Humanoid Robot Navigation in Complex Indoor Environments

Armin Hornung

Technische Fakultät
Albert-Ludwigs-Universität Freiburg im Breisgau

UNI FREIBURG

Dissertation zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften

Betreuerin: Juniorprof. Dr. Maren Bennewitz

März 2014

Humanoid Robot Navigation in Complex Indoor Environments

Armin Hornung

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

Dekan:	Prof. Dr. Yiannos Manoli
Erstgutachterin:	Juniorprof. Dr. Maren Bennewitz Albert-Ludwigs-Universität Freiburg im Breisgau
Zweitgutachter:	Dr. Olivier Stasse LAAS – CNRS, Toulouse, Frankreich
Tag der Disputation:	24.03.2014

Abstract

Humanoid service robots promise a high adaptability to environments designed for humans due to their human-like body layout. They are thus well-suited to assist humans in domestic environments for everyday tasks or providing elderly care, but also to replace human workers in hazardous environments. The human-like structure allows versatile manipulation with two arms as well as stepping over or onto obstacles with bipedal locomotion. Compared to wheeled platforms, however, the kinematic structure of humanoids requires active balancing while walking and allows only a limited payload, e.g., for additional sensors. Inaccurate motion execution can lead to foot slippage and thus to a poor estimation of ego-motion. Furthermore, the high number of degrees of freedom make planning and control a challenging problem for humanoid robots.

Autonomous navigation is a core capability of any service robot that performs high-level tasks involving mobility. Before a robot can fulfill a task, it needs to know where it is in the environment, what the environment looks like, and how it can successfully reach its goal. In this thesis, we present several novel contributions to the field of humanoid robotics with a focus on navigation in complex environments. We hereby cope with challenges in the areas of 3D environment representations, localization, perception, motion planning, and mobile manipulation.

As a basis, we first introduce a memory-efficient 3D environment representation along with techniques for building three-dimensional maps. In this representation, our probabilistic localization approach estimates the 6D pose of the humanoid based on data of its noisy onboard sensors. We compare different range sensors as well as sensor and motion models. For the critical task of climbing stairs, we develop an improved particle filter that additionally integrates vision data for a highly accurate pose estimate. We also present methods to perceive staircases in 3D range data. For reaching a navigation goal with a humanoid robot, we introduce and compare different search-based footstep planning approaches with anytime capabilities. We then investigate planning for the whole body of the humanoid while considering constraints such as maintaining balance during a manipulation task. This enables the robot to pick up objects or open doors. Finally, we present an efficient approach for navigation in three-dimensional cluttered environments that is particularly suited for mobile manipulation.

All techniques developed in this thesis were thoroughly evaluated both with real robots and in simulation. Our contributions generally advance the navigation capabilities of humanoid robots in complex indoor environments.

Zusammenfassung

Der weitläufige Einsatz von Automatisierung und Industrierobotern in den letzten Jahrzehnten hat zu einem signifikanten Anstieg der Produktivität von Fertigungsprozessen geführt. Aktuelle Industrieroboter sind hochspezialisiert auf einzelne Aufgaben und können diese mit einem hohen Maß an Genauigkeit, Wiederholbarkeit und Durchsatz ausüben. Zukünftige Generationen von Robotern werden sich neueste Erkenntnisse der künstlichen Intelligenz und des maschinellen Sehens zu Nutze machen und dabei auf eine gesteigerte Rechenleistung zurückgreifen. Sie haben dadurch das Potential noch nützlicher zu werden und Menschen universell in einem komplexen Umfeld zu unterstützen. Zum Beispiel könnten Serviceroboter im Haushalt zur Hand gehen oder gar bei der Altenpflege behilflich sein, was durch den demographischen Wandel immer wichtiger wird. Roboter könnten in der Zukunft aber auch Leben retten, indem sie menschliche Arbeitskräfte in gefährlichen Umgebungen ersetzen oder gar Katastrophenhilfe leisten.

Diesen Szenarien ist gemeinsam, dass sie hochgradig komplex und unstrukturiert sind, und dass die Umgebung größtenteils für den Menschen geschaffen wurde. Türen, Treppen und Leitern können von uns Menschen problemlos bewältigt werden, stellen jedoch für einen Radroboter Hindernisse dar. Ein Roboter mit einem Körperbau ähnlich zu dem eines Menschen kann sich hingegen besser an diese Umgebungen anpassen. Die Hauptmerkmale eines menschenähnlichen, auch anthropomorph genannten Körperbaus sind zwei Arme zum Greifen und Manipulieren von Objekten sowie zwei Beine, welche einen menschenähnlichen Gang ermöglichen. Dadurch kann ein zweibeiniger Roboter eine kontinuierliche Bewegung für seinen Oberkörper mittels diskreter Fußkontakte auf dem Untergrund erzeugen. Diskrete Fußkontakte erlauben es dem Roboter, unstetigen Untergrund zu überschreiten, indem er seine Schritte auf Objekte oder um Hindernisse herum setzt.

Der vergleichsweise komplexe Körperbau von humanoiden Robotern hat jedoch auch Nachteile. Die vielen Freiheitsgrade sind eine Herausforderung für die Bewegungsplanung und Kontrolle von humanoiden Robotern. Während des Laufens muss der Roboter aktiv seine Balance halten um nicht zu fallen. Eine ungenaue Bewegungsausführung kann zu kleinen Fehltritten oder Rutschen führen und dadurch zu einer ungenauen Schätzung der Eigenbewegung. Außerdem können humanoide Roboter oft nur wenig Nutzlast tragen und benötigen deshalb leichte Sensoren. Diese Sensoren sind aber oft nur vergleichsweise ungenau.

In der Forschung wurde das Potential, aber auch die Herausforderung erkannt, was zu einem Trend hin zu humanoiden Robotern führte. Dieser wurde zum Teil durch den "Robotics Challenge" Wettbewerb der DARPA verstärkt, in welchem ein Roboter halb-

autonom in einem eingestürzten Fabrik- oder Kraftwerksgebäude navigieren soll um Katastrophenhilfe zu leisten. Da hier sowohl die Umgebung als auch die Werkzeuge für den Einsatz durch Menschen vorgesehen sind, greifen aktuell die meisten Wettbewerbsteilnehmer auf einen humanoiden Roboter zurück.

Die autonome Navigation ist eine Kernkompetenz für jeden Roboter der allgemeine Aufgaben erledigen soll, welche ein gewisses Maß an Mobilität erfordern. Bevor der Roboter eine Aufgabe erledigen kann, muss er wissen wo er sich befindet, wie die Umgebung aussieht und wie er sein Ziel erreichen kann. Trotz dieser Notwendigkeit und den damit verbundenen Herausforderungen gibt es bisher kaum Forschungsergebnisse die sich mit der Navigation von humanoiden Robotern befassen. Der Hauptbeitrag der vorliegenden Arbeit ist es, humanoide Roboter mit Fertigkeiten zur Navigation in komplexen Umgebungen auszustatten. Dabei werden die Themengebiete der dreidimensionalen Umgebungsrepräsentation, Lokalisierung, Wahrnehmung, Bewegungsplanung sowie der mobilen Manipulation behandelt.

Zusammenfassend werden in der vorliegenden Arbeit die folgenden Fragestellungen beantwortet:

- Wie kann ein Roboter eine dreidimensionale Repräsentation seiner Umgebung effizient aufbauen und darauf zugreifen?
- Wie kann ein humanoider Roboter sich in solch einer Umgebungsrepräsentation aufgrund von verrauschten und ungenauen Sensordaten lokalisieren?
- Wie kann ein zweibeiniger Roboter schwierige Hindernisse wie zum Beispiel Treppen bewältigen, ohne dass er fällt?
- Wie kann ein zweibeiniger Roboter ein Navigationsziel effizient erreichen und dabei Hindernisse vermeiden oder sie übersteigen?
- Wie können schwierige Manipulationsaufgaben geplant werden, für die ein humanoider Roboter seinen ganzen Körper bewegen muss?
- Wie kann ein mobiler Roboter ein Navigationsziel effizient und kollisionsfrei erreichen, während er Objekte durch eine Umgebung mit vielen Hindernissen transportiert?

Zunächst wird als Grundlage eine effiziente 3D-Umgebungsrepräsentation vorgestellt, welche auf Octrees basiert und die um probabilistische Methoden zum Kartenaufbau sowie zum effizienten Zugriff ergänzt wird. Aufbauend darauf wird ein Ansatz zur probabilistischen Lokalisierung der dreidimensionalen Position und Orientierung eines humanoiden Roboters entwickelt. Das Verfahren verwendet dazu Entfernungsdaten von einem kleinen 2D Lasersensor oder einer handelsüblichen Tiefenkamera, gemessene Winkel von einem Lagesensor (IMU) und die Gelenkwinkel des Roboters. Das Ergebnis ist eine genaue Schätzung der Roboterpose während der Navigation. Jedoch stellt kompliziertes

Terrain wie zum Beispiel Treppenstufen eine Sturzgefahr dar und erfordert eine noch genauere Lokalisierung. Deshalb steht auch das Treppensteigen im Fokus dieser Arbeit. Nachdem der Roboter eine Treppe mit seinen Sensoren erkannt und ein Modell erstellt hat, lernt er die Treppensteigbewegung mittels kinesthetischer Demonstration durch einen Menschen. Um die Lokalisierungsgenauigkeit während des Treppensteigens zu verbessern, integriert der Roboter Messungen von extrahierten Kanten aus Kamerabildern zu den zuvor verwendeten Entfernungsdaten. Der verbesserte Lokalisierungsansatz kombiniert alle Sensordaten in einem erweiterten Partikel-Filter und führt zu einer hochgradig genauen Posenschätzung. Dies ermöglicht es dem Nao-Roboter, eine Treppe mit zehn Stufen und einem gewendelten Teil zu bewältigen.

Als Nächstes wird die Navigationsplanung für zweibeinige Roboter betrachtet, insbesondere die Planung von Fußschritten in einem Umgebungsmodell mit ebenen Hindernissen auf dem Boden. Sein Navigationsziel erreicht der Roboter, indem er die geplante Abfolge von Fußschritten mittels seines Laufverhaltens ausführt. Es werden verschiedene Suchverfahren vorgestellt, welche in kurzer Zeit Pläne mit garantiert begrenzter Suboptimalität erzeugen und diese verbessern, sofern noch Planungszeit zur Verfügung steht. Die betrachteten Algorithmen beinhalten ARA*, R* sowie Anytime D*, welcher inkrementelle und effiziente Neuplanung während der Ausführung ermöglicht. Um längere Fußschrittpfade zur Navigation zu planen, wird ein Verfahren vorgestellt, welches den Detailgrad der Planung anpasst. So wird in weitläufigen, offenen Gebieten ein schneller 2D-Pfadplaner verwendet und in der Nähe von Hindernissen ein detaillierter Fußschrittplaner.

Darüber hinaus werden Bewegungen für den gesamten Körper eines humanoiden Roboters geplant, was notwendig ist um zum Beispiel Objekte aufzuheben oder Türen zu öffnen und sich somit zwischen mehreren Räumen fortzubewegen. Aufgrund der vielen Freiheitsgrade die berücksichtigt werden müssen, wird der randomisierte Planungsalgorithmus RRT-Connect verwendet. Während der Planung müssen eine Reihe von Bedingungen berücksichtigt werden: Der Roboter darf nicht umfallen, seine Gelenkwinkel müssen im gültigen Bereich bleiben, Kollisionen mit sich selbst oder der Umgebung müssen vermieden werden und der Endeffektor muss einer bestimmten Trajektorie folgen wenn zum Beispiel eine Türe oder Schublade geöffnet wird.

Zuletzt wird in dieser Arbeit die Navigation in dreidimensionalen Umgebungen mit Hindernissen untersucht, insbesondere im Hinblick auf die mobile Manipulation. Das entwickelte Navigationssystem für den Radroboter PR2 verwendet den Planungsalgorithmus ARA* und die zu Beginn vorgestellte 3D-Umgebungsrepräsentation. Durch eine Kombination der dreidimensionalen Umgebung mit mehreren 2D-Projektionen zur Kollisionsprüfung wird die Planungszeit verkürzt. Das Navigationssystem erlaubt es dem PR2, große Objekte mit beiden Armen durch enge Passagen in einem Raum mit Hindernissen zu transportieren.

Alle vorgestellten Verfahren wurden eingehend evaluiert, sowohl mit echten Robotern als auch in Simulation. Die vorliegende Arbeit liefert wichtige Beiträge, um humanoide Roboter mit der Fähigkeit zur autonomen Navigation auszustatten.

Acknowledgements

This thesis would not have been possible without the support and encouragement by many people. First and foremost, I would like to thank my advisor Maren Bennewitz for giving me the opportunity to work in an exciting area of research. With her experience, she guided my work scientifically and provided many helpful suggestions along the way. At the same time, she gave me the opportunity to explore and pursue new ideas into various directions. I would also like to thank Olivier Stasse for agreeing to review this thesis.

I would like to thank all previous and current members of the Humanoid Robots Lab for the great atmosphere and work environment, and especially Daniel Maier and Felix Burget for the time we shared in our office. At the same time, I thank Wolfram Burgard and all members of the AIS group for providing a great work environment, interesting discussions, and support.

My thanks also go to Sachin Chitta and Radu Rusu for giving me the opportunity of a research internship at Willow Garage. The work environment and colleagues there provided a fruitful atmosphere for new ideas and input.

For their contributions and our collaborations on joint projects and research, I thank my co-authors Felix Atmanspacher, Wolfram Burgard, Felix Burget, Sachin Chitta, Andrew Dornbush, Johannes Garimort, Attila Görög, Steffen Gutmann, E. Gil Jones, Maxim Likhachev, Christian Lutz, Daniel Maier, Stefan Oßwald, Mike Phillips, Cyrill Stachniss, Hauke Strasdat, and Kai M. Wurm.

For proof-reading earlier versions of this document and providing valuable feedback, I would like to thank Christoph Sprunk, Jürgen Hess, Felix Burget, Felix Endres, Daniel Maier, and Stefan Oßwald.

I also appreciate the support by Dagmar Sonntag, Susanne Bourjaillat, and Michael Keser on all technical and administrative matters.

Finally, my deepest gratitude goes to my friends, my family, and especially to my wife Sanne for their support and encouragement at all times over the past years.

The work on this thesis has been generously supported by the German Research Foundation (DFG) under contract number SFB/TR-8 “Spatial Cognition”. Their support is gratefully acknowledged.

"Robotics is not an exact art. Sometimes unusual things happen. . . . We're talking about a stochastic event here."

— Isaac Asimov and Robert Silverberg, *The Positronic Man*

Contents

Chapter 1

Introduction

In the last decades, the wide adoption of industrial robots and automation significantly increased the productivity of manufacturing processes. The current generation of industrial robots can perform specialized tasks with high accuracy, repeatability, and throughput. Future generations of robots can benefit from recent advances in computer vision and artificial intelligence while relying on increased processing power. They thus have the potential to become increasingly useful for assisting humans in more complex and general settings. In the domestic domain, service robots could ease daily chores and provide elderly care in an aging society, as illustrated in Figure 1.1. But service robots could also save lives in the future by replacing human workers in hazardous environments or providing disaster relief after major accidents, as shown in Figure 1.2.

Common to these settings is that the environments are highly complex and unstructured, while being mostly designed for humans. Doors, staircases, or ladders can be easily overcome by humans, but they pose serious obstacles to wheeled robotic platforms. A robot with a body plan similar to a human, on the other hand, promises a better adaptability to these environments. The main features of this human-like or anthropomorphic body layout are two arms for manipulating the environment and two legs which enable a human-like, bipedal locomotion. This allows a bipedal robot to generate a continuous motion for its upper body with a discrete placement of foot contacts in the environment. In this way, the robot can overcome discontinuous terrain by stepping over or onto objects.

However, the complex kinematic structure of humanoid robots also has drawbacks. The high number of degrees of freedom make efficient planning and control a challenge. The robot has to actively balance to avoid falling. Inaccurate motion execution may lead to foot slippage or missteps and thus to a noisy ego-motion estimate. Furthermore, limited payload capabilities impose weight constraints on the sensors usable by humanoids. Lightweight sensors, however, are typically noisy.

