as well. Robots could use screens for presenting information, for example on the Pepper humanoid's built-in tablet computer, on TV screens present in home environments, or on smartphones, for example for showing photos of landmarks along the route when giving directions as Hile et al. [72] propose.

# A

# APPENDIX

## A.1 NAO HUMANOID ROBOT: TECHNICAL DATA

In Chapters 4 and 5, we used the humanoid robot Nao for our real-world experiments. The robot is manufactured by Aldebaran Robotics (now part of SoftBank Robotics). In our work, we use versions 4 and 5 of the robot.

| Nao version 4 and 5 | |
|---|---|
| height | 57.3 cm |
| weight | 5.2 kg |
| degrees of freedom | 25 total (2 in neck, 1 in pelvis, 5 per arm, 5 per leg, 1 per hand) |
| operating system | Embedded GNU/Linux, distribution based on Gentoo |
| CPU | Intel Atom Z530 (single core, 1.60 GHz) |
| GPU | Intel GMA 500 on Intel System Controller Hub US15W, supports OpenGL 2.0 |
| cameras | 2 cameras, bottom camera tilted 40° downwards, model MT9M114, resolution 1280 px × 960 px at 30 fps, field of view 60.9° horizontal, 47.6° vertical, 72.6° diagonal, fixed focus |
| inertial unit | 2-axis (version 4) or 3-axis (vesrion 5) gyrometer, 3-axis accelerometer |
| sonar | 2 devices located in chest, resolution 1 cm to 4 cm, usable detection range 0.20 m to 0.80 m, effective cone 60° |
| weight sensors | 4 force sensitive resistors per foot, range 0 N to 25 N |
| joint sensors | 36 magnetic rotary encoders using hall effect sensor technology, resolution 0.1° |

Table A.1: Nao robot technical specifications. For more details see the manufacturer's data sheet [3]

## A.2 ASUS XTION PRO LIVE RGB-D CAMERA: TECHNICAL DATA

For our real-world experiments in Chapters 4 and 5, we used an ASUS Xtion Pro Live RGB-D camera mounted on top of a Nao's head.

| ASUS Xtion Pro Live | |
|---|---|
| RGB camera resolution | 1280 px × 1024 px (in our experiments: 640 px × 480 px) |
| depth camera resolution | 640 px × 480 px |
| field of view | 58° horizontal, 45° vertical, 70° diagonal |
| distance of use | 0.8 m to 3.5 m |
| power consumption | below 2.5 W |

Table A.2: ASUS Xtion Pro Live technical specifications. For more details see the manufacturer's data sheet [10]

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ALGORITHMS

# ACRONYMS

| | |
|---|---|
| API | Application Programming Interface |
| APX | Constant-factor Approximation Algorithms (complexity class) |
| ATE | Absolute Trajectory Error |
| BOW | Bag of Words |
| BRAND | Binary Robust Appearance and Normals Descriptor |
| BRIEF | Binary Robust Independent Elementary Features |
| CAD | Computer-Aided Design |
| CNN | Convolutional Neural Network |
| CPU | Central Processing Unit |
| CSHOT | Color Signature of Histograms of Orientations |
| DBOW2 | Dynamic Bag of Words |
| EKF | Extended Kalman Filter |
| FAB-MAP | Fast Appearance-Based Mapping |
| FAST | Features from Accelerated Segment Test |
| GPGPU | General-Purpose Computation on Graphics Processing Units |
| GPS | Global Positioning System |
| GPU | Graphics Processing Unit |
| IG | Information Gain |
| IMU | Inertial Measurement Unit |
| IRL | Inverse Reinforcement Learning |
| IRM | Inverse Reachability Map |
| LIDAR | Light Detection and Ranging |
| JFA | Jump Flood Algorithm |
| MAXENT IRL | Maximum Entropy Inverse Reinforcement Learning |
| MDP | Markov Decision Process |

| | |
|---|---|
| MI | Mutual Information |
| MTSP | Multiple Traveling Salesman Problem |
| NDK | Android Native Development Toolkit |
| NP | Non-deterministic Polynomial Time (complexity class) |
| OPENCL | Open Computing Language |
| OPENGL ES | Open Graphics Library for Embedded Systems |
| OPENGL | Open Graphics Library |
| ORB | Oriented FAST and Rotated BRIEF |
| PCL | Point Cloud Library |
| PTAM | Parallel Tracking and Mapping |
| RANSAC | Random Sample Consensus |
| RGB | Red Green Blue (color space) |
| RGB-D | Red Green Blue and Depth (color space) |
| RM | Reachability Map |
| RMSE | Root Mean Squared Error |
| ROS | Robot Operating System |
| RNN | Recurrent Neural Network |
| SIFT | Scale-Invariant Feature Transform |
| SIMD | Single Instruction Multiple Data |
| SLAM | Simultaneous Localization and Mapping |
| SSSP | Single-Source Shortest Path |
| SURF | Speeded Up Robust Features |
| SVD | Singular Value Decomposition |
| TSP | Traveling Salesman Problem |
| XFB | Transform Feedback Buffer |

# BIBLIOGRAPHY


[1] P. Abbeel and A. Ng. "Apprenticeship learning via inverse reinforcement learning." In: *Proc. of the Int. Conf. on Machine Learning (ICML)*. 2004.
DOI: 10.1145/1015330.1015430.

[2] P. Abbeel, A. Coates, and A. Y. Ng. "Autonomous helicopter aerobatics through apprenticeship learning." In: *Int. Journal of Robotics Research* 29.13 (2010), pp. 1608–1639.
DOI: 10.1177/0278364910371999.

[3] Aldebaran Robotics. *Datasheet NAO Next Gen – H21/H25 Model*. 2013.
URL: https://www.softbankrobotics.com/emea/sites/aldebaran/files/datasheet_nao_next_gen_en.pdf.

[4] Aldebaran Robotics. *Whole Body control – Aldebaran 2.1.4.13 documentation*. 2015.
URL: http://doc.aldebaran.com/2-1/naoqi/motion/control-wholebody.html.

[5] Aldebaran Robotics. *Nao Technical Overview: Motherboard – Aldebaran 2.8.2.15 Documentation*. 2018.
URL: http://doc.aldebaran.com/2-8/family/nao_technical/motherboard_naov6.html.

[6] G. L. Allen. "From knowledge to words to wayfinding: Issues in the production and comprehension of route directions." In: *Proc. of the Int. Conf. on Spatial Information Theory (COSIT)*. London, UK: Springer-Verlag, 1997, pp. 363–372. ISBN: 3-540-63623-4.
DOI: 10.1007/3-540-63623-4_61.