The research community has acknowledged the prospects but also the challenges in a growing trend towards humanoid robotics. This was in part fueled by the recent announcement of the DARPA Robotics Challenge in which a robot has to navigate semi-autonomously in a collapsed factory or power plant building and perform various tasks

Figure 1.1.: The Honda ASIMO demonstrates a vision for household assistance (Source: Honda).

for disaster relief. Since the environment and tools are engineered for human use, most of the currently participating teams indeed rely on a humanoid robot.

Autonomous navigation is a core capability of any service robot that performs high-level tasks involving mobility. Before a robot can fulfill a task, it needs to know where it is in the environment, what the environment looks like, and how it can successfully reach its destination. Despite this necessity and the unique challenges inherent to humanoid robots, there has been only little research in the past concerning the navigation of humanoid robots. The main contribution of this thesis is to provide humanoid robots with navigation capabilities for complex indoor environments. We hereby cope with challenges in the areas of 3D environment representations, localization, perception, motion planning, and mobile manipulation.

As an example illustrating the challenges and required navigation capabilities, consider a humanoid service robot as home assistant that is to help a user with tedious everyday tasks. As a first step, the robot needs to build a three-dimensional representation of its surroundings, which it can access later to plan paths or determine its own location. Since this map is an integral part of the robot's navigation system, queries and updates of the map need to be as efficient as possible, and the map should be efficient to store in memory. Let us assume that the robot is instructed with a delivery task in the home, such as fetching laundry. To fulfill this task, it first needs to determine its own position and orientation in the environment representation by means of a localization system. From its own pose estimate, it can then plan a path to the goal. In between the start and goal, there might be all kinds of obstacles such as small objects on the floor that the humanoid can step over, but also larger ones such as furniture that should be avoided. To successfully reach the goal, the humanoid needs to plan a sequence of safe stepping motions. While walking towards the goal, however, inaccuracies in its motion execution may lead to a deviation from the original plan. To safely reach the goal nevertheless, the humanoid constantly needs to update its pose estimate based on sensor data and adjust its plan accordingly.

In our example, the laundry that the robot has to reach could be in a different room on a different level of a multi-story building. The humanoid might therefore encounter

Figure 1.2.: *Left:* NASA employs the *Robonaut 2* humanoid alongside astronauts on the International Space Station, where instruments and tools are designed for humans (Source: NASA). *Right:* Artist's concept of humanoid robots in the DARPA Robotics Challenge, operating in a degraded human-engineered environment (Source: DARPA).

stairs in its path, which pose a serious fall hazard. To successfully pass them, it should use careful stepping motions and localize itself with high accuracy. Even small errors in the robot's pose estimate could result in missing or bumping into a step and, as a result, falling down. After safely mastering the stairs, the robot may encounter a closed door that it has to open for passing. To open the door, the humanoid must plan a motion for all of its joints. During this motion it must maintain contact with the door handle and avoid losing its balance. Finally, the robot reaches the navigation goal and can bring the laundry back to the user.

To successfully accomplish high-level tasks such as the one outlined above, a humanoid robot needs a number of navigation capabilities, which we develop in the presented work. In summary, our work aims to answer the following questions:

- How can a robot efficiently build and access a three-dimensional environment representation?
- How can a humanoid robot localize itself accurately within such an environment representation despite noisy sensor data?
- How can we enable a biped robot to reliably overcome challenging terrain such as stairs without falling?
- How can a biped robot efficiently reach a navigation goal, thereby stepping around or over obstacles?
- How can we plan challenging manipulation tasks that involve the whole body of a humanoid robot?
- How can a mobile robot reach a navigation goal efficiently and collision-free while carrying objects through cluttered environments?

This thesis is structured as follows. We first introduce an efficient 3D environment representation based on octrees in Chapter 2, along with probabilistic techniques to build three-dimensional maps from sensor data and to access the data in the map. The presented framework is called *OctoMap* and formed the basis of a variety of robotics applications since its release as open-source software.

Based on this environment representation, we then present a probabilistic localization approach to estimate the full 6D pose of a walking humanoid with onboard range and proprioceptive sensors in Chapter 3. As range sensors, we compare a small 2D laser range finder and a consumer-level depth camera with different sensor models. We present a method to calibrate the motion model in order to improve the localization performance. Our approach results in accurate pose estimates both globally and for tracking while walking, even with the low-cost Nao humanoid platform. However, complex terrain such as staircases pose an additional challenge and fall hazards to the robot, and thus require more accurate localization. In Chapter 4, we present our solutions for the task of climbing stairs. After perceiving staircases in range data, we demonstrate whole-body stair climbing motions to the robot by means of kinesthetic teaching. To improve the localization accuracy while climbing stairs, we additionally fuse vision data from the robot's onboard camera with the previously used range data. All measurements are integrated in an extended particle filter using an improved proposal distribution. The presented methods significantly improve the localization accuracy, thus enabling the Nao humanoid to reliably climb up a complicated staircase consisting of ten steps and a winding part.

Next, we discuss navigation planning for humanoid robots in Chapter 5. In particular, we consider footstep planning in a given environment representation containing planar obstacles on the floor. From a planned sequence of collision-free footsteps, a humanoid can then generate walking motions to reach its navigation goal. We present and adapt different anytime search-based planners, which can return paths with provable bounds on suboptimality in a short planning time and improve them if there is more time available. We compare the performance between the ARA* and R* planners with different heuristics for footstep planning, and introduce the Anytime D* algorithm for incremental re-planning during navigation. To efficiently plan long paths for navigation, we develop an adaptive level-of-detail approach that uses fast 2D planning in open spaces and detailed footstep planning close to obstacles. Combined with our localization approach, our planning methods allow a Nao humanoid to execute footsteps through an obstacle course.

In Chapter 6, we consider planning for the whole body of a humanoid robot, which is necessary to pick up objects, but also to open doors when moving in between rooms. Due to the high numbers of degrees of freedom that have to be considered, we here use a randomized planning approach based on RRT-Connect. We account for a number of constraints inherent to the task: the humanoid needs to maintain its balance, respect its joint limits, avoid collisions with itself and the environment, and the end-effector needs to follow a certain trajectory when manipulating an articulated object such as a door or a drawer.

Afterwards, Chapter 7 presents an efficient approach for navigation in three-dimensional cluttered environments that is particularly suited for mobile manipulation. Our approach uses the anytime search ARA* and our efficient 3D environment representation to plan for the wheeled PR2 platform. Efficient collision checks using a combination of the 3D representation and multiple 2D projections hereby speed up the planning time. Our integrated navigation system enables the PR2 to carry large objects with two arms through demanding passageways in a cluttered room.

Finally, Chapter 8 summarizes the results of this thesis and discusses open areas of research for future work.

We thoroughly evaluated all of the developed approaches. For most of the evaluations, we used real data of the small-scale, affordable Nao humanoid. The robot is described in detail with different sensor configurations in Appendix A.

1.1. Key Contributions

This thesis describes several scientific contributions to the field of humanoid robot navigation. In these contributions, we cope with challenges in the areas of 3D environment representations, localization, perception, motion planning, and mobile manipulation. In summary, our key contributions are:

- an efficient, volumetric 3D environment model that is well suited for localization and collision checking during navigation (Chapter 2),
- a Monte Carlo localization approach in this environment model that accurately estimates the 6D pose of a walking humanoid based on range measurements and proprioception (Chapter 3),
- methods to detect and traverse challenging terrain such as winding staircases, including an improvement to our localization approach that combines range and vision data in an improved proposals particle filter (Chapter 4),
- efficient anytime footstep planning for biped navigation (Chapter 5),
- whole-body motion planning for manipulating doors and drawers, hereby considering a number of constraints such as keeping balance and avoiding collisions (Chapter 6), and
- anytime planning with 3D collision checks for navigation in the context of mobile manipulation (Chapter 7).

1.2. Publications

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

Journal Articles

- A. Hornung, S. Oßwald, D. Maier, and M. Bennewitz. Monte Carlo localization for humanoid robot navigation in complex indoor environments. *International Journal of Humanoid Robotics*, 2014. To appear
- A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34:189–206, 2013. doi: 10.1007/s10514-012-9321-0

Conference Proceedings

- F. Burget, A. Hornung, and M. Bennewitz. Whole-body motion planning for manipulation of articulated objects. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2013
- A. Hornung, A. Dornbush, M. Likhachev, and M. Bennewitz. Anytime search-based footstep planning with suboptimality bounds. In *Proc. of the IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS)*, 2012
- 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 and Systems (IROS)*, 2012
- A. Hornung and M. Bennewitz. Adaptive level-of-detail planning for efficient humanoid navigation. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012
- A. Hornung, M. Phillips, E. G. Jones, M. Bennewitz, M. Likhachev, and S. Chitta. Navigation in three-dimensional cluttered environments for mobile manipulation. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012
- S. Oßwald, J.-S. 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
- 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 and Systems (IROS)*, 2011

- J. Garimort, A. Hornung, and M. Bennewitz. Humanoid navigation with dynamic footstep plans. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011

- A. Hornung, K. M. Wurm, and M. Bennewitz. Humanoid robot localization in complex indoor environments. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010

Workshop Proceedings

- A. Hornung, D. Maier, and M. Bennewitz. Search-based footstep planning. In *Proc. of the ICRA Workshop on Progress and Open Problems in Motion Planning and Navigation for Humanoids*, 2013

- A. Hornung, M. Phillips, E. G. Jones, M. Bennewitz, M. Likhachev, and S. Chitta. Navigation in three-dimensional cluttered environments for mobile manipulation. In *Proceedings of the Workshop on Robots in Clutter: Manipulation, Perception and Navigation in Human Environments at Robotics: Science and Systems (RSS) Conference*, 2012

- A. Hornung, E. G. Jones, S. Chitta, M. Bennewitz, M. Phillips, and M. Likhachev. Towards navigation in three-dimensional cluttered environments. In *Proc. of the IROS 2011 PR2 Workshop: Results, Challenges and Lessons Learned in Advancing Robots with a Common Platform*, 2011

- A. Hornung and M. Bennewitz. Robust and adaptive navigation with humanoid robots. In *Proceedings of the Workshop on Motion Planning: From Theory to Practice at Robotics: Science and Systems (RSS) Conference*, 2012

- K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *Proc. of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, 2010

Publications not Covered by This Thesis

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

- D. Maier, A. Hornung, and M. Bennewitz. Real-time navigation in 3D environments based on depth camera data. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2012

- C. Lutz, F. Atmanspacher, A. Hornung, and M. Bennewitz. Nao walking down a ramp autonomously. In *Video Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012

- M. Bennewitz, D. Maier, A. Hornung, and C. Stachniss. Integrated perception and navigation in complex indoor environments. In *Proc. of the HUMANOIDS 2011 workshop on Humanoid service robot navigation in crowded and dynamic environments*, 2011

- A. Hornung, M. Bennewitz, and W. Burgard. Learning efficient vision-based navigation. In *Proc. of the Int. Conference on Indoor Positioning and Indoor Navigation (IPIN)*, 2010

- A. Hornung, M. Bennewitz, C. Stachniss, H. Strasdat, S. Oßwald, and W. Burgard. Learning adaptive navigation strategies for resource-constrained systems. In *Proc. of the 3rd Int. Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS)*, 2010

- A. Hornung, M. Bennewitz, and H. Strasdat. Efficient vision-based navigation – Learning about the influence of motion blur. *Autonomous Robots*, 29:137–149, 2010. doi: 10.1007/s10514-010-9190-3

- 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

1.3. Open Source Software

A substantial amount of the work developed in this thesis has been released as open-source implementations. This allows other researchers to build upon our results by integrating our frameworks or comparing against them.

- The OctoMap framework, available from http://octomap.github.io under the BSD license, was developed in collaboration with Kai M. Wurm and is currently maintained by the author of this thesis. It provides a self-contained C++ library to build memory-efficient 3D occupancy maps and is already used by a large number of researchers worldwide. Example use cases are obstacle avoidance, manipulation planning, and localization. Several extensions provide integration and mapping algorithms in ROS and are released in the packages *octomap_msgs*, *octomap_ros*, and *octomap_server* (see http://wiki.ros.org/octomap). We describe the approach in detail in Chapter 2.

- Based on OctoMap, we developed a 3D localization approach for humanoid robots and released it as a ROS package at http://wiki.ros.org/humanoid_localization. Stefan Oßwald and Daniel Maier contributed to the implementation. See Chapter 3 for a description of the approach.

- We developed a framework for footstep planning in collaboration with Johannes Garimort. The implementation builds upon the Search-Based Planning Library (Likhachev, 2010) and was released as ROS package at http://wiki.ros.org/footstep_planner. While originally applied to the Nao humanoid, it has been used by other researchers by now to plan for the HRP-2 and REEM-C humanoids as well as the ATLAS humanoid within the context of the DARPA Robotics Challenge. See Chapter 5 for a description of the approach.

- Our framework for navigation in three-dimensional cluttered environments was released as a ROS stack for the PR2 robot at http://wiki.ros.org/3d_navigation. The implementation was developed in collaboration with Mike Phillips and is based on the ROS navigation stack and SBPL. Our implementation was part of the basic functionality provided to the participants of the ICRA 2012 Mobile Manipulation Challenge[1], enabling the teams to perform demanding tasks such as setting or clearing a table. See Chapter 7 for a description of the approach.

- We developed and released an integration of the Nao humanoid in ROS[2], and maintained the implementation over the last years. By now it has been used by a number of different research groups worldwide, with contributions by many other authors. The BSD-licensed source code is available at https://github.com/ahornung/nao_robot and https://github.com/ahornung/nao_extras with documentation at http://wiki.ros.org/nao.

1.4. Collaborations

Parts of this thesis are the result of collaborative work with other authors. The OctoMap framework, described in Chapter 2, was jointly developed with Kai M. Wurm. Navigation in three-dimensional cluttered environments with the PR2 was developed during a research internship at Willow Garage with Mike Phillips and the support of E. Gil Jones and Sachin Chitta. While working on this thesis, I supervised several student projects, Bacherlor's, and Master's theses. Some of these lead to joint research and publications afterwards. The footstep planning framework in Chapter 5 was originally developed within the Bachelor's thesis of Johannes Garimort. Detecting and climbing stairs, described in Chapter 4, was originally developed in a project by Stefan Oßwald and later extended in

[1] http://mobilemanipulationchallenge.org/?page_id=131 (retrieved November 5, 2013)
[2] ROS News, Dec. 2009: http://www.ros.org/news/2009/12/first-proper-release-of-freiburgs-nao-stack.html

his Master's thesis. Whole-body motion planning described in Chapter 6 was developed during the Master's thesis of Felix Burget.

1.5. Notation

We shall use the following notation throughout this work.

Symbol	Meaning
a, x, y, ...	scalar value
φ, θ, ψ	Euler angles roll, pitch, and yaw
$\mathbf{a}$, $\mathbf{x}$, $\mathbf{y}$, ...	vector, usually a column vector
$\mathbf{x} = (x, y, z)$	vector with the scalar values x, y,z
A, B, ...	matrix
$\mathbf{x}^\top$, $A^\top$	transpose of a vector or a matrix
$\|\mathbf{x}\|$	norm (length) of a vector
$\mathbf{x} \oplus \mathbf{y}$	concatenation of 6D rigid body transforms
$\mathcal{S} = \{s_1, s_2, \dots\}$	set
$\|\mathcal{S}\|$	cardinality (number of elements) of a set
$\langle \dots \rangle$	tuple, an ordered set
$x \leftarrow a$	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
$\mathcal{N}(\mu, \sigma)$	Gaussian probability distribution with mean μ and variance σ

Chapter 2

Memory-Efficient 3D Environment Representation

In this chapter we present OctoMap, an open-source framework to generate and store volumetric 3D environment models. Our mapping approach is based on octrees and uses probabilistic occupancy estimation. It explicitly represents not only occupied space, but also free and unknown areas. Furthermore, we propose an octree map compression method that keeps the 3D models compact. We present a series of experimental results carried out with real robots and on publicly available real-world datasets. The results demonstrate that our approach is able to update the representation efficiently and models the data consistently while reducing the memory usage. Our framework is available as an open-source C++ library and has already been successfully applied in several robotics projects beyond the area of humanoid robot navigation. The following chapters will build on this representation for localization and collision checking.