[7] G. L. Allen. "Gestures accompanying verbal route directions: Do they point to a new avenue for examining spatial representations?" In: *Spatial Cognition & Computation* 3.4 (2003), pp. 259–268.
DOI: 10.1207/s15427633scc0304_1.

[8] G. M. Amdahl. "Validity of the single processor approach to achieving large scale computing capabilities." In: *Proc. of the AFIPS Spring Joint Computer Conference*. Vol. 30. 1967, pp. 483–485.
DOI: 10.1145/1465482.1465560.

[9] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. "TSP cuts which do not conform to the template paradigm." In: *Computational Combinatorial Optimization*. Ed. by M. Jünger and D. Naddef. Vol. 2241. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001, pp. 261–304. ISBN: 978-3-540-42877-0. DOI: 10.1007/3-540-45586-8_7.

[10] ASUSTeK Computer Inc. *Xtion PRO LiVE Datasheet*. 2018. URL: https://www.asus.com/3D-Sensor/Xtion_PRO_LIVE/specifications/.

[11] B. Bacca, J. Salvi, and X. Cufí. "Appearance-based slam for mobile robots." In: *Frontiers in Artificial Intelligence and Applications* 202 (2009), pp. 55–64. ISSN: 0922-6389. DOI: 10.3233/978-1-60750-061-2-55.

[12] N. Basilico and F. Amigoni. "Exploration strategies based on multi-criteria decision making for searching environments in rescue operations." In: *Autonomous Robots* 31.4 (2011), pp. 401–417. ISSN: 0929-5593. DOI: 10.1007/s10514-011-9249-9.

[13] E. Bastianelli, D. D. Bloisi, R. Capobianco, F. Cossu, G. Gemignani, L. Iocchi, and D. Nardi. "On-line semantic mapping." In: *Proc. of the Int. Conf. on Advanced Robotics (ICAR)*. 2013. DOI: 10.1109/icar.2013.6766501.

[14] H. Bay, T. Tuytelaars, and L. V. Gool. "SURF: Speeded-up robust features." In: *Proc. of the European Conference on Computer Vision (ECCV)* (2006). ISSN: 1077-3142. DOI: 10.1016/j.cviu.2007.09.014.

[15] L. van Beek, D. Holz, M. Matamoros, C. Rascon, and S. Wachsmuth. *RoboCup@Home 2018: Rules and Regulations*. 2018. URL: http://www.robocupathome.org/rules/.

[16] R. Bellman. "A Markovian decision process." In: *Indiana University Mathematics Journal* 6 (4 1957), pp. 679–684. ISSN: 0022-2518.

[17] R. Bellman. "On a routing problem." In: *Quarterly of Applied Mathematics* 16 (1958), pp. 87–90. ISSN: 0033-569X.

[18] R. Bernardi, R. Cakici, D. Elliott, A. Erdem, E. Erdem, N. Ikizler-Cinbis, F. Keller, A. Muscat, and B. Plank. "Automatic description generation from images: A survey of models, datasets, and evaluation measures." In: *Journal of Artificial Intelligence Research* 55 (2016), pp. 409–442. DOI: 10.1613/jair.4900.

[19] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart. "Receding horizon "next-best-view" planner for 3D exploration." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2016, pp. 1462–1468.
DOI: 10.1109/ICRA.2016.7487281.

[20] L. Bishop, C. Kubisch, and M. Schott. "High-performance, low-overhead rendering with OpenGL and Vulkan." In: *Proc. of the Game Developers Conference (GDC)*. 2016.

[21] C. F. Bissmarck, M. Svensson, and G. Tolt. "Efficient algorithms for next best view evaluation." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. 2015.
DOI: 10.1109/IROS.2015.7354212.

[22] Blendswap user "cenobi". *"Bathroom"*. Blender model, available under a Creative Commons Attribution (CC-BY 3.0) license. 2012.
URL: https://www.blendswap.com/blends/view/52486.

[23] Blendswap user "ThePefDispenser". *"Library-Home Office"*. Blender model, available under a Creative Commons Attribution (CC-BY 3.0) license. 2017.
URL: https://www.blendswap.com/blends/view/88906.

[24] N. Blodow, L. C. Goron, Z.-C. Marton, D. Pangercic, T. Ruhr, M. Tenorth, and M. Beetz. "Autonomous semantic mapping for robots performing everyday manipulation tasks in kitchen environments." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. 2011.
DOI: 10.1109/iros.2011.6094665.

[25] F. Boniardi, B. Behzadian, W. Burgard, and G. D. Tipaldi. "Robot navigation in hand-drawn sketched maps." In: *Proc. of the European Conference on Mobile Robots (ECMR)*. 2015.
DOI: 10.1109/ECMR.2015.7324188.

[26] F. Boniardi, A. Valada, W. Burgard, and G. D. Tipaldi. "Autonomous indoor robot navigation using a sketch interface for drawing maps and routes." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2016.
DOI: 10.1109/ICRA.2016.7487453.

[27] B. Brumitt and A. Stentz. "GRAMMPS: A generalized mission planner for multiple mobile robots." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 1998, pp. 1564–1571.
DOI: 10.1109/ROBOT.1998.677360.

[28] F. Burget and M. Bennewitz. "Stance selection for humanoid grasping tasks by inverse reachability maps." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. Seattle, USA, 2015, pp. 5669–5674.
DOI: 10.1109/ICRA.2015.7139993.

[29] C. Camporesi and M. Kallmann. "Computing shortest path maps with GPU shaders." In: *Proc. of the Int. Conf. on Motion in Games (MIG)*. 2014.
DOI: 10.1145/2668064.2668092.

[30] N. Christofides. *Worst-case analysis of a new heuristic for the travelling salesman problem*. Tech. rep. 388. Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[31] G. Chronis and M. Skubic. "Sketch-based navigation for mobile robots." In: *Proc. of the IEEE Int. Conf. on Fuzzy Systems (FUZZ)*. 2003, pp. 284–289.
DOI: 10.1109/FUZZ.2003.1209376.

[32] V. Chu and A. L. Thomaz. "Understanding the role of haptics in affordances." In: *RSS Workshop on Affordances in Vision for Cognitive Robotics*. 2014.