An environment representation, also called model or map, is an integral part of robots performing autonomous navigation tasks. Maps are used to determine the robot's pose while it is moving, to avoid obstacles by means of collision checks, and to determine traversable areas for path planning.

Humanoid robots usually cannot be assumed to move on a plane to which their sensors are parallel due to their swaying walking motion. Additionally, their capability to step over or onto objects needs to be taken into account when planning in environments containing staircases or obstacles on the floor. Thus, the 2D grid map representations which are popular for wheeled platforms (Moravec and Elfes, 1985) are usually not sufficient for navigation with humanoid robots. Accordingly, 2.5D models that also represent the height of objects above the ground level are often used in the humanoid robotics community (Chestnutt et al., 2009; Michel et al., 2007; Thompson et al., 2006). These environment representations are nearly as efficient as 2D grid maps, while they are based on strong assumptions about the environment. For example, they are suited for structured,

mostly planar environments. Compared to that, highly unstructured and cluttered environments in which a robot operates in 3D may require a fully volumetric, three-dimensional environment model.

Beyond the area of humanoid robotics, a 3D environment model is generally useful when planning for a robot's base or manipulator, e.g., in domestic or even disaster relief scenarios. Recently, there is also a growing interest in combining navigation and manipulation in the field of mobile manipulation. Although 3D maps are such an integral component of many robotic systems, there exist few readily available, reliable, and efficient implementations. The lack of such implementations leads to the re-creation of basic software components and, thus, can be seen as a bottleneck in robotics research. We therefore developed an open-source 3D mapping framework as a foundation for this thesis and to generally facilitate the development of robotic systems that require a three-dimensional geometric representation of the environment.

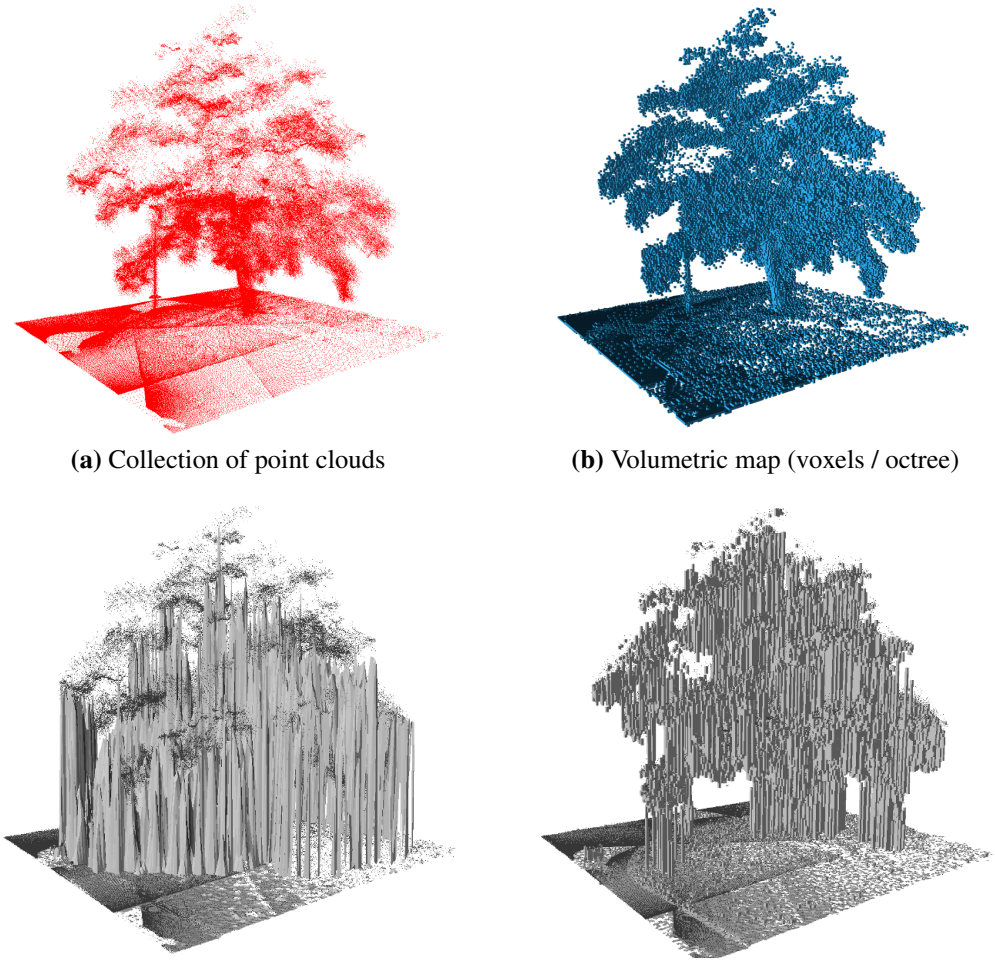
Most robotics applications require a probabilistic representation, the modeling of free and unmapped areas in addition to the occupied ones, and additionally efficiency with respect to runtime and memory usage. We will now discuss these requirements in detail.

- **Probabilistic representation:** Mobile robots create 3D maps from range measurements of their sensors. The sensors used in mobile robotics, however, are afflicted with noise and errors that can be in the order of centimeters. Additional uncertainties originate from systematic errors such as random reflections or dynamic obstacles that move while building a map. In order to create an accurate model of the environment, multiple uncertain measurements have to be fused in a probabilistic representation to estimate the true state of the environment. In addition, a probabilistic representation allows to integrate measurements from different sensors or even multiple robots.

- **Modeling of free and unmapped areas:** In order to plan safe collision-free paths, a mobile robot should avoid not only areas that are measured to be occupied, but also areas that it has collected no information about yet. These unknown areas can be either safe free space or obstacles. Therefore, the robot needs to distinguish free space from unknown space. By reasoning about the unknown space, the robot can specifically explore these unmapped areas when autonomously creating a map.

- **Efficiency:** Since the map is such a central component of a robotic system, it needs to be efficient with respect to access times and memory usage. The efficiency of the map directly affects the robot's performance in using it, e.g., for localization and navigation planning. For fully three-dimensional maps, the memory usage is often the major bottleneck. It is thus our priority to reduce the memory footprint of our environment representation. This enables a robot to keep even large and detailed maps in memory.

Several approaches have been proposed to model 3D environments in the past, illustrated in Figure 2.1. These include point clouds, elevation maps (Hebert et al., 1989),



(a) Collection of point clouds

(b) Volumetric map (voxels / octree)

(c) Elevation map

(d) Multi-level surface map

Figure 2.1.: Different 3D representations of a tree scanned with a laser range sensor. Our volumetric octree representation explicitly models free space but for clarity only occupied volumes are visualized in (b).

multi-level surface maps (Triebel et al., 2006), or volumetric (voxel) representations in the form of 3D occupancy grids. These approaches, however, do not meet our requirements on a suitable environment representation. Point clouds directly accumulate the measured endpoints and thus require large amounts of memory. Since only the endpoints are stored, they do not allow to differentiate between areas that are free and areas that have not been observed. Point clouds also do not allow to fuse multiple measurements in a probabilistic manner. Elevation maps and multi-level surface maps are memory-efficient environment representations, but also do not differentiate between free and unmapped areas. Most importantly, these approaches cannot represent arbitrary three-dimensional structures but assume a preference to planar surfaces in the environment. A 3D voxel grid represents the environment volumetrically and can encode occupied, free, and unknown areas, but requires large amounts of memory. Compared to that, we aim at representing the environment with the same expressivity as a voxel grid, but in a memory-efficient way.

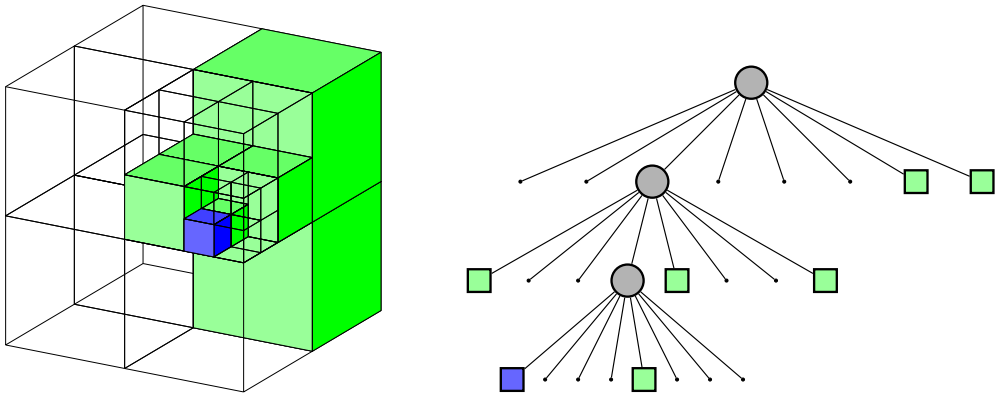
In this chapter we introduce our 3D environment representation called OctoMap, an integrated mapping framework based on octrees. We combine the advantages of previous approaches to 3D environment modeling in order to meet the requirements discussed above and obtain an environment model particularly suited for all kinds of humanoid navigation tasks. The framework presented in this chapter will form the foundation of this thesis, enabling 3D localization, path planning, navigation, and mobile manipulation in cluttered and multi-level environments.

A central property of our approach is that it allows for efficient and probabilistic updates of occupied and free space while reducing the memory consumption. Occupied space is obtained from the end points of a distance sensor such as a laser range finder, while free space corresponds to the observed area between the sensor and the end point. As a key contribution of our approach, we introduce a compression method that reduces the memory requirement by locally combining coherent map volumes, both in the mapped free areas and the occupied space. We implemented our approach and thoroughly evaluated it using various publicly available real-world robotics datasets of both indoor and outdoor environments.

Our open source implementation is freely available in form of a self-contained C++ library. It was released under the BSD-license and can be obtained from http://octomap.github.io. The library supports several platforms, such as Linux, Mac OS, and Windows. It has been integrated into the Robot Operating System (ROS) and can be used in other software frameworks in a straightforward way. Since its first introduction in 2010 (Wurm et al.), the OctoMap framework was constantly improved (Hornung et al., 2013b) and used in an increasing number of robotics research projects.

2.1. The OctoMap Framework

Our mapping approach uses a tree-based representation to offer maximum flexibility with regard to the mapped area and resolution. It performs a probabilistic occupancy estimation



(a) Volumetric model of an octree **(b)** Corresponding tree structure of the octree

Figure 2.2.: Example of an octree that stores free (green) and occupied (blue) cells.

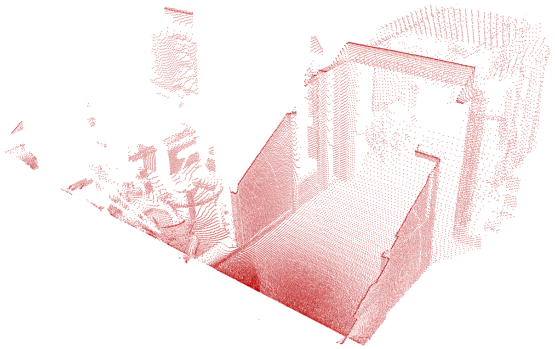
to ensure updatability and to cope with sensor noise. Furthermore, compression methods ensure the compactness of the resulting models.

2.1.1. Octrees

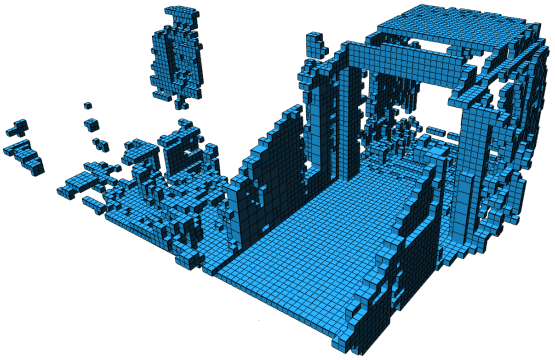
An octree is a hierarchical data structure for spatial subdivision in 3D (Meagher, 1982; Wilhelms and Van Gelder, 1992). Each node in an octree represents the space contained in a cubic volume, usually called a voxel. This volume is recursively subdivided into eight sub-volumes until a given minimum voxel size is reached, as illustrated in Figure 2.2. The minimum voxel size determines the resolution of the octree. Since an octree is a hierarchical data structure, the tree can be cut at any level to obtain a coarser subdivision if the inner nodes are maintained accordingly.

By storing a Boolean value in each volume of the octree, we can denote whether the volume is free or occupied. Any measurement of an obstacle in the world leads to the initialization of an occupied volume in the octree. By explicitly setting the free volumes as well, a robot can differentiate between areas that are known to be free and areas that have not been observed before. These free volumes are created in the area between the sensor and the measured end point, e.g., along a ray determined with ray casting. Areas that are not initialized implicitly model unknown space. An illustration of an octree containing free and occupied nodes from real laser sensor data can be seen in Figure 2.3. Using Boolean occupancy states or discrete labels allows for compact representations of the octree: If all children of a node have the same state (occupied or free) they can be pruned. This leads to a substantial reduction in the number of nodes that need to be maintained in the tree.

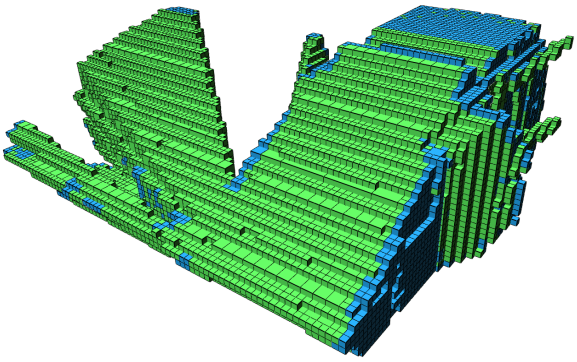
Measurements from robotic systems and real, physical sensor are typically affected by noise. Furthermore, environments in the real world are often not completely static. These cases require the integration of multiple measurements, for which a discrete occupancy



(a) Point cloud



(b) Occupied voxels in the octree



(c) Free and occupied voxels in the octree

Figure 2.3.: An octree map generated from example data. The point cloud sensor data (a) was recorded in a corridor with a tilting laser range finder. From this, occupied voxels (blue) are created in the octree as well as free ones (green) by clearing the space on a ray from the sensor origin to each end point. Lossless pruning results in leaf nodes of different sizes, mostly visible in the free areas in (c).

label is not sufficient. By estimating occupancy as a probabilistic variable in an occupancy grid (Moravec and Elfes, 1985), we can integrate multiple measurements to account for sensor noise and changes in the environment. In the occupancy grid setting, each node in the octree maintains a float value that denotes its occupancy probability. However, such a probabilistic model will reduce the effect of lossless compression by pruning, since nodes can have different occupancy probabilities between 0 and 1. In our approach, we combine the compactness of octrees that use discrete labels with the updatability and flexibility of probabilistic modeling, as we will discuss in Section 2.1.5.

In terms of data access complexity, octrees require an overhead compared to a fixed-size 3D grid due to the tree structure. A single random query on a tree data structure containing N nodes with a tree depth of d can be performed with a complexity of $\mathcal{O}(d) = \mathcal{O}(\log N)$. Traversing the complete tree in a depth-first manner requires a complexity of $\mathcal{O}(N)$. Note that, in practice, our octree is limited to a fixed maximum depth $d_{\max}$. This results in a random node lookup complexity of $\mathcal{O}(d_{\max})$ with $d_{\max}$ being constant. Therefore, for a fixed depth $d_{\max}$, the overhead compared to a corresponding 3D grid is constant. We used a maximum depth of 16 in all our experiments, which is sufficient to cover a cube with a volume of $(655.36\,\mathrm{m})^3$ at 1 cm resolution. We provide the exact timings for this setting in Section 2.3.4.