[33] H. Cuayáhuitl, N. Dethlefs, L. Frommberger, K.-F. Richter, and J. Bateman. "Generating adaptive route instructions using hierarchical reinforcement learning." In: *Proc. of Spacial Cognition*. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 319–334.
DOI: 10.1007/978-3-642-14749-4_27.

[34] M. Cummins and P. Newman. "FAB-MAP: Probabilistic localization and mapping in the space of appearance." In: *Int. Journal of Robotics Research* 27.6 (2008).
DOI: 10.1177/0278364908090961.

[35] M. Cummins and P. Newman. "Appearance-only SLAM at large scale with FAB-MAP 2.0." In: *The International Journal of Robotics Research* 30.9 (2011), pp. 1100–1123.
DOI: 10.1177/0278364910385483.

[36] R. Dale, S. Geldof, and J. Prost. "Using natural language generation in automatic route description." In: *Journal of Research and Practice in Information Technology* 37.1 (2005).

[37] A. F. Daniele, M. Bansal, and M. R. Walter. "Navigational instruction generation as inverse reinforcement learning with neural machine translation." In: *Proc. of the ACM/IEEE Int. Conf. on Human-Robot Interaction (HRI)*. ACM Press, 2017.
DOI: 10.1145/2909824.3020241.

[38] J. Daudelin and M. Campbell. "An adaptable, probabilistic, next best view algorithm for reconstruction of unknown 3D objects." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2017.
DOI: 10.1109/LRA.2017.2660769.

[39] A. Davidson, S. Baxter, M. Garland, and J. D. Owens. "Work-efficient parallel GPU methods for single-source shortest paths." In: *Proc. of the IEEE Int. Parallel and Distributed Processing Symp. (IPDPS)*. 2014.
DOI: 10.1109/ipdps.2014.45.

[40] C. Dornhege and A. Kleiner. "A frontier-void-based approach for autonomous exploration in 3D." In: *Advanced Robotics* 27.6 (2013), pp. 459–468.
DOI: 10.1080/01691864.2013.763720.

[41] C. Dornhege, A. Kleiner, and A. Kolling. "Coverage search in 3D." In: *Proc. of the IEEE Int. Symp. on Safety, Security, and Rescue Robotics (SSRR)*. 2013, pp. 1–8.
DOI: 10.1109/SSRR.2013.6719340.

[42] C. DuHadway. *The ROS exploration stack*. 2012.
URL: http://wiki.ros.org/exploration.

[43] 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 Int. Conf. on Robotics & Automation (ICRA)*. 2012.
DOI: 10.1109/ICRA.2012.6225199.

[44] J. Faigl, M. Kulich, and L. Preucil. "Goal assignment using distance cost in multi-robot exploration." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. 2012, pp. 3741–3746.
DOI: 10.1109/IROS.2012.6385660.

[45] L. Férmin-León, J. Neira, and J. A. Castellanos. "TIGRE: Topological graph based robotic exploration." In: *Proc. of the European Conference on Mobile Robots (ECMR)*. IEEE, 2017.
DOI: 10.1109/ecmr.2017.8098718.

[46] N. Figueroa, H. Dong, and A. El Saddik. "A combined approach toward consistent reconstructions of indoor spaces based on 6D RGB-D odometry and KinectFusion." In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 6.2 (2015).
DOI: 10.1145/2629673.

[47] L. G. Fischer, R. Silveira, and L. Nedel. "GPU accelerated path-planning for multi-agents in virtual environments." In: *Brazilian Symp. on Games and Digital Entertainment*. 2009.
DOI: 10.1109/sbgames.2009.20.

[48] T. Foissotte, O. Stasse, A. Escande, P. Wieber, and A. Kheddar. "A two-steps next-best-view algorithm for autonomous 3D object modeling by a humanoid robot." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2009, pp. 1159–1164.
DOI: 10.1109/ROBOT.2009.5152350.

[49] T. Foissotte, O. Stasse, P. Wieber, A. Escande, and A. Kheddar. "Autonomous 3D object modeling by a humanoid using an optimization-driven next-best view formulation." In: *Int. Journal of Humanoid Robotics, Special issue on Cognitive Humanoid Vision* 7.3 (2010), pp. 407–428.
DOI: 10.1142/S0219843610002246.

[50] T. Fong, C. E. Thorpe, and C. Baur. "Collaboration, dialogue, human-robot interaction." In: *Proc. of the Int. Symposium of Robotics Research (ISRR)*. Ed. by R. A. Jarvis and A. Zelinsky. Vol. 6. Springer Tracts in Advanced Robotics. Springer Berlin Heidelberg, 2003, pp. 255–266. ISBN: 978-3-540-00550-6.
DOI: 10.1007/3-540-36460-9_17.

[51] L. R. Ford. *Network flow theory*. Paper. RAND Corporation, 1956.

[52] D. Fox, J. Ko, K. Konolige, and B. Stewart. "A hierarchical Bayesian approach to the revisiting problem in mobile robot map building." In: *Proc. of the Int. Symposium of Robotics Research (ISRR)*. 2003, pp. 60–69.
DOI: 10.1007/11008941_7.

[53] M. Fredman and R. Tarjan. "Fibonacci heaps and their uses in improved network optimization algorithms." In: *Annual Symp. on Foundations of Computer Science*. 1984.
DOI: 10.1109/sfcs.1984.715934.

[54] C. Freksa, R. Moratz, and T. Barkowsky. "Schematic maps for robot navigation." In: *Spatial Cognition II, Integrating Abstract Theories, Empirical Studies, Formal Methods, and Practical Applications*. London, UK: Springer-Verlag, 2000, pp. 100–114. ISBN: 3-540-67584-1.
DOI: 10.1007/3-540-45460-8_8.

[55] D. Gálvez-López and J. Tardos. "Bags of binary words for fast place recognition in image sequences." In: *IEEE Trans. on Robotics (TRO)* 28.5 (2012).
DOI: 10.1109/ICRA.2012.6224843.

[56] B. P. Gerkey and M. J. Mataric. "Sold!: Auction methods for multirobot coordination." In: *IEEE Transactions on Robotics and Automation* 18.5 (2002), pp. 758–768. ISSN: 1042-296X.
DOI: 10.1109/TRA.2002.803462.

[57] B. Gerkey, R. T. Vaughan, and A. Howard. "The Player/Stage project: Tools for multi-robot and distributed sensor systems." In: *Proc. of the Int. Conf. on Advanced Robotics (ICAR)*. 2003, pp. 317–323.