2.1.2. Probabilistic Sensor Fusion

In our approach, we integrate sensor readings in an occupancy grid as originally proposed by Moravec and Elfes (1985). We assume independence between all map cells, which are represented as leaf nodes n in the octree. Analogously to Moravec (1988), the probability $p(n \mid z_{1:t})$ of n to be occupied given the sensor measurements $z_{1:t}$ is estimated as

$$p(n \mid z_{1:t}) = \left(1 + \frac{1 - p(n \mid z_t)}{p(n \mid z_t)} \frac{1 - p(n \mid z_{1:t-1})}{p(n \mid z_{1:t-1})} \frac{p(n)}{1 - p(n)}\right)^{-1}. \tag{2.1}$$

In this formulation, $p(n)$ denotes the prior probability, $p(n \mid z_{1:t-1})$ the previous estimate, and $p(n \mid z_t)$ the probability of voxel n to be occupied given the measurement z_t. This probability is specific to the sensor that observed the measurement z_t. In this formulation, we estimate the map occupancy based on given sensor measurements, which is usually referred to as an inverse sensor model.

If we assume a uniform prior, which means that all nodes are initially equally probable to be free or occupied, we can set $p(n) = 0.5$. We additionally employ the log-odds notation as Moravec (1988) and can thus rewrite Eq. (2.1) as

$$\mathrm{L}(n \mid z_{1:t}) = \mathrm{L}(n \mid z_{1:t-1}) + \mathrm{L}(n \mid z_t), \tag{2.2}$$

where $L(n)$ is the log-odds value of n being occupied:

$$\mathrm{L}(n) = \log\left(\frac{p(n)}{1-p(n)}\right) \tag{2.3}$$

The straightforward formulation of the recursive update in Eq. (2.2) enables an efficient implementation since multiplications are replaced by additions. With a pre-computed sensor model, $\mathrm{L}(n \mid z_t)$ is a constant and the logarithms do not have to be computed during the update. Since log-odds can be computed from probabilities and vice versa, we directly store the occupancy log-odds for each voxel in our implementation. Note that for certain configurations of the sensor model that are symmetric, i.e., the update hitting a node has the same magnitude as the update for missing a node, this probability update has the same effect as counting hits and misses similar to Kelly et al. (2006).

When a robot uses a 3D map for navigation, it is usually interested in its maximum likelihood representation for path planning and localization. For this the two discrete states *free* and *occupied* of a voxel are sufficient. A voxel is considered to be occupied when its occupancy $p(n \mid z_{1:t})$ is above a threshold, and free when the occupancy is below. Changing the state of a voxel from free to occupied based on Eq. (2.2) may require as many observations as were previously integrated to define the current state, under the assumption that both measurements have the same log-odds magnitude. While this is suitable for static environments, it may be desirable for a mobile robot to react to changes more quickly, which requires quickly adapting the map. Yguel et al. (2007a) proposed a clamping update policy for this behavior. Instead of using Eq. (2.2) directly, the occupancy estimates must remain within upper and lower bounds $l_{\min}$ and $l_{\max}$ after the update:

$$\mathrm{L}(n \mid z_{1:t}) = \max(\min\left(\mathrm{L}\left(n \mid z_{1:t-1}\right) + \mathrm{L}\left(n \mid z_t\right), l_{\max}\right), l_{\min}). \tag{2.4}$$

This modified update rule limits the number of updates a voxel can receive towards the same state. By applying this clamping update policy in our approach, we can limit the map confidence and thus keep the map updatable. Another advantage is that pruning as described in Section 2.1.5 can compress the octree more effectively. The compression achieved with clamping is no longer completely lossless in terms of the full probabilities, since information close to zero and one is lost. In between the clamping thresholds, however, full probabilities are preserved.

The final update method of a single node in the octree including pruning and clamping is listed in Algorithm 1 for a log-odds update at coordinate $\mathbf{x}$. Initially, the recursive update function is called with the root node and depth 0 as arguments. The function then recursively proceeds to deeper tree levels, expanding previously pruned nodes (line 6) and creating new branches until the maximum tree level is reached. There, the leaf node's occupancy is updated with the clamping update from Eq. (2.4) (line 14). When the recursive call returns, previously traversed inner nodes will be updated from the bottom up (line 10) and pruned if needed (line 12). The updating of inner nodes enables multi-resolution queries, as we will discuss in Section 2.1.4.

Algorithm 1: Recursive update of a single node in the octree

Input: target coordinate **x**, log-odds occ_change, node n, depth d;
Initially, n is the root node and $d = 0$.
Output: The octree's occupancy at coordinate p is adjusted by occ_change

```
1  Function updateNode(x, occ_change, n, depth)
2      k ← branch index of child towards x at depth d
3      if d < max_tree_depth then
4          if k-th child of n exists then
5              if n is a pruned node then
6                  n.expand()    // create children, set their occupancy to occupancy of n
7              else
8                  create k-th child of n                    // initializes occupancy to prior
9          updateNode(x, occ_change, n.getChild(k), d + 1)              // recursive call
10         n.occupancy ← max_{i=1:8}(n.getChild(i))                 // update of inner nodes
11         if all children of n are identical and have no own children then
12             delete children of n                                              // pruning
13     else                                                  // leaf node at lowest level reached
           // log-odds update with clamping:
14         n.occupancy ← max(min(n.occupancy+occ_change), l_min), l_max)
```

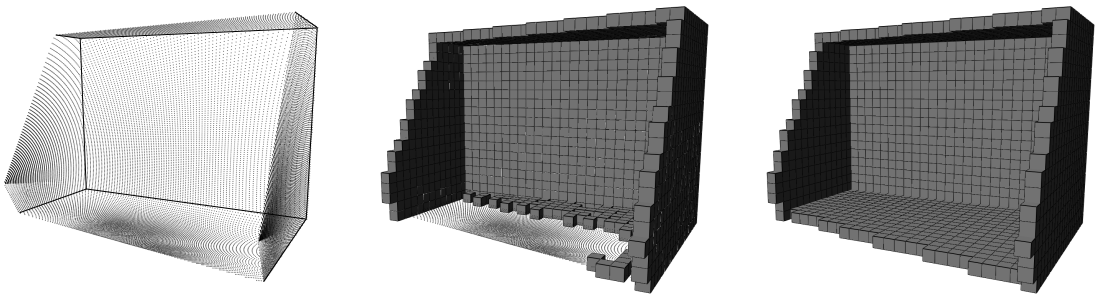


Figure 2.4.: Integration of a simulated noise-free 3D range measurement (left) into our 3D map data structure. Discretization effects of sensor sweeps at shallow angles lead to an undesired clearing of occupied volumes on the floor (center). By updating each volume at most once and preferring occupied over free updates during a sensor sweep, the map correctly represents the environment (right). For clarity, only occupied volumes are shown.

2.1.3. Sensor Model for Laser Range Data

OctoMap can be used with any kind of distance sensor, as long as an inverse sensor model is available. Since our real-world datasets were mostly acquired with laser range finders, we employ a beam-based inverse sensor model. We hereby assume that measured endpoints correspond to reflections from obstacle surfaces and that there is only free space between the sensor origin and a measured endpoint. By means of 3D ray casting from the sensor origin to the endpoint, we first determine all cells affected by the update and then apply the log-odds update described in Eq. (2.4) with

$$\mathrm{L}(n \mid z_t) = \begin{cases} l_{\mathrm{occ}} & \text{if beam is reflected within volume} \\ l_{\mathrm{free}} & \text{if beam traversed volume.} \end{cases} \tag{2.5}$$

The constants l_{occ} and l_{free} are parameters of the sensor model that correspond to the probability of measuring a volume as occupied or free with the sensor. For ray casting we rely on the efficient algorithm of Amanatides and Woo (1987).

With a sweeping laser range finder or a 3D sensor close to a flat surface, discretization effects can lead to undesired results when integrating multiple scans. At shallow angles, multiple ray-casting operations can mark voxels as free along the ray that cancel out previous measurements of occupied voxels. As a result, the modeled surface misses information as demonstrated in a noise-free simulation in Figure 2.4. To solve this problem, we first group a collection of 2D scan lines in a sensor sweep from the same location into a single 3D point cloud in our mapping approach. Since range measurements usually indicate the reflection of the beam from an obstacle surface, we ensure that all voxels that contain measurement endpoints are updated as occupied. This means that whenever a voxel is updated as occupied according to Eq. (2.5), it is not updated as free in the same measurement update of the map. As can be seen in Figure 2.4 (right), this leads to a more accurate representation of the actual environment.

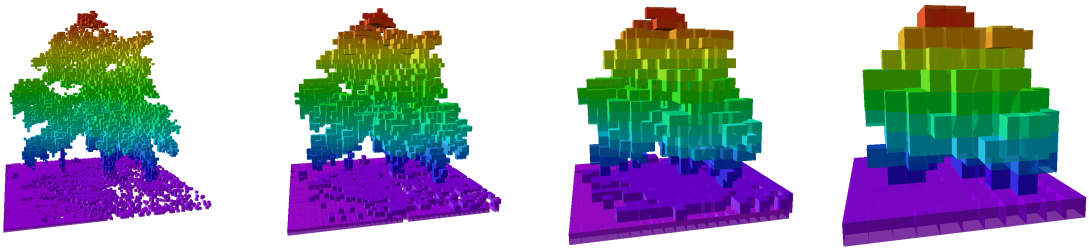


Figure 2.5.: Multi-resolution queries on an octree map by limiting the query depth. Occupied voxels are displayed in resolutions 0.2 m, 0.4 m, 0.8 m, and 1.6 m with height indicated by color.

2.1.4. Multi-Resolution Queries

The integration of measurement updates into our octree data structure corresponds to updating a 3D occupancy grid. Accordingly, sensor updates directly affect leaf nodes at the lowest level of the octree. At the lowest tree level, nodes have the size of the minimum octree grid resolution. By exploiting the hierarchical structure of the octree, we can enable multi-resolution queries. To this end, we use the inner nodes, which span the volume of their eight children. During a query, traversing the octree only up to a given depth limit instead of all the way to its leafs will yield a coarser subdivision of 3D space, and thus a faster query time. But this requires that an inner node also captures the occupancy probabilities of its children. There are several different strategies possible to aggregate the child occupancy into a parent node (Kraetzschmar et al., 2004). Depending on the application at hand, either the average occupancy

$$\bar{l}(n) = \frac{1}{8} \sum_{i=1}^{8} \mathrm{L}(n_i) \tag{2.6}$$

or the maximum occupancy

$$\hat{l}(n) = \max_i \mathrm{L}(n_i) \tag{2.7}$$

is best suited, where n is the parent node, n_i is the i-th child node, and $\mathrm{L}(n)$ returns the log-odds occupancy value of a node n.

The conservative strategy of the maximum child occupancy is well-suited for robot navigation, which is why we use it as default policy for updating inner nodes. For collision-free path planning based on a coarser octree subdivision, assuming that a volume is occupied if any smaller part is occupied leads to a safe behavior. An even more conservative strategy can be obtained by returning a positive occupancy also for unknown child nodes.

As an example use case, Figure 2.5 illustrates an octree that is queried for occupied voxels at different resolutions.

2.1.5. Octree Map Compression

In an octree with discrete labels of occupancy, we can reduce redundant information by pruning the tree: Whenever a node has eight children that have the same label, we can discard the children and instead store the information in the node itself. We can extend this compression technique to octrees that store an occupancy probability by pruning the children whenever they have the same probability values. However, even neighboring nodes that contain the same obstacle can have different probability values due to sensor noise, discretization effects, or observations from different viewpoints. This can inhibit an effective compression. A potential solution, as suggested by Fairfield et al. (2007), is to apply a threshold on the occupancy probability and then convert the nodes to their maximum likelihood state, that is, free or occupied. While this again allows an effective compression based on discrete labels, the individual probability estimates are lost and cannot be recovered anymore after pruning.

Instead, we achieve a more effective octree compression in our approach by relying on the clamping update policy in Eq. (2.4). This policy ensures that the log-odds occupancy values of a node remain within the bounds $l_{\min}$ and $l_{\max}$. Nodes that have been measured with high confidence will eventually reach the clamping thresholds, which enables an effective pruning since agreeing nodes have the same log-odds. We can remove the eight children of a node if they have the same occupancy state. In a static environment, the integration of multiple measurements will eventually converge all nodes to the upper or lower occupancy thresholds. For example, with our sensor parameterization, five agreeing measurements will create a completely free or occupied voxel. Should future measurements contradict the state of a pruned node, its children are regenerated as needed and updated accordingly. Applying this compression based on pruning only leads to a loss of information close to $p(n) = 0$ and $p(n) = 1$ while preserving the probabilities in between. In our experiments, combining octree pruning and clamping lead to a compression improvement of up to 44%.

In many robotic navigation tasks such as obstacle avoidance or localization, only the maximum likelihood map containing either free or occupied nodes is sufficient. In these cases, we can perform the lossy compression based on the occupancy threshold suggested by Fairfield et al. (2007). For this compression, all nodes are first converted to their clamped maximum likelihood probabilities and then pruned. This yields an even greater tree compression and less memory requirements.

2.2. Implementation Details

Our implementation is available in the form of the self-contained C++ library OctoMap. It contains the discussed mapping and compression techniques, as well as 3D visualization and evaluation tools. The library is released under the permissive BSD-license and can be obtained from http://octomap.github.io. The source code is thoroughly documented

and the library uses CMake to support Linux, Mac OS X, and Windows. Within the Robot Operating System (ROS), OctoMap is available as a pre-compiled package for the Ubuntu distribution.[1] Further ROS integration is available in the packages *octomap_ros* and *octomap_msgs*. OctoMap can be easily integrated into other frameworks with the help of CMake or pkg-config.

The library allows map updates from single measurement rays or complete point clouds to support a variety of sensors. The map can be accessed with single queries or batch queries via iterators analogously to a standard C++ container. This allows for a flexible access or further filtering, e.g., of leaf nodes or all octree nodes within a certain bounding box. For visibility checks and localization we provide ray casting functionality directly in the map, which we will build upon in Chapter 3.

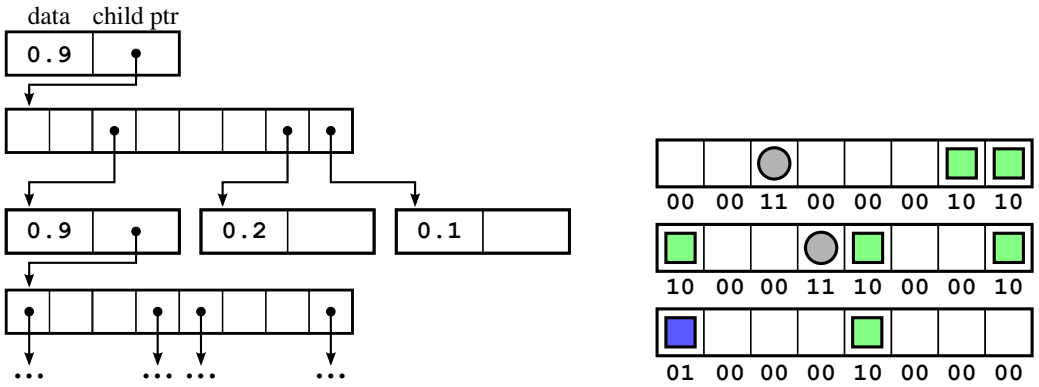
2.2.1. Memory-Efficient Node Implementation

In a straightforward octree implementation, each node in the tree stores in addition to the data payload the coordinate of its center location, its voxel size, and pointers to its children. This, however, can lead to a substantial memory overhead. Since the node location and its voxel size can be reconstructed while traversing the octree, we do not explicitly store this information in the nodes to reduce the memory overhead.

In general, octree nodes need to maintain an ordered list of their children. This can be directly achieved by using eight pointers per node. If sparse data are modeled, however, the memory usage of those pointers ($8 \times 4\,\text{byte} = 32\,\text{byte}$ on a 32 bit architecture) can result in a substantial memory overhead (Wilhelms and Van Gelder, 1992). We overcome that by using one child pointer per node that points to an array of eight pointers, as shown in Figure 2.6a. This array is only allocated if the node indeed has children and is not allocated for leaf nodes. Thus, any leaf node in the octree only stores the mapping data itself (e.g., the occupancy probability) and one (null) pointer. Inner nodes additionally store eight pointers to their children. In the robotics-related datasets used in our evaluation, 80% – 85% of the octree nodes are leafs. In our experiments, the above-mentioned implementation saved 60% – 65% of memory compared to allocating 8 pointers for each node.

To store a per-voxel occupancy probability, a single float value (usually 4 byte) is sufficient to represent the log-odds value. This results in a node size of 40 byte for inner nodes and 8 byte for leafs on a 32-bit architecture. Note that most compilers align member data in memory for runtime efficiency, that is, the data of a node are padded to be multiples of one word large (4 byte on a 32-bit architecture). 64-bit architectures can address large amounts of memory at the cost of pointers and words having twice the size. On such architectures, the memory size of inner nodes increases to 80 byte and the size of leaf nodes to 16 byte. Note that the actual size of the data structure (76 byte for inner nodes and

[1] See http://wiki.ros.org/octomap



(a) The first three nodes of the octree in memory

(b) Complete octree as maximum-likelihood bit stream

Figure 2.6.: Example octree from Figure 2.2 in memory connected by pointers and as compact serialized bit-stream. In memory, data is stored in one float denoting occupancy. Serialized, all maximum-likelihood occupancy information can be stored in only six bytes, using two bits for each of a node's eight child labels (00: unknown; 01: occupied; 10: free; 11: inner node with child next in the stream). Each row represents one node, with the upper row corresponding to the root node. The last row only contains leafs so no further nodes are added.

12 byte for leaf nodes) is again padded to multiples of the word size (8 byte on a 64-bit architecture) by most compilers.

In our approach, the octree is homogeneous by design, that is, all nodes have the same structure and store occupancy. While inner nodes could potentially save 8 byte by omitting occupancy information, maintaining it according to Eq. (2.6) or Eq. (2.7) enables multi-resolution queries, where tree traversal is stopped at a fixed depth.

Virtual inheritance between classes allows dynamic dispatch during run-time, at the cost of one extra pointer to the virtual function table (*vtable*) for each object instance. To minimize the memory footprint, we avoided this overhead in the octree node implementation. We apply direct inheritance and casts for the nodes, while using virtual inheritance only in the octree classes. This method results in an overhead of the size of only one pointer per octree map.

2.2.2. Map File Generation

In many scenarios, a robot needs to serialize the map, either to exchange it over network or to store it into a file. For example, a human or robot could build a map during an initial setup phase and the robot then uses this map for localization or path planning. Further examples include using hard disk space as temporary storage, or the exchange between multiple coordinating robots. All of these scenarios benefit from a compact serialized representation, which saves hard disk space and communication bandwidth.

When storing the complete map including all node probabilities and potential additional payload such as color and terrain information, we recursively serialize all data stored in

memory. We encode nodes by storing their data — occupancy with additional payload — and one bit for each of the eight children that specifies whether the child node exists. This is recursively repeated for each existing child.

Whenever a maximum likelihood representation is sufficient, we can create a more compact serialization. This representation contains only the information whether a volume is free or occupied instead of the actual occupancy probabilities and is usually sufficient for tasks like collision checking and localization. In addition, we preserve the information of unknown space since it can be of special interest in robotics as motivated before. In our compact maximum likelihood serialization, we thus encode nodes as either occupied, free, unknown, or as inner nodes. The nodes themselves are hereby not encoded, only the list of their eight children. A node is fully represented in the child encoding of its parent. This allows us to recursively encode the complete map in a compact bit stream. Starting at the root node, the labels of its eight children are added to the bit stream. Then, each child that is not a leaf is recursively added as well. Leaf nodes do not have to be added since they can be reconstructed from the parent's label during the decoding process. Figure 2.6b illustrates our maximum-likelihood bit-stream encoding.

In this maximum likelihood representation, each node occupies only 2 bits for each of its children, resulting in a total of 16 bits per node. This allows for a compact serialized representation. In our experiments, file sizes never exceeded 15 MB even for large outdoor environments at a fine resolution (see Section 2.3.3 for details).

Note that, analog to the octree representation in memory, the serialized stream does not contain any actual 3D coordinates. To reconstruct a map, only the location of the root node needs to be known. All other spatial relationships between the nodes are implicitly stored in the encoding.

2.3. Evaluation

We evaluated our mapping approach using several real-world datasets.[2] The experiments are designed to demonstrate that our approach is able to model the environments accurately and memory-efficient, while the data structure is efficient to build and access. We used the current implementation of OctoMap 1.5.6 [3] for our evaluation.

Unless noted otherwise, we used log-odds values of $l_{\text{occ}} = 0.85$ and $l_{\text{free}} = -0.4$, corresponding to probabilities of 0.7 and 0.4 for occupied and free measurements, respectively. We set the clamping thresholds to $l_{\text{min}} = -2$ and $l_{\text{max}} = 3.5$, corresponding to the probabilities of 0.12 and 0.97. We experimentally determined these values to work best for our use case of mapping mostly static environments with laser range finders, while still preserving map updatability for occasional changes. By adapting these changeable thresholds, a stronger compression can be achieved. As we will evaluate in Section 2.3.5, there is a trade-off between map confidence and compression.

[2] Available online at http://ais.informatik.uni-freiburg.de/projects/datasets/octomap

[3] https://github.com/OctoMap/octomap/archive/v1.5.6.tar.gz

Map dataset	Mapped area [m^3]	# scan end points
FR-079 corridor	$43.7 \times 18.2 \times 3.3$	$6 \cdot 10^6$
Freiburg campus	$292 \times 167 \times 28$	$20 \cdot 10^6$
New College (Epoch C)	$250 \times 161 \times 33$	$14 \cdot 10^6$
freiburg1_360 (RGBD)	$7.9 \times 7.3 \times 4.6$	$210 \cdot 10^6$
Nao environment	$5.9 \times 4 \times 2.1$	–

Table 2.1.: Overview of 3D map datasets

2.3.1. 3D Maps From Sensor Data

In this experiment, we demonstrate the ability of our approach to model real-world environments in a variety of different datasets. Table 2.1 gives an overview of the used datasets. Note that the free space was explicitly modeled in the experiments but is not shown in the figures for clarity.

The indoor dataset *FR-079 corridor* was recorded by a Pioneer2 AT robot equipped with a SICK LMS laser range finder on a pan-tilt unit. We reduced odometry errors by applying a 3D scan matching approach. The robot traversed the corridor of building 079 at the Freiburg campus three times, resulting in 66 3D scans with 6 million end points in total. When processing this dataset, we limited the maximum range of the laser beams to 10 m. This removed stray measurements outside of the building which were observed through windows. Figure 2.7 shows the resulting map.

For evaluating the performance in an outdoor setting, we used the *Freiburg campus* dataset.[4] It consists of 81 3D laser scans, each covering 360° from a pan-tilt unit. The complete dataset covers an area of 292 m × 167 m along a trajectory of 723 m. Additionally, we used laser range data of the *New College* dataset (Smith et al., 2009), Epoch C. This data also covers a large-scale outdoor environment. A wheeled robot captured the data with two fixed laser scanners sweeping at the left and right side. For this dataset, we used an optimized estimate of the robot's trajectory from a visual odometry approach (Sibley et al., 2009). The resulting outdoor maps are shown in Figure 2.8.

Furthermore, we integrated data of the *freiburg1_360* RGBD-dataset (Sturm et al., 2012) into our map representation with a total of 210 million end points from the Microsoft Kinect sensor. The final map, visualized in Figure 2.9, represents an office environment at a resolution of 2 cm. For this map, we augmented individual octree nodes to store color information in addition to occupancy. This creates immersive visualizations for the user and enables a color-based classification of the environment or appearance-based robot localization from virtual views, similar to Einhorn et al. (2011) and Mason et al. (2011).

Finally, we constructed a 3D map of an experimental environment for a small Nao humanoid. The environment consists of two different levels connected by a winding stair-

[4] Courtesy of B. Steder, available at http://ais.informatik.uni-freiburg.de/projects/datasets/fr360/

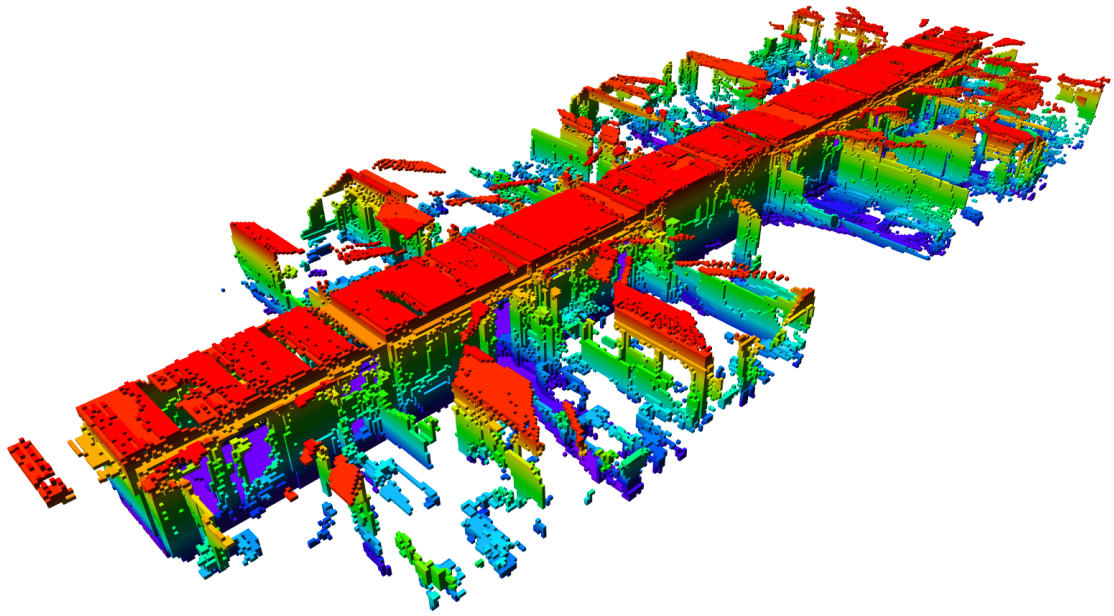


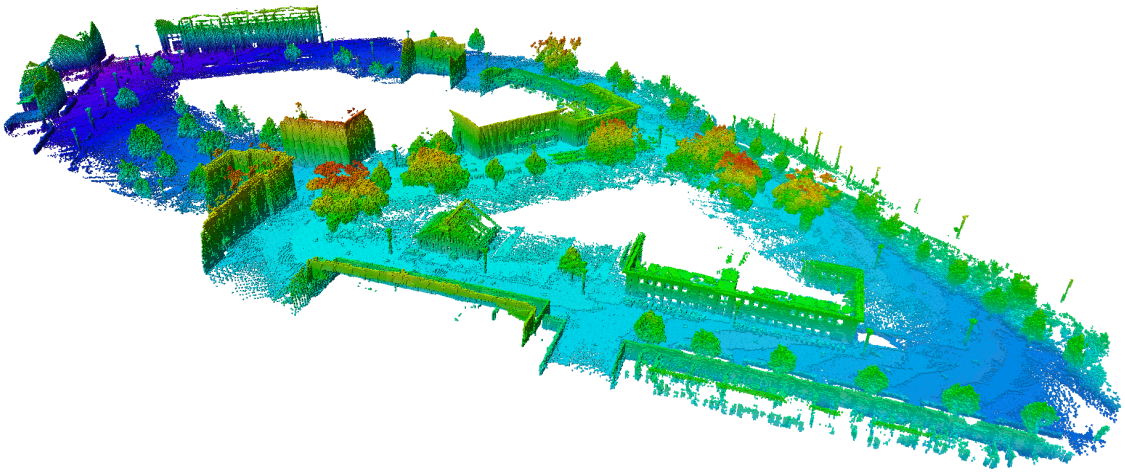
Figure 2.7.: Octree map of the FR-079 corridor dataset at 0.1 m resolution, as seen from the top. For clarity, only occupied volumes are shown with height visualized by a color coding. Size of the scene: 43.7 m × 18.2 m × 3.3 m.

case and a ramp. The resulting occupancy map is shown in Figure 2.10 and was converted from a manually designed CAD model. To accurately model the detailed structures, e.g. of the handrail, a fine resolution of 1 cm is required.

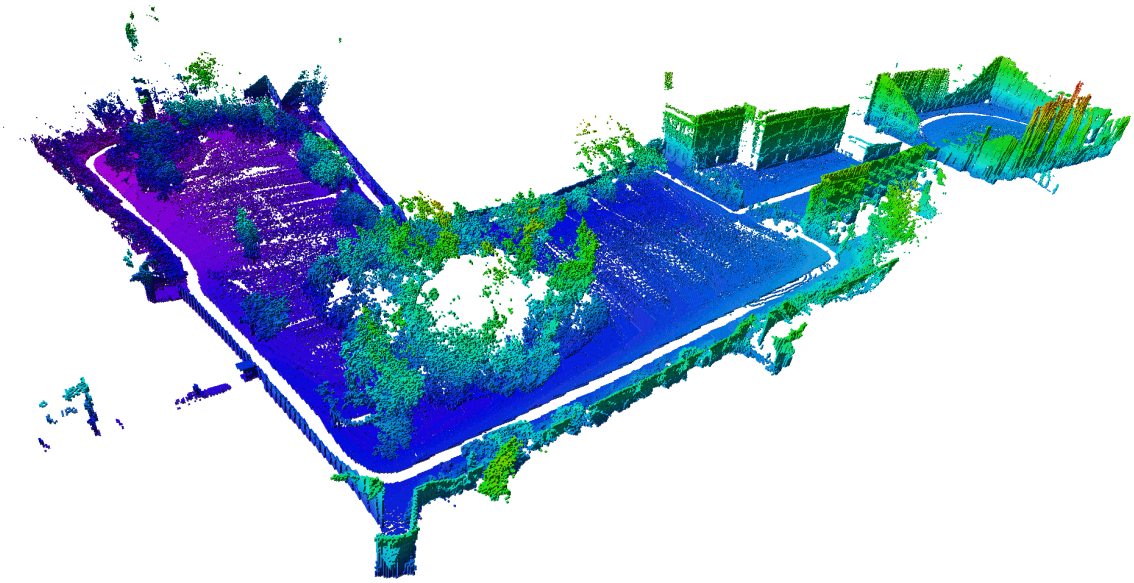
2.3.2. Map Accuracy

This experiment evaluates how accurately a 3D map represents the data that was used to build that map. Note that this particular evaluation is independent of the underlying octree structure since our mapping approach is able to model the same data as a 3D grid. We measure the accuracy as the percentage of correctly mapped cells in all 3D scans. A 3D map cell counts as correctly mapped, if it has the same maximum-likelihood state (free or occupied) in the map and the evaluated 3D scan. The scan is hereby treated as if it were inserted into the already-built map, i.e., endpoints must be occupied and all cells along a ray between the sensor and the endpoint must be free. As a second measure, we cross-validate the map by skipping each 5th scan when building the map, and using these skipped scans to evaluate the percentage of correctly mapped cells.

The results in Table 2.2 show that our mapping approach accurately represents the environment. The remaining error is most likely due to sensor noise, discretization effects, or a not completely perfect scan alignment. The cross-validation results only lose little accuracy, which demonstrates that the probabilistic sensor model yields realistic and predictive results.



(a) Freiburg campus dataset at 0.2 m resolution. Size of the scene: 292 m × 167 m × 28 m.



(b) New College dataset at 0.2 m resolution. Size of the scene: 250 m × 161 m × 33 m.

Figure 2.8.: Octree maps of real-world outdoor datasets. For clarity, only occupied volumes are shown with height visualized by a color coding.

Figure 2.9.: Detail of a volumetric indoor OctoMap containing color information. The complete map covers an area of 7.3 m × 7.9 m × 4.6 m at 2 cm resolution.