[58] A. Glover, W. Maddern, M. Warren, S. Reid, M. Milford, and G. Wyeth. "OpenFABMAP: An open source toolbox for appearance-based loop closure detection." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2012.
DOI: 10.1109/ICRA.2012.6224843.

[59] R. Goeddel and E. Olson. "DART: A particle-based method for generating easy-to-follow directions." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. 2012, pp. 1213–1219.
DOI: 10.1109/IROS.2012.6385471.

[60] A. Gonzalez. *"Kitchen Nr 2"*. Blender model, available under a Creative Commons Attribution (CC-BY 3.0) license. 2013.
URL: https://www.blendswap.com/blends/view/70272.

[61] Google. *Google Maps*. 2018.
URL: https://maps.google.com.

[62] G. Grisetti, L. Iocchi, B. Leibe, V. Ziparo, and C. Stachniss. "Digitization of inaccessible archeological sites with autonomous mobile robots." In: *Conference on Robotics Innovation for Cultural Heritage*. 2012.

[63] R. Guodong. "Jump flooding algorithm on graphics hardware and its applications." PhD thesis. National University of Singapore, 2007.

[64] D. Ha and D. Eck. "A neural representation of sketch drawings." In: *Proc. of the Int. Conf. on Learning Representations (ICLR)*. 2018.

[65] H. J. Hannay, P. J. Ciaccia, J. W. Kerr, and D. Barrett. "Self-report of right-left confusion in college men and women." In: *Perceptual and Motor Skills* 70.2 (1990), 451–457E.
DOI: 10.2466/pms.1990.70.2.451.

[66] R. Hänsch, T. Weber, and O. Hellwich. "Comparison of 3D interest point detectors and descriptors for point cloud fusion." In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 2.3 (2014).
DOI: 10.5194/isprsannals-II-3-57-2014.

[67] S. Haque, L. Kulik, and A. Klippel. "Algorithms for reliable navigation and wayfinding." In: *Spatial Cognition V Reasoning, Action, Interaction*. Ed. by T. Barkowsky, M. Knauff, G. Ligozat, and D. Montello. Vol. 4387. Lecture Notes in Computer Science. Springer Verlag, 2007, pp. 308–326. ISBN: 978-3-540-75665-1.
DOI: 10.1007/978-3-540-75666-8_18.

[68] J. Hardy. *"The White Room"*. Blender model, available under a Creative Commons Attribution (CC-BY 3.0) license. 2012.
URL: https://www.blendswap.com/blends/view/41683.

[69] K. Helsgaun. "An effective implementation of the Lin-Kernighan traveling salesman heuristic." In: *European Journal of Operational Research* 126.1 (2000), pp. 106–130.
DOI: 10.1016/S0377-2217(99)00284-2.

[70] 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." In: *Int. Journal of Robotics Research* 31.5 (2012).
DOI: 10.1177/0278364911434148.

[71] M. Herlihy and N. Shavit. *The Art of Multiprocessor Programming, Revised Reprint*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012. ISBN: 9780123973375.

[72] H. Hile, R. Vedantham, G. Cuellar, A. Liu, N. Gelfand, R. Grzeszczuk, and G. Borriello. "Landmark-based pedestrian navigation from collections of geotagged photos." In: *Proc. of the Int. Conf. on Mobile and Ubiquitous Multimedia (MUM)*. ACM Press, 2008.
DOI: 10.1145/1543137.1543167.

[73] D. Holz, N. Basilico, F. Amigoni, and S. Behnke. "A comparative evaluation of exploration strategies and heuristics to improve them." In: *Proc. of the European Conference on Mobile Robots (ECMR)*. Örebro, Sweden, 2011, pp. 25–30.

[74] A. Hornung, A. Dornbush, M. Likhachev, and M. Bennewitz. "Anytime search-based footstep planning with suboptimality bounds." In: *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*. 2012, pp. 674–679.
DOI: 10.1109/HUMANOIDS.2012.6651592.

[75] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. "OctoMap: An efficient probabilistic 3D mapping framework based on octrees." In: *Autonomous Robots* 34.3 (2013). Software available at http://octomap.github.com, pp. 189–206.
DOI: 10.1007/s10514-012-9321-0.

[76] A. Hornung, S. Oßwald, D. Maier, and M. Bennewitz. "Monte Carlo localization for humanoid robot navigation in complex indoor environments." In: *Int. Journal of Humanoid Robotics* 11.2 (2014).
DOI: 10.1142/S0219843614410023.

[77] A. M. Hund, M. Schmettow, and M. L. Noordzij. "The impact of culture and recipient perspective on direction giving in the service of wayfinding." In: *Journal of Environmental Psychology* 32.4 (2012), pp. 327–336. ISSN: 0272-4944.
DOI: 10.1016/j.jenvp.2012.05.007.

[78] Intel Corp. *Intel® Atom™ Processor E3800 Product Family and Intel® Celeron® Processor N2807/N2930/J1900 – User Guide for Yocto Project* Board Support Package (BSP) Graphics Driver*. 2014.

[79] S. Isler, R. Sabzevari, J. A. Delmerico, and D. Scaramuzza. "An information gain formulation for active volumetric 3D reconstruction." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2016, pp. 3477–3484. DOI: 10.1109/ICRA.2016.7487527.

[80] E. T. Jaynes. "Where do we stand on maximum entropy." In: *Maximum Entropy Formalism* (1978). Ed. by R. D. Levine and M. Tribus, pp. 15–118.

[81] D. B. Johnson. "Efficient algorithms for shortest paths in sparse networks." In: *ACM* 24.1 (1977), pp. 1–13. ISSN: 0004-5411. DOI: 10.1145/321992.321993.

[82] D. Joho, M. Senk, and W. Burgard. "Learning search heuristics for finding objects in structured environments." In: *Robotics & Autonomous Systems* 59.5 (2011), pp. 319–328. DOI: 10.1016/j.robot.2011.02.012.

[83] P. Kaiser and T. Asfour. "Autonomous detection and experimental validation of affordances." In: *IEEE Robotics and Automation Letters (RA-L)* 3.3 (2018), pp. 1949–1956. DOI: 10.1109/lra.2018.2808367.

[84] P. Karkowski, S. Oßwald, and M. Bennewitz. "Real-time footstep planning in 3D environments." In: *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*. 2016, pp. 69–74. DOI: 10.1109/HUMANOIDS.2016.7803256.