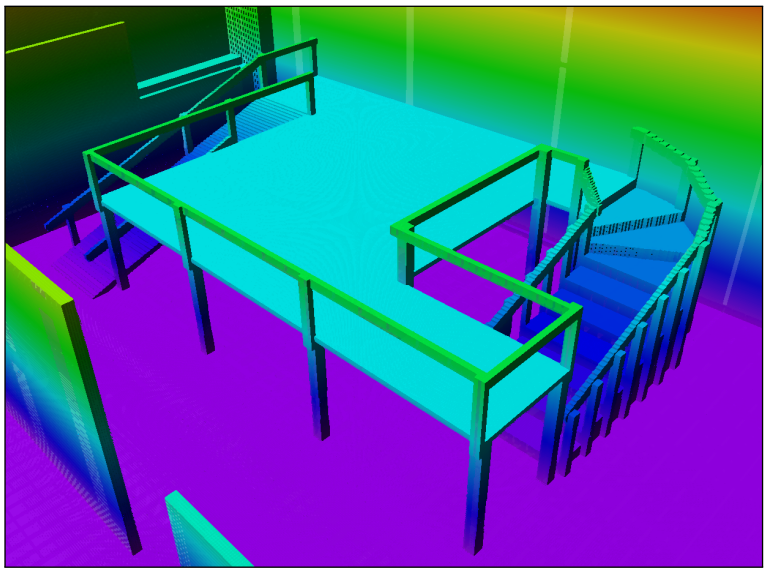


Figure 2.10.: Experimental indoor environment for a small Nao humanoid, containing two levels connected by a staircase and a ramp. This occupancy map was converted from a 3D CAD model. The complete map covers an area of 5.9 m × 4 m × 2.1 m at 1 cm resolution.

Map dataset (resolution)	Accuracy	Cross-validation
FR-079 corridor (5 cm)	97.27%	96.00%
Freiburg campus (10 cm)	97.89%	95.80%
New College (10 cm)	98.79%	98.46%

Table 2.2.: Map accuracy and cross-validation as percentage of correctly mapped cells between evaluated 3D scans and the built map. For the accuracy, we used all scans for map construction and evaluation. For cross-validation, we used 80% of all scans to build the map and the remaining 20% for evaluation.

2.3.3. Memory Consumption

We analyzed the memory usage of our representation for different datasets at different resolutions, and evaluated the effects of octree compression as well as the maximum-likelihood compression that converts each node a completely free or occupied state. For comparison, we also determined the amount of memory that would be required by an optimally aligned 3D grid of minimal size that is initialized linearly in memory. According to Section 2.2.1, the memory consumption of occupancy stored in an octree on a 32-bit architecture is given by

$$\text{mem}_{\text{tree}} = n_{\text{inner}} \times 40\,\text{byte} + n_{\text{leafs}} \times 8\,\text{byte}\,, \tag{2.8}$$

where n_{inner} is the number of inner nodes and n_{leafs} the number of leaf nodes. The size of a minimal 3D grid storing the same information (one float for the occupancy probability) is given by

$$\text{mem}_{\text{grid}} = \frac{x \times y \times z}{r^3}\, 4\,\text{byte}\,, \tag{2.9}$$

where x, y, z is the size of the map's minimal bounding box in each dimension and r is the map resolution.

We furthermore wrote each map to disk using the full probabilistic model and the compressed binary format described in Section 2.2.2, and evaluated the resulting file sizes.

Table 2.3 lists the memory usage of the datasets for different exemplary resolutions. As can be seen, we achieved high compression ratios particularly in the outdoor datasets. These datasets contain large volumes of coherent free and unknown space. Coherent free space will be merged by pruning, while unknown space does not use any memory in the octree. Note that a 3D grid of the outdoor datasets with a resolution of 10 cm would not even fit into the addressable main memory of a 32-bit machine. Our data structure is also able to model smaller indoor environments with fine resolutions and efficiently store them in memory. Only an uncompressed octree required more memory than an optimally aligned 3D grid, and only when most areas of the map have been observed (i.e., there are only few unknown areas). This effect is particularly evident in the Nao environment dataset, which was constructed from a CAD model.

	3D grid	Octree in memory [MB]			Octree file [MB]	
Map dataset (resolution)	[MB]	Uncompr.	Pruned	Max. likelih.	Full	Lossy
FR-079 corridor (5 cm)	78.88	73.55	41.62	24.72	15.76	0.67
FR-079 corridor (10 cm)	10.01	10.87	7.22	5.02	2.70	0.14
Freiburg campus (10 cm)	5162.90	1257.57	990.66	504.76	379.70	13.82
Freiburg campus (20 cm)	648.52	187.93	130.24	74.12	49.68	2.00
Freiburg campus (80 cm)	10.58	4.55	4.12	3.09	1.53	0.08
New College (10 cm)	5058.76	607.92	395.42	230.33	148.75	6.40
New College (20 cm)	633.64	91.33	50.57	35.95	18.65	0.99
New College (80 cm)	10.13	2.34	1.79	1.69	0.63	0.05
freiburg1_360 (2 cm)	252.99*	159.97*	45.52*	20.05	21.59*	0.52
freiburg1_360 (5 cm)	16.19*	11.24*	4.55*	2.52	2.11*	0.07
Nao environment (1 cm)	189.06	660.08	50.46	50.46	20.10	1.14

Table 2.3.: Memory consumption in megabyte (MB) of different octree compression types compared to full 3D occupancy maps (*3D grid*) on a 32-bit architecture. Octree compression in memory (*Pruned*) is achieved by merging identical children into the parent node. A more efficient but more lossy compression in memory is achieved by converting each node to its maximum-likelihood value (completely free or occupied) followed by pruning the complete tree. A maximum-likelihood tree containing only free and occupied nodes can then be serialized to a compact binary file format (*Lossy*).
(*): Voxels contain the full color information from the RGBD dataset.

Figure 2.11 shows the evolution of memory consumption over time. Memory usage grows when the robot explores new areas (scans 1–22 and 39–44 in FR-079 corridor, scans 1–50 and 65–81 in Freiburg campus). In between, the robot revisited previously mapped areas, which results in a constant memory usage or even a small reduction due to pruning.

As expected, memory usage increases exponentially with the tree resolution. This effect can be seen in Figure 2.12, where we used a logarithmic scaling in the plot.

Table 2.3 also lists the file sizes of the serialized binary maximum likelihood map (denoted as "Lossy") and the full probabilistic model ("Full"). Even the large outdoor datasets *Freiburg campus* and *New College* resulted in file sizes of less than 14 MB, which demonstrates the efficiency of our approach for storing maps and exchanging them between different processes or robots. The files can be compressed even more with standard file compression methods.

2.3.4. Run Time Performance

In the following experiments, we analyzed the time required to integrate and access data in our framework. All runtimes were evaluated on a single core of a standard desktop CPU (Intel Core i7-2600, 3.4 GHz) for various map datasets.

Map Generation

First, we analyzed the time required to generate maps by integrating range data. This time depends on the map resolution and the length of the beams that are integrated. We processed the *FR-079 corridor* and *Freiburg campus* datasets both with the full laser range (up to 50 m) and with a limited maximum range of 10 m for several resolutions. The average insert times for one beam are given in Figure 2.13.

In our experiments, 3D scans usually consisted of about 90 000 – 250 000 valid measurements. Typically, such a scan could be integrated into the map in less than a second. This demonstrates that our current implementation can cope even with the demanding data of RGBD-cameras that output up to 300 000 points at fast frame rates, albeit at shorter ranges.

With long measurement beams and large outdoor areas as in *Freiburg campus*, a speedup can be obtained by limiting the map update range. Indoors, however, where only few sensor beams reach far, there is no noticeable speedup by limiting the sensor range.

Map Queries

We evaluated the time to traverse all leaf nodes (free or occupied) in an existing map. The depth of a query can be limited during run time which in our data structure is equivalent to a map query in a coarser map. This allows for more efficient tree traversals in those cases when a coarser resolution is sufficient.

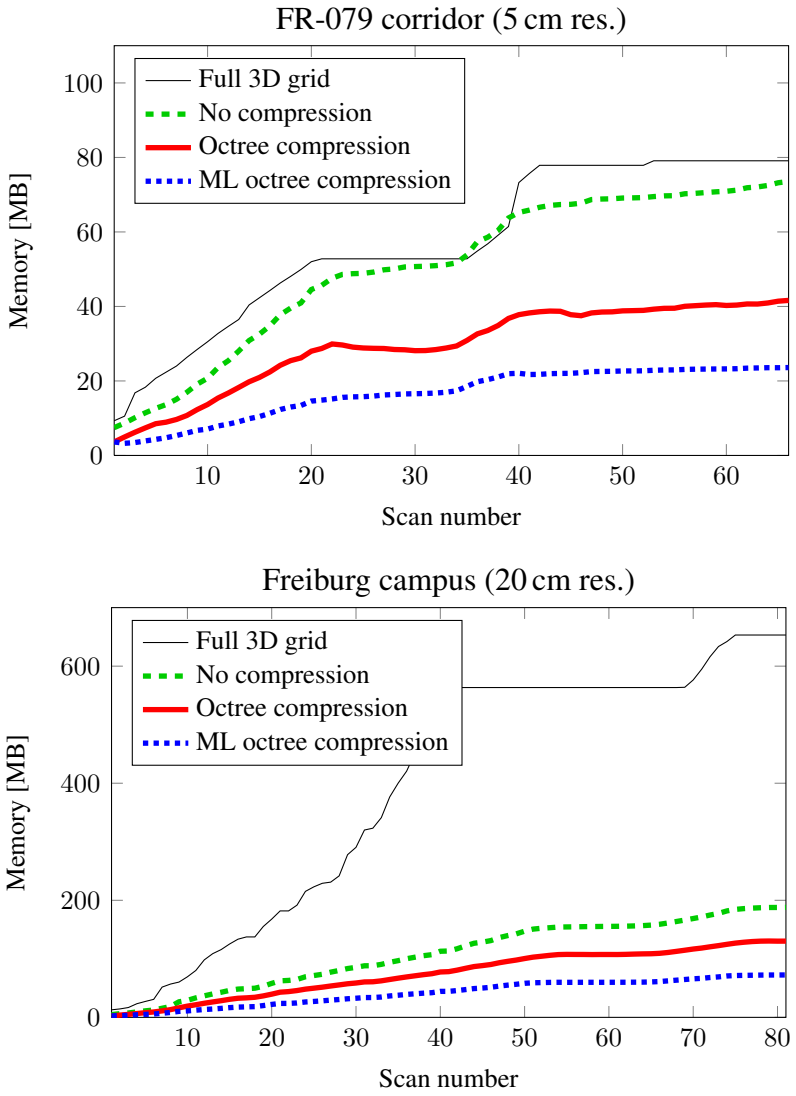


Figure 2.11.: Memory usage while mapping the two datasets FR-079 corridor and Freiburg campus.

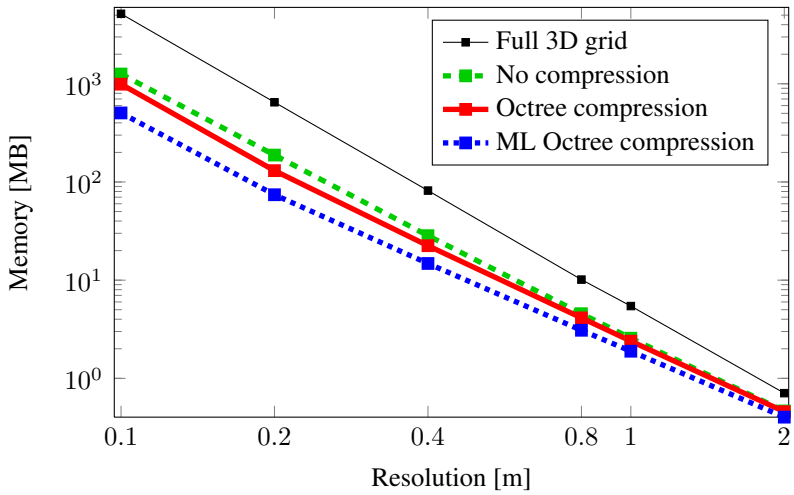


Figure 2.12.: Effect of resolution on memory usage of the Freiburg campus dataset. Note that a logarithmic scaling is used.

Figure 2.14 shows the time to traverse several maps to their maximum tree depth of 16, which corresponds to the full map resolution (depth cutoff=0). The plot furthermore gives the times to query all leaf nodes when the query depth is limited. Each increment in the depth cutoff doubles the edge length of the smallest voxels and speeds up the traversal by a factor of about two. It can be seen that map traversals are efficient: Even at full map resolution, the large map of the Freiburg campus containing 1 087 014 occupied and 3 377 882 free leaf nodes can be traversed within 51 ms.

2.3.5. Clamping Parameters

Finally, we analyzed the impact of the clamping thresholds on map accuracy and compression. Since these thresholds provide a lower and upper bound for the occupancy probability, information close to $p = 0$ and $p = 1$ is lost compared to the full map with no clamping. A clamped map represents an approximation of the full map, thus we use the Kullback-Leibler divergence (KLD) summed over the complete map as measure. Since occupancy is a binary random variable with the discrete states *free* and *occupied*, the KLD of a clamped map M_c from the full map M_f can be computed by summing over all map nodes n:

$$\mathrm{KLD}(M_f, M_c) = \sum_n \left(\ln\left(\frac{p(n)}{q(n)}\right) p(n) + \ln\left(\frac{1-p(n)}{1-q(n)}\right) (1-p(n)) \right), \tag{2.10}$$

where $p(n)$ is the occupancy probability of node n in M_f, and $q(n)$ in M_c.

The results for a series of occupancy ranges from $[0:1]$ (no clamping, lossless) to $[0.4:0.6]$ (strong clamping, high loss) and different maps can be seen in Figure 2.15. The values for our chosen default threshold $[0.12:0.97]$ are shown as thin horizontal lines,

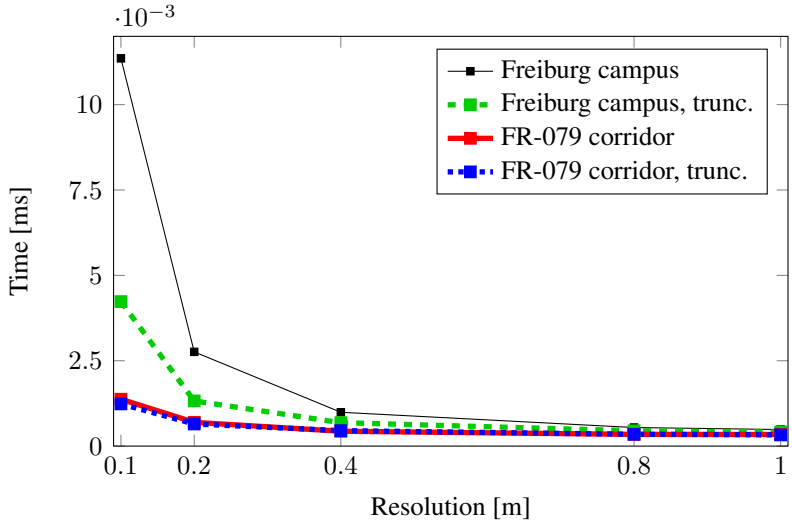


Figure 2.13.: Average time to update an octree map by inserting one data point for the datasets Freiburg campus and FR-079 corridor. The truncated versions insert rays up to a maximum sensor range of 10 m only.

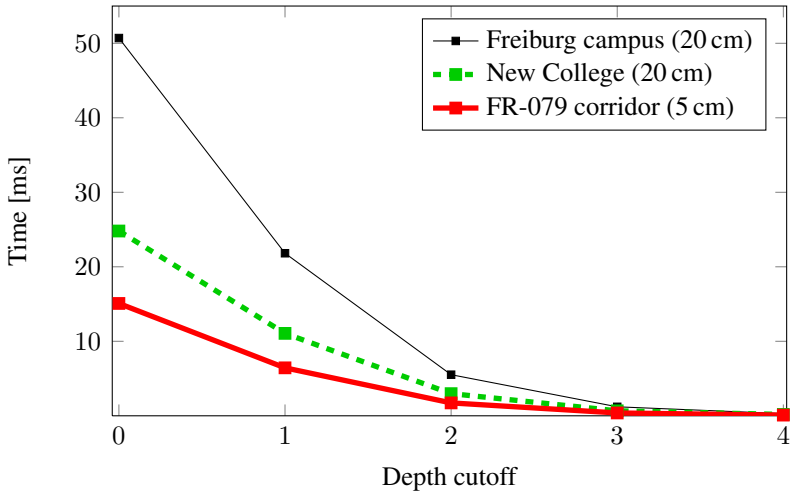


Figure 2.14.: Time to traverse all octree leaf nodes in different maps. By limiting the depth of the query (called depth cutoff) a coarser map is traversed.