[85] N. Kaushik and A. Kaushik. "Extended Bellman Ford algorithm with optimized time of computation." In: *Advances in Intelligent Systems and Computing*. Springer Singapore, 2016, pp. 241–247. DOI: 10.1007/978-981-10-0135-2_23.

[86] J. Ko, B. Stewart, D. Fox, K. Konolige, and B. Limketkai. "A practical, decision-theoretic approach to multi-robot mapping and exploration." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. Vol. 4. Las Vegas, NV, USA, 2003, pp. 3232–3238. DOI: 10.1109/IROS.2003.1249654.

[87] N. Koenig and A. Howard. "Design and use paradigms for Gazebo, an open-source multi-robot simulator." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. 2004, pp. 2149–2154.

[88] T. Kollar. "Learning to understand spatial language for robotic navigation and mobile manipulation." PhD thesis. Cambridge, USA: Massachusetts Institute of Technology, 2011.

[89] M. Krainin, B. Curless, and D. Fox. "Autonomous generation of complete 3D object models using next best view manipulation planning." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2011, pp. 5031–5037.
DOI: 10.1109/ICRA.2011.5980429.

[90] S. Kriegel, T. Bodenmüller, M. Suppa, and G. Hirzinger. "A surface-based next-best-view approach for automated 3D model completion of unknown objects." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2011, pp. 4869–4874.
DOI: 10.1109/ICRA.2011.5979947.

[91] G. M. Kruijff et al. "Rescue robots at earthquake-hit Mirandola, Italy: A field report." In: *Proc. of the IEEE Int. Symp. on Safety, Security, and Rescue Robotics (SSRR)*. 2012, pp. 1–8.
DOI: 10.1109/SSRR.2012.6523866.

[92] M. Kuderer, H. Kretzschmar, C. Sprunk, and W. Burgard. "Feature-based prediction of trajectories for socially compliant navigation." In: *Proc. of Robotics: Science and Systems (RSS)*. Sydney, Australia, 2012.
DOI: 10.15607/RSS.2012.VIII.025.

[93] R. Kuemmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. "g²o: A general framework for graph optimization." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2011.
DOI: 10.1109/ICRA.2011.5979949.

[94] H. Kuhn. "The Hungarian method for the assignment problem." In: *Naval Research Logistics Quarterly* 2.1 (1955), pp. 83–97.

[95] B. Kuipers and Y. Byun. "A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations." In: *Journal of Robotics & Autonomous Systems* 8.1-2 (1991), pp. 47–63.
DOI: 10.1016/0921-8890(91)90014-C.

[96] M. Kulich, J. Faigl, and L. Přeučil. "On distance utility in the exploration task." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. IEEE, 2011.
DOI: 10.1109/icra.2011.5980221.

[97] C. S. G. Lee and C. T. Lin. "Parallel algorithms and fault-tolerant reconfigurable architecture for robot kinematics and dynamics computations." In: *Advances in Robotic Systems*. Ed. by C. T. Leondes. Vol. 40. Control and Dynamic Systems 2. Elsevier, 1991, pp. 33–103.
DOI: 10.1016/b978-0-12-012740-5.50007-4.

[98] Y. Leviathan and Y. Matias. *Google Duplex: An AI System for Accomplishing Real-World Tasks Over the Phone*. Google Inc. 2018.
URL: https://ai.googleblog.com/2018/05/duplex-ai-system-for-natural-conversation.html.

[99] G. W. K. Look. "Cognitively-inspired direction giving." PhD thesis. Cambridge, USA: Massachusetts Inst. of Technology, 2008.

[100] K. L. Lovelace, M. Hegarty, and D. R. Montello. "Elements of good route directions in familiar and unfamiliar environments." In: *Proc. of the Int. Conf. on Spatial Information Theory (COSIT)*. London, UK: Springer-Verlag, 1999, pp. 65–82. ISBN: 3-540-66365-7.
DOI: 10.1007/3-540-48384-5_5.

[101] D. Lowe. "Object recognition from local scale-invariant features." In: *Proc. of the Int. Conf. on Computer Vision (ICCV)*. IEEE, 1999.
DOI: 10.1109/iccv.1999.790410.

[102] D. V. Lu, D. Hershberger, and W. D. Smart. "Layered costmaps for context-sensitive navigation." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. 2014.
DOI: 10.1109/iros.2014.6942636.

[103] M. T. MacMahon. "Following natural language route instructions." PhD thesis. Electrical and Computer Engineering Department, University of Texas at Austin, 2007.

[104] A. Makarenko, S. B. Williams, F. Bourgault, and H. F. Durrant-Whyte. "An experiment in integrated exploration." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. 2002, pp. 534–539.
DOI: 10.1109/IRDS.2002.1041445.

[105] R. Mannadiar and I. M. Rekleitis. "Optimal coverage of a known arbitrary environment." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2010, pp. 5525–5530.
DOI: 10.1109/ROBOT.2010.5509860.

[106] MapQuest. *Official MapQuest*. 2018.
URL: https://www.mapquest.com/.

[107] D. M. Mark. "Automated route selection for navigation." In: *Aerospace and Electronic Systems Magazine, IEEE* 1.9 (1986), pp. 2–5. ISSN: 0885-8985.
DOI: 10.1109/MAES.1986.5005198.

[108] P. J. Martín, R. Torres, and A. Gavilanes. "CUDA solutions for the SSSP problem." In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2009, pp. 904–913.
DOI: 10.1007/978-3-642-01970-8_91.

[109] A. Martínez, C. Domínguez, H. Hassan, J.-M. Martínez, and P. López. "Using GPU and SIMD implementations to improve performance of robotic emotional processes." In: *IEEE Int. Workshop on Multicore and Multithreaded Architectures and Algorithms (M2A2)*. 2015.
DOI: 10.1109/hpcc-css-icess.2015.288.

[110] J. McDonald. "Avoiding catastrophic performance loss: Detecting CPU-GPU sync points." In: *Proc. of the Game Developers Conference (GDC)*. 2014.

[111] D. Merrill, M. Garland, and A. Grimshaw. "Scalable GPU graph traversal." In: *ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*. 2012.
DOI: 10.1145/2370036.2145832.

[112] O. Michel. "Webots: Professional mobile robot simulation." In: *Journal of Advanced Robotics Systems* 1.1 (2004), pp. 39–42.
DOI: 10.5772/5618.

[113] Microsoft Corporation. *Bing Maps*. 2018.
URL: https://www.bing.com/maps.

[114] R. Mur-Artal and J. D. Tardós. "ORB-SLAM2: An open-source SLAM system for monocular, stereo and RGB-D cameras." In: *IEEE Trans. on Robotics (TRO)* 33.5 (2017).
DOI: 10.1109/TRO.2017.2705103.

[115] R. Mur-Artal, J. Montiel, and J. D. Tardós. "ORB-SLAM: A versatile and accurate monocular SLAM system." In: *IEEE Trans. on Robotics (TRO)* 31.5 (2015).
DOI: 10.1109/TRO.2015.2463671.

[116] R. Mur-Artal and J. D. Tardós. "Fast relocalisation and loop closing in keyframe-based SLAM." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2014.
DOI: 10.1109/ICRA.2014.6906953.

[117] E. Nascimento, G. Oliveira, M. Campos, A. Vieira, and W. Schwartz. "BRAND: A robust appearance and depth descriptor for RGB-D images." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. 2012.
DOI: 10.1109/IROS.2012.6385693.

[118] R. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. "KinectFusion: Real-time dense surface mapping and tracking." In: *Proc. of the IEEE/ACM Int. Symp. on Mixed and Augmented Reality (ISMAR)*. 2011.
DOI: 10.1109/ISMAR.2011.6092378.

[119] A. Y. Ng and S. J. Russell. "Algorithms for inverse reinforcement learning." In: *Proc. of the Int. Conf. on Machine Learning (ICML)*. Morgan Kaufmann Publishers, 2000, pp. 663–670. ISBN: 1-55860-707-2.

[120] T. Nguyễn. *"Living Room"*. Blender model, available under a Creative Commons Zero (CC0 1.0) license. 2013.
URL: https://www.blendswap.com/blends/view/70842.

[121] C. Nothegger, S. Winter, and M. Raubal. "Selection of salient features for route directions." In: *Spatial Cognition & Computation* 4.2 (2004), pp. 113–136.
DOI: 10.1207/s15427633scc0402_1.

[122] Y. Okuno, T. Kanda, M. Imai, H. Ishiguro, and N. Hagita. "Providing route directions: Design of robot's utterance, gesture, and timing." In: *Proc. of the ACM/IEEE Int. Conf. on Human-Robot Interaction (HRI)*. ACM Press, 2009.
DOI: 10.1145/1514095.1514108.

[123] G. Oriolo, A. Paolillo, L. Rosa, and M. Vendittelli. "Humanoid odometric localization integrating kinematic, inertial and visual information." In: *Autonomous Robots* 40.5 (2016).
DOI: 10.1007/s10514-015-9498-0.

[124] J. O'Rourke. *Art Gallery Theorems and Algorithms*. New York, NY, USA: Oxford University Press, Inc., 1987. ISBN: 0-19-503965-3.

[125] S. Oßwald and M. Bennewitz. "GPU-accelerated next-best-view coverage of articulated scenes." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. In press. 2018.

[126] S. Oßwald, A. Hornung, and M. Bennewitz. "Learning reliable and efficient navigation with a humanoid." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2010, pp. 2375–2380.
DOI: 10.1109/ROBOT.2010.5509420.

[127] S. Oßwald, A. Görög, A. Hornung, and M. Bennewitz. "Autonomous climbing of spiral staircases with humanoids." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. 2011, pp. 4844–4849.
DOI: 10.1109/IROS.2011.6094533.

[128] S. Oßwald, J. Gutmann, A. Hornung, and M. Bennewitz. "From 3D point clouds to climbing stairs: A comparison of plane segmentation approaches for humanoids." In: *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*. 2011, pp. 93–98.
DOI: 10.1109/Humanoids.2011.6100836.

[129] S. Oßwald, A. Hornung, and M. Bennewitz. "Improved proposals for highly accurate localization using range and vision data." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. 2012, pp. 1809–1814.
DOI: 10.1109/IROS.2012.6385657.

[130] S. Oßwald, H. Kretzschmar, W. Burgard, and C. Stachniss. "Learning to give route directions from human demonstrations." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2014, pp. 3303–3308.
DOI: 10.1109/ICRA.2014.6907334.

[131] S. Oßwald, M. Bennewitz, W. Burgard, and C. Stachniss. "Speeding-up robot exploration by exploiting background information." In: *IEEE Robotics and Automation Letters (RA-L)* 1.2 (2016), pp. 716–723. ISSN: 2377-3766.
DOI: 10.1109/LRA.2016.2520560.

[132] S. Oßwald, P. Karkowski, and M. Bennewitz. "Efficient coverage of 3D environments with humanoid robots using inverse reachability maps." In: *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*. 2017, pp. 151–157.
DOI: 10.1109/humanoids.2017.8239550.

[133] E. Palazzolo and C. Stachniss. "Information-driven autonomous exploration for a vision-based MAV." In: *Proc. of the ISPRS Int. Conf. on Unmanned Aerial Vehicles in Geomatics (UAV-g)*. 2017.
DOI: 10.5194/isprs-annals-IV-2-W3-59-2017.

[134] J. Pan, S. Chitta, and D. Manocha. "FCL: A general purpose library for collision and proximity queries." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2012, pp. 3859–3866.
DOI: 10.1109/ICRA.2012.6225337.

[135] F. Pasqualetti, A. Franchi, and F. Bullo. "On cooperative patrolling: Optimal trajectories, complexity analysis, and approximation algorithms." In: *IEEE Transactions on Robotics* 28.3 (2012), pp. 592–606. ISSN: 1552-3098.
DOI: 10.1109/TRO.2011.2179580.

[136] D. Perea Strom, F. Nenci, and C. Stachniss. "Predictive exploration considering previously mapped environments." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2015, pp. 2761–2766.
DOI: 10.1109/ICRA.2015.7139574.

[137] H. A. Pierson and M. S. Gashler. "Deep learning in robotics: A review of recent research." In: *Advanced Robotics* 31.16 (2017), pp. 821–835.
DOI: 10.1080/01691864.2017.1365009.

[138] D. Portugal and R. Rocha. "A survey on multi-robot patrolling algorithms." In: *Technological Innovation for Sustainability*. Ed. by L. M. Camarinha-Matos. Vol. 349. IFIP Advances in Information and Communication Technology. Springer Berlin Heidelberg, 2011, pp. 139–146. ISBN: 978-3-642-19169-5.
DOI: 10.1007/978-3-642-19170-1_15.

[139] A. Pretto, E. Menegatti, M. Bennewitz, W. Burgard, and E. Pagello. "A visual odometry framework robust to motion blur." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2009, pp. 2250–2257.
DOI: 10.1109/ROBOT.2009.5152447.