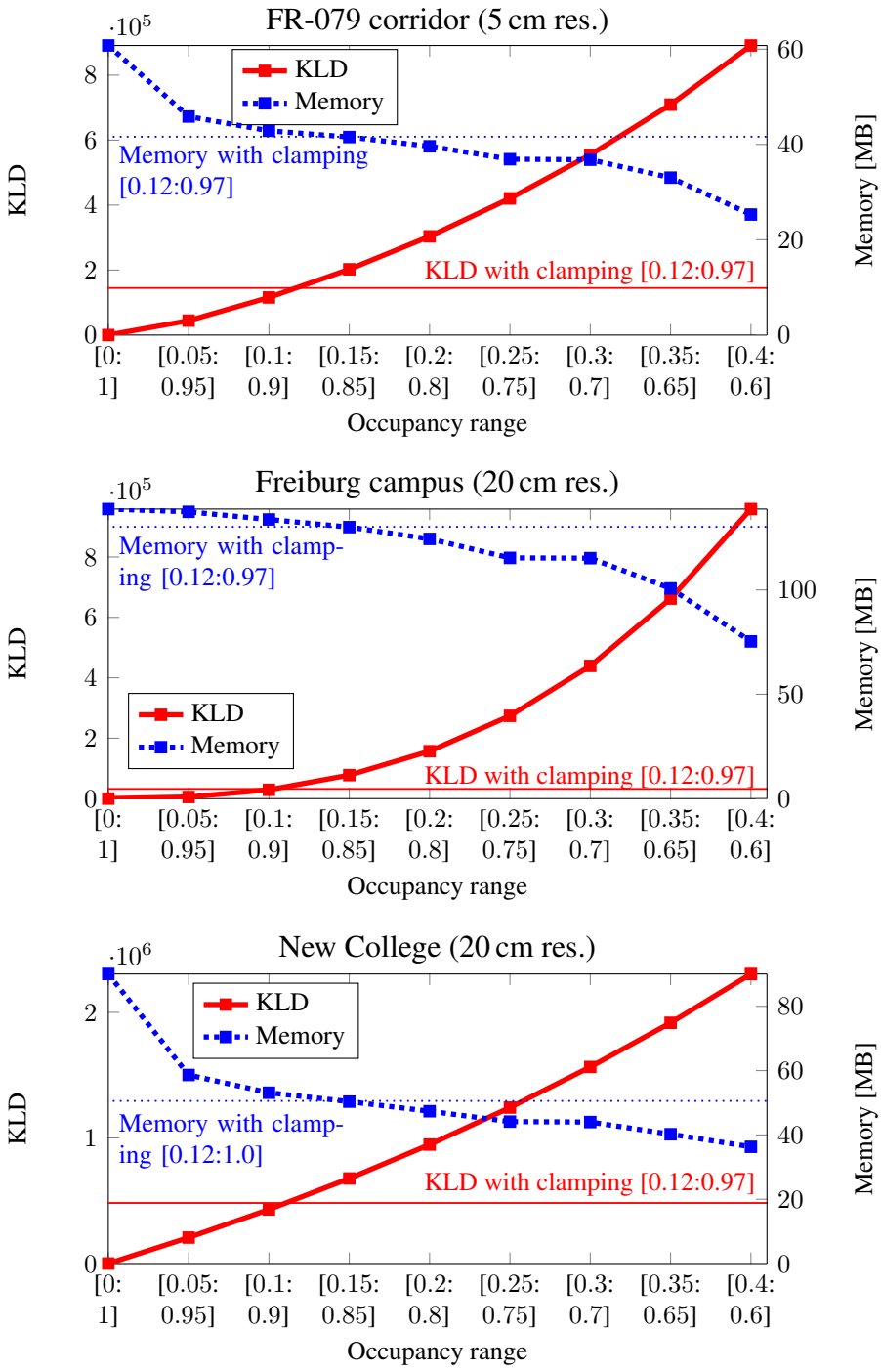


Figure 2.15.: Effect of different clamping ranges on map compression and accuracy in our three datasets. A higher clamping, resulting in a smaller occupancy range, increases the efficiency of the octree compression (memory consumption, dashed blue). The Kullback-Leibler divergence (KLD, red) measures the information loss between the unclamped map with full probabilities in [0:1] and a clamped representation. Our default clamping range [0.12:0.97] is shown for comparison by horizontal lines in blue (dashed) for memory consumption and red for the KLD.

dashed blue for the memory consumption and red for the KLD. This clamping range was chosen primarily to work best in the context of laser-based mapping and occasional changes in the environment, such as people moving through the scans or doors closing. As can be seen, a stronger compression can be achieved with higher clamping, at the cost of losing map confidence. In the most degenerated case, one sensor update can be enough to mark a voxel as completely free or occupied, losing any ability to filter noise with a probabilistic update. Note that, while clamping is beneficial for map compression, even with no clamping the losslessly compressed maps are smaller than a 3D grid (cf. Table 2.3).

2.4. Related Work

Since three-dimensional environment representations are required for many robotic tasks, they are a common area of research. In the last two decades, multiple different approaches have been presented.

By extending the idea of two-dimensional occupancy grids to 3D, Roth-Tabak and Jain (1989) and Moravec (1996) presented early works about 3D occupancy grids, which contain equally-sized cubic volumes or voxels. Such a 3D occupancy grid, however, requires large amounts of memory since the grid has to encompass all of the mapped area even when measurements in this area are only sparse. Furthermore, the complete 3D occupancy grid needs to be allocated beforehand or the map needs to be expanded with expensive copy operations when the robot encounters new areas.

Discretizing the environment into a grid can be avoided by directly aggregating the end points of 3D range measurements into point clouds. Several 3D SLAM systems use this approach (Cole and Newman, 2006; Nüchter et al., 2007). However, since only end points are captured, this does not allow the robot to reason about free and unknown space, or to cope with sensor noise and dynamic objects. Point cloud representations may thus require highly accurate sensors in a mostly static environment. Another disadvantage is that the memory consumption increases with each additional measurement and is not bounded by the mapped area.

By assuming certain properties about the environment, 2.5D maps can present a sufficient model. In an elevation map, a 2D grid stores the height of the ground above a reference level (Hebert et al., 1989). Hadsell et al. (2009) demonstrated an outdoor navigation system based on elevation maps. For robot navigation on a single underlying surface, elevation maps provide a sufficient model. All objects overhanging at higher levels than the robot, such as trees or bridges, can be safely ignored since they pose no obstacles. Extensions that allow multiple levels for each 2D cell (Pfaff et al., 2007; Triebel et al., 2006) or different obstacle classifications (Gutmann et al., 2008) relax the strict assumptions of elevation maps. Nevertheless, a general drawback of most 2.5D maps is that they do not represent the environment in a volumetric way but discretize it in the vertical dimension based on the robot's height. While this is sufficient for path planning and navigation

with a fixed robot shape, the map does not represent the actual environment accurately. For use cases such as localization or collision avoidance with manipulators, an accurate environment model is required instead.

To overcome this problem, a related approach was proposed by Ryde and Hu (2010), in which each cell of a 2D grid map contains a list of voxels that represent occupied volumes over the cell. Whereas their approach does not differentiate between free and unknown volumes, Dryanovski et al. (2010) store lists of both occupied and free voxels for each 2D cell in their Multi-Volume Occupancy Grid. In contrast to our approach, however, the map extent needs to be known beforehand, map updates are more computationally involved, and there is no multi-resolution capability. Another potential problem is that subsequent map updates cannot subdivide existing volumes, leading to an incorrect model of the environment. Douillard et al. (2010) present a combination of a coarse elevation map with high-resolution voxel maps. The elevation map is used for background structures, while the voxel maps are used to represent objects. In contrast to our work, this approach focuses on 3D segmentation of single measurements and does not integrate multiple measurements into an environment model.

In robotic mapping, octrees avoid one of the main shortcomings of fixed occupancy grid maps, which require that the extent of the environment is known beforehand. Instead, octrees initialize nodes only as needed as soon as the corresponding area was observed by a sensor. If inner nodes of a tree are updated as in our approach, the tree can also be used as a multi-resolution representation since it can be cut at any level to obtain a coarser subdivision. Originally, Meagher (1982) proposed to use octrees for mapping and early works modeled occupancy as a Boolean property (Wilhelms and Van Gelder, 1992). Probabilistic 3D occupancy mapping based on octrees was introduced by Payeur et al. (1997), with similar approaches used by Fournier et al. (2007) and Pathak et al. (2007). Compared to these approaches, our framework uses a bounded map confidence and we thoroughly compare different methods of map compression.

Fairfield et al. (2007) also used an octree-based 3D environment representation, although mainly for efficient map updates in the context of particle filter SLAM. To reduce the memory usage, the authors regularly perform a lossy maximum-likelihood compression. Compared to the compression technique used in our approach, this discards the probability information for future updates. The authors also did not address multi-resolution queries and the problem of map overconfidence.

As a data structure, octrees are applied in a variety of applications, most notably in the area of computer graphics for efficient rendering (Botsch et al., 2002; Laine and Karras, 2010; Surmann et al., 2003) and in the field of photogrammetry to store and address large point clouds (Elseberg et al., 2011; Girardeau-Montaut et al., 2005). Another popular use case is the compression of static point clouds (Schnabel and Klein, 2006) or point cloud streams (Kammerl et al., 2012). While our framework is general enough to also store raw point clouds, its main purpose is to integrate these point clouds into a memory-efficient, volumetric occupancy map, since point clouds as environment representation in robotics have a number of disadvantages as detailed at the beginning of this section.

A different probabilistic 3D environment representation for robotics was proposed by Yguel et al. (2007b). Their approach uses the Haar wavelet data structure to build memory-efficient multi-resolution maps. However, the approach does not distinguish between free and unknown space.

Surface representations such as the 3D Normal Distribution Transform (Magnusson et al., 2007) or Surfels (Habbecke and Kobbelt, 2007) were recently used for 3D path planning (Stoyanov et al., 2010) and object modeling (Krainin et al., 2011; Weise et al., 2009). Similarly, an accurate realtime 3D SLAM system based on a low-cost depth camera and GPU processing was proposed by Newcombe et al. (2011) to reconstruct dense surfaces in indoor scenes. This work was recently extended towards larger indoor environments (Whelan et al., 2013). However, surface representations are unable to distinguish between free and unknown space, may require large memory particularly outdoors, and are often based on strong assumptions about the corresponding environment. In mobile manipulation scenarios, for example, being able to differentiate free from unknown space is essential for safe navigation.

Finally, to the best of our knowledge, no open source implementation of a 3D occupancy mapping framework meeting our requirements is freely available.

2.5. Conclusion

2.5.1. Summary

In this chapter, we presented an efficient 3D mapping framework called OctoMap that is suitable for a number of robotic applications such as localization, collision avoidance, and exploration. Our approach uses an efficient data structure based on octrees that enables a compact memory representation and multi-resolution map queries. Compared to previous approaches, our environment representation is able to represent volumetric 3D information in a probabilistic manner, can distinguish between free and previously unseen areas, and is memory efficient. We proposed a bounded per-volume confidence that maintains the updatability of the map and leads to substantially reduced memory usage. We evaluated our approach with various real-world data sets. The results demonstrate that our approach is able to model the environment in an accurate way while reducing the memory consumption. Our implementation is available online as BSD-licensed C++ source code. The used datasets are publicly available as well to verify our experimental results and to compare against them.

2.5.2. Impact

Since its first introduction in 2010 (Wurm et al.), the OctoMap framework received a considerable interest in the robotics community due to its efficiency, versatility, and ease of integration. Beyond localization for humanoid robots and collision checking for mobile

manipulation presented in this thesis in chapters 3 and 7, OctoMap is used in a variety of robotics applications. These include exploration in 3D (Dornhege and Kleiner, 2011; Shade and Newman, 2011), 3D SLAM (Hertzberg et al., 2011), as well as navigation with aerial vehicles (Bry et al., 2012; Heng et al., 2011; Müller et al., 2011), humanoid robots (Maier et al., 2012), underwater robots (Papadopoulos et al., 2014), or even planetary rovers (Jayasekara et al., 2012). In *Project Tango*, Google employs OctoMap for 3D mapping with a mobile device that is equipped with a depth sensor.[5]

Our integration of OctoMap into ROS lead to a rapid adoption as standard 3D environment model for collision checking, first in the arm navigation and grasping pipeline (Ciocarlie et al., 2010), then in its successor *MoveIt!* (Chitta et al., 2012b). Example applications include mobile manipulation in unstructured environments (Chitta et al., 2012a) and constraint-aware teleoperation based on real sensor data (Leeper et al., 2013). OctoMap can be directly used as a native data type for collision checking in the Flexible Collision Library FCL (Pan et al., 2012), or in dynamically updatable distance maps (Lau et al., 2013).

[5] http://www.osrfoundation.org/blog/project-tango-announced.html (retrieved February 28, 2014)

Chapter 3

Monte Carlo Localization for Humanoid Robots

Accurate and reliable localization is a prerequisite for autonomously performing high-level tasks with any robot. We present a localization method for humanoid robots in arbitrary complex indoor environments using only onboard sensing. Localizing humanoid robots in these environments is a challenging task due to their large odometry drift and only noisy sensor readings. Since most humanoids walk with a swaying motion and are not forced to walk on flat ground only, a 6D torso pose has to be estimated. To this end, we apply Monte Carlo localization in the 3D environment model introduced in the previous chapter. We integrate range measurements from a 2D laser or depth camera as well as attitude data and information from the joint encoders. In extensive experiments, we demonstrate that our approach is able to globally localize a Nao humanoid and accurately track its 6D pose while walking.

When a humanoid robot is to perform a high-level task, such as cleaning up a room or delivering an object, it requires an accurate estimate of not only the objects and obstacles in the environment, but also its own location within the environment. Based on this pose estimate, the robot can then plan how to perform the task, i.e., how to reach the goal from its current location. The problem of determining this pose estimate in a known environment model is called localization and constitutes a classical problem in robotics.

While localization can be considered to be mostly solved for wheeled robots, it is still a challenging problem for humanoid robots. One major challenge of biped robots is the inaccurate execution of motions. This is due to their complex kinematic structure of a free-floating main body — usually the torso — connected to the environment over a series of joints in the legs. With the high number of joints, backlash adds up and cannot be neglected. While walking, inaccurate motion execution typically leads to foot slippage and consequently to only noisy and inaccurate odometry estimation. A second challenge is the limited payload of humanoid robots, imposed by their kinematic structure. Lightweight

sensors that are typically noisy have to be used for perception. Under all these constraints, robust techniques for localizing a humanoid robot are needed.

Since humanoids walk with a swaying motion and as they can step over various obstacles and rough terrain, the 2D localization approaches popular for wheeled robots are usually not applicable. For 2D localization, it is required that the sensor strictly moves on the plane that is represented in the 2D map. These restrictions become even more apparent when a humanoid robot is required to navigate in environments containing staircases or other three-dimensional structures, as we will discuss in Chapter 4. Thus, we propose the estimation of a 6D state of the humanoid including the height above the ground as well as the roll and pitch angles in addition to the 2D position and yaw angle. With this, we will be able to accurately estimate its pose while navigating in all kinds of environments without any restrictions on the environment structure.

In the previous chapter, we introduced an efficient, volumetric 3D environment model. This environment representation will be particularly useful for the three-dimensional localization approach presented here. We apply the probabilistic Monte Carlo localization (MCL), also known as particle filter localization, to estimate the robot's 6D torso pose using range data in the 3D map, e.g., from a laser scanner or depth camera mounted on the robot. Furthermore, our system integrates proprioceptive data provided by an attitude sensor and the joint encoders of the robot. To account for the noisy odometry information inherent to humanoid robots, we perform a calibration of the systematic drift and noise of the motion model based on least squares optimization. We furthermore present techniques for improving the integration of range data and evaluate different sensor models.

As we show in extensive real-world experiments with different Nao humanoids, our approach enables a humanoid to determine its global 6D pose and accurately track it while walking through different environments. We will rely on this ability in the following chapters to perform more complex navigation tasks.