[140] A. Pronobis. "Semantic Mapping with Mobile Robots." PhD thesis. Stockholm, Sweden: KTH Royal Institute of Technology, 2011. ISBN: 978-91-7501-039-7.

[141] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. "ROS: An open-source Robot Operating System." In: *ICRA Workshop on Open Source Software*. 2009.

[142] J. Quinlan. "Induction of decision trees." In: *Machine Learning* 1.1 (1986), pp. 81–106. ISSN: 0885-6125.

[143] P. Regier, S. Oßwald, P. Karkowski, and M. Bennewitz. "Foresighted navigation through cluttered environments." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. 2016, pp. 1437–1442.
DOI: 10.1109/IROS.2016.7759234.

[144] K.-F. Richter. *Context-specific route directions: Generation of cognitively motivated wayfinding instructions*. Dissertationen zur künstlichen Intelligenz. Akademische Verlagsgesellschaft (Aka), 2007. ISBN: 978-1-58603-852-6.

[145] K.-F. Richter and M. Duckham. "Simplest instructions: Finding easy-to-describe routes for navigation." In: *Proc. of the Int. Conf. on Geographic Information Science (GIScience)*. Park City, UT, USA: Springer-Verlag, 2008, pp. 274–289. ISBN: 978-3-540-87472-0.
DOI: 10.1007/978-3-540-87473-7_18.

[146] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. "ORB: An efficient alternative to SIFT or SURF." In: *Proc. of the Int. Conf. on Computer Vision (ICCV)*. 2011.
DOI: 10.1109/ICCV.2011.6126544.

[147] R. B. Rusu and S. Cousins. "3D is here: Point Cloud Library (PCL)." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. IEEE, 2011.
DOI: 10.1109/icra.2011.5980567.

[148] L. T. Sarjakoski, P. Kettunen, H.-M. Flink, M. Laakso, M. Rönneberg, and T. Sarjakoski. "Analysis of verbal route descriptions and landmarks for hiking." In: *Personal and Ubiquitous Computing* 16.8 (2011), pp. 1001–1011.
DOI: 10.1007/s00779-011-0460-7.

[149] D. Shah, J Schneider, and M Campbell. "A robust sketch interface for natural robot control." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. 2010.
DOI: 10.1109/iros.2010.5649345.

[150] D. C. Shah and M. E. Campbell. "A qualitative path planner for robot navigation using human-provided maps." In: *Int. Journal of Robotics Research* 32.13 (2013), pp. 1517–1535. DOI: 10.1177/0278364913496485.

[151] R. Sheikh, S. Oßwald, and M. Bennewitz. "A combined RGB and depth descriptor for SLAM with humanoids." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. In press. 2018.

[152] S. Shen, N. Michael, and V. Kumar. "Autonomous indoor 3D exploration with a micro-aerial vehicle." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2012, pp. 9–15. DOI: 10.1109/ICRA.2012.6225146.

[153] J. G. Siek, L.-Q. Lee, and A. Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual*. Boston, MA, USA: Addison-Wesley Professional, 2002. ISBN: 0-201-72914-8.

[154] R. Sim and N. Roy. "Global a-optimal robot exploration in SLAM." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. Barcelona, Spain, 2005, pp. 661–666. DOI: 10.1109/ROBOT.2005.1570193.

[155] M. Skubic, P. Matsakis, B. Forrester, and G. Chronis. "Extracting navigation states from a hand-drawn map." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. IEEE, 2001. DOI: 10.1109/robot.2001.932563.

[156] M. Skubic, D. Anderson, S. Blisard, D. Perzanowski, and A. Schultz. "Using a hand-drawn sketch to control a team of robots." In: *Autonomous Robots* 22.4 (2007), pp. 399–410. DOI: 10.1007/s10514-007-9023-1.

[157] W. C. So, T. H.-W. Ching, P. E. Lim, X. Cheng, and K. Y. Ip. "Producing gestures facilitates route learning." In: *PLoS ONE* 9.11 (2014). Ed. by M. W. Greenlee. DOI: 10.1371/journal.pone.0112543.

[158] SQUIRREL – Clearing Clutter Bit by Bit. *Nao cleaning up*. Official website of the SQUIRREL EU Project. URL: http://squirrel-project.eu.

[159] C. Stachniss, G. Grisetti, and W. Burgard. "Information gain-based exploration using Rao-Blackwellized particle filters." In: *Proc. of Robotics: Science and Systems (RSS)*. Cambridge, MA, USA, 2005, pp. 65–72. DOI: 10.15607/RSS.2005.I.009.

[160] C. Stachniss, Ó. M. Mozos, and W. Burgard. "Speeding-up multi-robot exploration by considering semantic place information." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2006, pp. 1692–1697. DOI: 10.1109/ROBOT.2006.1641950.

[161] C. Stachniss, Ó. M. Mozos, and W. Burgard. "Efficient exploration of unknown indoor environments using a team of mobile robots." In: *Annals of Mathematics and Artificial Intelligence* 52.2-4 (2 2008), pp. 205–227.
DOI: 10.1007/s10472-009-9123-z.

[162] O. Stasse, A. J. Davison, R. Sellaouti, and K. Yokoi. "Real-time 3D SLAM for humanoid robot considering pattern generator information." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. 2006, pp. 348–355.
DOI: 10.1109/IROS.2006.281645.

[163] O. Stasse, T. Foissotte, D. Larlus, A. Kheddar, and K. Yokoi. "Treasure hunting for humanoids robot." In: *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*. Workshop on Cognitive Humanoid Vision. Daejeon, South Korea, 2008.

[164] J. Sturm. "Approaches to Probabilistic Model Learning for Mobile Manipulation Robots." PhD thesis. Germany: University of Freiburg, 2011.

[165] 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 Int. Conf. on Intelligent Robots & Systems (IROS)*. 2012.
DOI: 10.1109/IROS.2012.6385773.

[166] G. Suddrey, A. Jacobson, and B. Ward. "Enabling a Pepper robot to provide automated and interactive tours of a robotics laboratory." In: *Computing Research Repository (CoRR)*. 2018.

[167] N. Suenderhauf, S. Shirazi, A. Jacobson, F. Dayoub, E. Pepperell, B. Upcroft, and M. Milford. "Place recognition with ConvNet landmarks: Viewpoint-robust, condition-robust, training-free." In: *Proc. of Robotics: Science and Systems (RSS)*. 2015.
DOI: 10.15607/RSS.2015.XI.022.

[168] G. Taylor and L. Kleeman. *Visual Perception and Robotic Manipulation – 3D Object Recognition, Tracking and Hand-Eye Coordination*. Vol. 26. STAR Springer tracts in advanced robotics. Springer Berlin Heidelberg, 2006. ISBN: 978-3-540-33454-5.
DOI: 10.1007/978-3-540-33455-2.

[169] Team NimbRo@Home. *Cosero opening a fridge*. Autonomous Intelligent Systems, University of Bonn.
URL: https://www.ais.uni-bonn.de/nimbro/@Home.

[170] M. Thorup. "Undirected single-source shortest paths with positive integer weights in linear time." In: *Journal of the ACM* 46.3 (1999), pp. 362–394.
DOI: 10.1145/316542.316548.

[171] S. Thrun, S. Thayer, W. Whittaker, C. R. Baker, W. Burgard, D. Ferguson, D. Hähnel, M. D. Montemerlo, A. Morris, Z. Omohundro, and C. F. Reverte. "Autonomous exploration and mapping of abandoned mines." In: *IEEE Robotics and Automation* 11.4 (2005), pp. 79–91.
DOI: 10.1109/MRA.2004.1371614.

[172] F. Tombari, S. Salti, and L. Di Stefano. "A combined texture-shape descriptor for enhanced 3D feature matching." In: *Proc. of the IEEE Int. Conf. on Image Processing (ICIP)*. 2011.
DOI: 10.1109/ICIP.2011.6116679.

[173] N. Vahrenkamp, T. Asfour, G. Metta, G. Sandini, and R. Dillmann. "Manipulability analysis." In: *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*. IEEE, 2012.
DOI: 10.1109/humanoids.2012.6651576.

[174] N. Vahrenkamp, D. Muth, P. Kaiser, and T. Asfour. "IK-Map: An enhanced workspace representation to support inverse kinematics solvers." In: *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*. 2015, pp. 785–790.
DOI: 10.1109/HUMANOIDS.2015.7363443.

[175] J. I. Vasquez-Gomez, L. E. Sucar, R. Murrieta-Cid, and E. Lopez-Damian. "Volumetric next-best-view planning for 3D object reconstruction with positioning error." In: *Int. Journal of Advanced Robotics Systems* 11.159 (2014).
DOI: 10.5772/58759.

[176] S. Venkataraman. "Programming multi-GPUs for scalable rendering." In: *Proc. of the GPU Technology Conference (GTC)*. 2012.

[177] D. Waller and Y. Lippa. "Landmarks as beacons and associative cues: Their role in route learning." In: *Memory & Cognition* 35.5 (2007), pp. 910–924. ISSN: 0090-502X.
DOI: 10.3758/BF03193465.

[178] S. Wang, R. Clark, H. Wen, and N. Trigoni. "DeepVO: Towards end-to-end visual odometry with deep recurrent convolutional neural networks." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2017.
DOI: 10.1109/ICRA.2017.7989236.

[179] Y.-S. Wang, Y.-X. Gai, and F.-Y. Wu. "A robot kinematics simulation system based on OpenGL." In: *IEEE Int. Conf. on Robotics, Automation and Mechatronics (RAM)*. 2011.
DOI: 10.1109/ramech.2011.6070474.

[180] S. L. Ward, N. Newcombe, and W. F. Overton. "Turn left at the church, or three miles north: A study of direction giving and sex differences." In: *Environment and Behavior* 18.2 (1986), pp. 192–213.
DOI: 10.1177/0013916586182003.

[181] H. Westerbeek and A. Maes. "Route-external and route-internal landmarks in route descriptions: Effects of route length and map design." In: *Applied Cognitive Psychology* 27.3 (2013), pp. 297–305.
DOI: 10.1002/acp.2907.

[182] 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.

[183] M. Wolschon et al. *Traveling Salesman – open source navigation and routing-application for OpenStreetMap*. Software available under the GPLv3 license.
URL: https://sourceforge.net/projects/travelingsales/.

[184] K. M. Wurm, C. Stachniss, and W. Burgard. "Coordinated multi-robot exploration using a segmentation of the environment." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. 2008, pp. 1160–1165.
DOI: 10.1109/IROS.2008.4650734.

[185] A. Xu, C. Viriyasuthee, and I. Rekleitis. "Efficient complete coverage of a known arbitrary environment with applications to aerial operations." In: *Autonomous Robots* 36.4 (2014), pp. 365–381. ISSN: 0929-5593.
DOI: 10.1007/s10514-013-9364-x.

[186] L. Xu and A. Stentz. "A fast traversal heuristic and optimal algorithm for effective environmental coverage." In: *Proc. of Robotics: Science and Systems (RSS)*. 2010.
DOI: 10.15607/RSS.2010.VI.021.

[187] B. Yamauchi. "Frontier-based exploration using multiple robots." In: *Proc. of the Int. Conf. on Autonomous Agents*. 1998, pp. 47–53.
DOI: 10.1145/280765.280773.

[188] T. Yoshikawa. "Manipulability of robotic mechanisms." In: *Int. Journal of Robotics Research* 4.2 (1985), pp. 3–9.
DOI: 10.1177/027836498500400201.

[189] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey. "Maximum entropy inverse reinforcement learning." In: *Proc. of the National Conference on Artificial Intelligence (AAAI)*. Vol. 3. Chicago, Illinois, 2008, pp. 1433–1438. ISBN: 978-1-57735-368-3.

[190] B. D. Ziebart, A. L. Maas, A. K. Dey, and J. A. Bagnell. "Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior." In: *Proc. of the Int. Conf. on Ubiquitous Computing (Ubicomp)*. 2008, pp. 322–331.
DOI: 10.1145/1409635.1409678.

[191] B. D. Ziebart, N. D. Ratliff, G. Gallagher, C. Mertz, K. M. Peterson, J. A. Bagnell, M. Hebert, A. K. Dey, and S. S. Srinivasa. "Planning-based prediction for pedestrians." In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*. 2009, pp. 3931–3936.
DOI: 10.1109/IROS.2009.5354147.

[192] R. Zlot, A. Stentz, M. B. Dias, and S. Thayer. "Multi-robot exploration controlled by a market economy." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. Vol. 3. Washington, DC, USA, 2002, pp. 3016–3023.
DOI: 10.1109/ROBOT.2002.1013690.