3.1. Monte Carlo Localization (MCL)

Our approach estimates the six-dimensional pose $\mathbf{x} = (x, y, z, \varphi, \theta, \psi)$ of the robot's body reference frame in the global 3D map of the environment. The 6D pose $\mathbf{x}$ consists of the 3D position (x, y, z) and orientation as roll, pitch, and yaw angles (φ, θ, ψ).[1] The humanoid's reference frame is located in the center of its torso or waist, which is also the origin of all of its kinematic chains. For estimating the robot's 6D state, we apply the probabilistic Monte Carlo localization technique (Dellaert et al., 1998), which we briefly introduce in the following.

[1] While we refer to the 3D orientation (also called attitude) in the group SO(3) as three Euler angles for simplicity, the actual implementation is based on unit quaternions. This avoids the singularities inherent to Euler angles (Diebel, 2006).

MCL is a Bayes filtering technique that recursively estimates the belief about the robot's pose $Bel(\mathbf{x})$ as posterior distribution

$$Bel(\mathbf{x}) = p(\mathbf{x}_{0:t} \mid m, \mathbf{o}_{1:t}, \mathbf{u}_{1:t}) \tag{3.1}$$

given an environment model or map m, a sequence of observations $\mathbf{o}_{1:t}$, and a sequence of motions as odometry readings $\mathbf{u}_{1:t}$. At time t, the belief distribution over the robot's pose is approximated by a set of M particles as weighted samples or pose hypotheses

$$\mathcal{X}_t = \left\{ \langle \mathbf{x}_t^{[1]}, w_t^{[1]} \rangle, \ldots, \langle \mathbf{x}_t^{[M]}, w_t^{[M]} \rangle \right\}. \tag{3.2}$$

Here, each $\mathbf{x}_t^{[i]}$ is one pose hypothesis and $w_t^{[i]}$ is the corresponding weight. The weight of a particle is proportional to the likelihood that the robot is in that state.

Since we usually cannot directly sample from the target distribution $p(\mathbf{x}_{0:t} \mid m, \mathbf{o}_{1:t}, \mathbf{u}_{1:t})$, the algorithm recursively draws a new generation of particles from the proposal distribution $\pi(\mathbf{x}_{0:t} \mid m, \mathbf{u}_{1:t}, \mathbf{o}_{1:t})$. To account for the discrepancy between the proposal distribution and the target distribution, the particle weights are adjusted according to

$$w_t^{[i]} = \frac{p(\mathbf{x}_{0:t}^{[i]} \mid m, \mathbf{o}_{1:t}, \mathbf{u}_{1:t})}{\pi(\mathbf{x}_{0:t}^{[i]} \mid m, \mathbf{o}_{1:t}, \mathbf{u}_{1:t})}. \tag{3.3}$$

The target distribution can then be reconstructed as

$$Bel(\mathbf{x}) \simeq \sum_{i=1}^{M} w_t^{[i]} \, \delta(\mathbf{x}_t^{[i]} - \mathbf{x}), \tag{3.4}$$

where δ is the Dirac delta function.

Commonly, for localization the proposal distribution is represented by the odometry motion model and the particle weights are computed based on the observation model (Dellaert et al., 1998; Thrun et al., 2005). Under the Markov assumption, the current state $\mathbf{x}_t$ only depends on the previous state $\mathbf{x}_{t-1}$ for given sequences of observations and motions. This leads to a recursive formulation of estimating the state according to

$$p(\mathbf{x}_t \mid m, \mathbf{o}_{1:t}, \mathbf{u}_{1:t}) = \eta \underbrace{p(\mathbf{o}_t \mid m, \mathbf{x}_t)}_{\text{obs. model}} \int_{\mathbf{x}_{t-1}} \underbrace{p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t)}_{\text{motion model}} \underbrace{p(\mathbf{x}_{t-1} \mid m, \mathbf{o}_{1:t-1}, \mathbf{u}_{1:t-1})}_{\text{recursive term}} d\mathbf{x}_{t-1}. \tag{3.5}$$

Here, η is a normalization constant resulting from Bayes' rule. The observation or sensor model $p(\mathbf{o}_t \mid m, \mathbf{x}_t)$ denotes the likelihood of obtaining observation $\mathbf{o}_t$ given the robot is at pose $\mathbf{x}_t$ in the map m. The term $p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t)$ is called motion model and denotes the probability that the robot ends up in state $\mathbf{x}_t$ given it executed the motion $\mathbf{u}_t$ in state $\mathbf{x}_{t-1}$.

The update of the belief, also called particle filtering, now consists of the following steps for each iteration, which is usually referred to as *sampling-importance-resampling*:

1. **Prediction**: The current set of particles is propagated according to the proposal distribution, in this case the motion model $p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t)$. Hereby, a new pose is sampled from a probability distribution for each particle to account for the motion noise.

2. **Correction**: The importance weight of each particle is computed according to the observation model $p(\mathbf{o}_t \mid m, \mathbf{x}_t)$ given the map. This step is also referred to as the measurement update. Particles from the previous distribution that fit well to the observations will be assigned a high weight, which corresponds to a high likelihood.

3. **Resampling**: New particles at time step $t+1$ are drawn with replacement from the distribution proportional to the particle weights $w_t^{[i]}$. Afterwards, their weights are reset to $w_{t+1}^{[i]} = 1/M$. This step ensures that the filter converges to pose hypotheses with high likelihoods.

The filter is either initialized with a distribution of equally weighted samples around an initial pose estimate for tracking, or with a uniform distribution over all possible hypotheses for global localization.

To avoid the problem of particle depletion, where potentially good particles are replaced, we perform selective resampling as proposed by Doucet (1998). The resampling step is only applied when the effective number of particles

$$n_{\text{eff}} = 1 / \sum_i \left(w_t^{[i]} \right)^2 \tag{3.6}$$

drops below the threshold of $M/2$. Additionally, we apply low-variance resampling, which ensures that each particle with a weight $w_t^{[i]} \geq \frac{1}{M}$ is drawn at least once (Thrun et al., 2005, Chapter 4).

Following Eq. (3.4), we obtain the most likely pose estimate from the state distribution $\mathcal{X}_t$ by the weighted mean of its particle poses. In case a multimodal distribution remains after resampling, the particles need to be clustered first. Then, the weighted mean pose of each cluster represents a potential pose estimate. Computing the mean of rotations can be problematic since they represent a non-linear manifold of the vector space. However, by assuming reasonably small differences (below 40°) between the particle orientations, we can estimate the mean rotation by averaging over the unit quaternions and re-normalizing the average quaternion afterwards (Gramkow, 2001).

Compared to a Kalman filter, MCL is a non-parametric representation that is not restricted to unimodal Gaussian distributions, and it can be directly applied for an initial global localization by using a uniform random distribution over the complete map. Further advantages are that, due to the independent estimation of state hypotheses, a particle

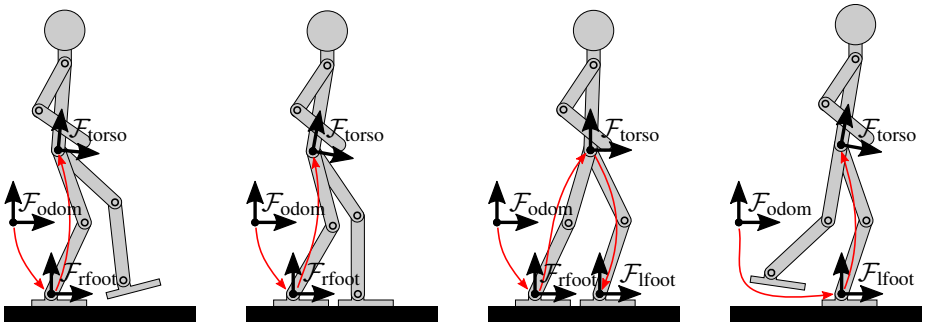


Figure 3.1.: Coordinate frames for kinematic walking odometry of a biped robot under ideal conditions. Initially, the frame $\mathcal{F}_{\text{rfoot}}$ of the right stance foot is fixed in the world frame $\mathcal{F}_{\text{odom}}$ and the torso frame $\mathcal{F}_{\text{torso}}$ can be computed with forward kinematics over the right leg. In the next step, the left swing leg touches the ground. While both feet remain on the ground, the left foot frame $\mathcal{F}_{\text{lfoot}}$ is computed with forward kinematics over both leg chains. Finally, for the next stepping phase the left leg becomes the stance leg and $\mathcal{F}_{\text{lfoot}}$ remains static with respect to $\mathcal{F}_{\text{odom}}$, which creates a new reference to compute $\mathcal{F}_{\text{torso}}$.

filter can be easily parallelized and it can directly benefit from more computational power by increasing the number of particles. Thus, the possible estimation accuracy directly scales with the available computational resources.

3.2. Motion Model

In the prediction step of MCL, a new pose is drawn for each particle according to the motion model $p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t)$. In our case, $\mathbf{u}_t$ corresponds to the incremental motion of the humanoid's torso in a local coordinate frame and is computed with forward kinematics from the current stance foot while walking. This kind of odometric estimate is usually referred to as kinematic odometry. Figure 3.1 shows a single stepping phase and the coordinate frames used to compute the kinematic odometry. As illustrated in Figure 3.2, $\mathbf{u}_t$ can then be computed from two consecutive torso poses $\widetilde{\mathbf{x}}_{t-1}$ and $\widetilde{\mathbf{x}}_t$, which are 6D rigid body transforms in the odometry coordinate frame $\mathcal{F}_{\text{odom}}$:

$$T(\mathbf{u}_t) = (\widetilde{\mathbf{x}}_{t-1})^{-1} \oplus \widetilde{\mathbf{x}}_t \,. \tag{3.7}$$

Here, $T(\mathbf{u}_t)$ denotes the 6D rigid body transform of the odometry motion $\mathbf{u}_t$. On the other hand, we can think of the humanoid's odometric pose estimates as a concatenation of the incremental motions

$$\widetilde{\mathbf{x}}_t = \widetilde{\mathbf{x}}_{t-1} \oplus T(\mathbf{u}_t) \,. \tag{3.8}$$

Under ideal conditions the stance foot of the robot rests firmly and precisely on the ground, leading to an accurate 6D pose estimate with kinematic odometry. However, the feet of the robot typically slip on the ground due to inaccurate execution of motions. Additionally, gear backlash in individual joints can aggravate the problem. Hence, in practice,

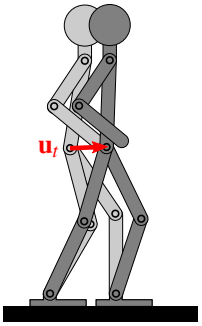


Figure 3.2.: Odometry estimate $\mathbf{u}_t$ from two consecutive torso poses during the walking cycle.

only noisy estimates of the odometry are available while walking and a substantial amount of drift accumulates over time. Consequently, a particle filter has to account for that noise with a higher variance, requiring a higher number of particles and thus more computational power for successful pose estimation. By learning the motion model parameters instead, the localization performance can be increased, both in terms of computational load as well as accuracy.

Here, we consider the most general case of any kind of 3D positional and rotational displacement, for instance originating from an omnidirectional walking engine. We furthermore assume that systematic drift affects the motion reported by odometry in the 2D plane, i.e., only $(\widetilde{x},\widetilde{y},\widetilde{\psi})$ are affected from $\widetilde{\mathbf{x}} = (\widetilde{x},\widetilde{y},\widetilde{z},\widetilde{\varphi},\widetilde{\theta},\widetilde{\psi})$. This is not a strong restriction as long as the humanoid walks on a solid surface, since its motion is constrained by this surface and it cannot leave the ground. Even when climbing challenging terrain such as stairs, the drift of the motion occurs in the 2D plane of the stance leg as long as the robot does not fall or slide down a slope. General noise in the kinematic estimate of the humanoid's height above the ground does not lead to a systematic drift.

3.2.1. Motion Model Calibration

For odometry calibration, we will refer to the reduced state vectors containing 2D position and orientation as $\mathbf{x}' = (x, y, \psi)^\top$. Corresponding to Eq. (3.8), $\mathbf{u}'_t = (u_{x,t}, u_{y,t}, u_{\psi,t})^\top$ estimates the displacement between two poses reported by odometry

$$\widetilde{\mathbf{x}}'_t = \widetilde{\mathbf{x}}'_{t-1} + \mathbf{u}'_t \,. \tag{3.9}$$

To calibrate the drift of $\mathbf{u}'_t$, we assume that a ground truth pose $\mathbf{x}'$ is available in a prior learning phase, e.g., from an external motion capture system, scan matching, or visual odometry. Based on the deviations from the ground truth, values of a calibration matrix $\mathbf{M} \in \mathbb{R}^{3\times3}$ can be determined to correct the 2D drift of odometry, such that

$$\mathbf{x}'_t = \mathbf{x}'_{t-1} + \mathbf{M}\,\mathbf{u}'_t \,. \tag{3.10}$$

We describe how such a calibration matrix can be learned in the next section.

To account for general motion noise, the particle prediction step adds multivariate Gaussian noise to $\mathbf{u}'_t$ for each particle i, whereas the systematic drift is accounted for by the mean of the Gaussian distribution $\mathcal{N}$. Similar to Eliazar and Parr (2004), we assume independent noise sources for the drift in forwards and sideways direction, as well as in orientation. This results in a noise model in which the mean depends linearly on $\mathbf{u}'_t$ and the variance quadratically. Thus, the prediction step samples for particle i a new pose according to:

$$\mathbf{x}_t'^{(i)} = \mathbf{x}_{t-1}'^{(i)} + \mathcal{N}(\mathbf{M}\,\mathbf{u}'_t, \boldsymbol{\Sigma}\,\mathbf{u}_t'^2)\,. \tag{3.11}$$

Here, $\boldsymbol{\Sigma}$ is a matrix containing vectors of noise parameters and $\mathbf{u}_t'^2 = (u_{x,t}^2, u_{y,t}^2, u_{\psi,t}^2)^\top$ contains the squared odometry readings. The first element of $\mathbf{x}_t'^{(i)}$, for example, computes to

$$x_t^{(i)} = x_{t-1}^{(i)} + \mathcal{N}(M_{xx}u_{x,t} + M_{xy}\,u_{y,t} + M_{x\psi}\,u_{\psi,t}, \sigma_{xx}^2 u_{x,t}^2 + \sigma_{xy}^2 u_{y,t}^2 + \sigma_{x\psi}^2 u_{\psi,t}^2)\,. \tag{3.12}$$

By filling the remaining values for z, roll, and pitch we obtain the final motion model $p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t)$ in 3D. Since these dimensions are constrained by the ground plane when walking, we assume zero-mean noise with variances that depend on the traveled distance since the last odometry reading.

3.2.2. Learning the Motion Model Parameters

In this section, we introduce our approach to learn the motion model parameters, so that the Gaussian distribution in Eq. (3.11) best approximates the actual drift and noise by the walking humanoid.

Let $\mathbf{U} \in \mathbb{R}^{N\times 3}$, where each row $U_i = (u_{x,i}, u_{y,i}, u_{\psi,i})$ contains the incremental odometry motion estimates for each of the N timesteps acquired while recording the ground truth data for calibration, e.g., with an external tracking system. Following the idea of least-squares odometry estimation by Eliazar and Parr (2004), $\mathbf{M}_x = (M_{xx}, M_{xy}, M_{x\psi})^\top$ is the column vector containing the elements of the mean calibration matrix $\mathbf{M}$ that influence x, and $\mathbf{X}_x \in \mathbb{R}^{N\times 1}$ contains the ground truth measurements of the incremental displacements in x-direction. We can now solve the overdetermined system

$$\mathbf{U}\mathbf{M}_x = \mathbf{X}_x \tag{3.13}$$

for $\mathbf{M}_x$ with least squares to determine the first row of $\mathbf{M}$. The remaining rows $\mathbf{M}_y$ and $\mathbf{M}_\psi$ can be solved analogously.

To estimate the variance parameters $\boldsymbol{\Sigma}$, we define $\mathbf{U}^2 \in \mathbb{R}^{N\times 3}$ as a vector of squared odometry motions such that $U_i^2 = (u_{x,i}^2, u_{y,i}^2, u_{\psi,i}^2)$, $\boldsymbol{\Sigma}_x = (\Sigma_{xx}, \Sigma_{xy}, \Sigma_{x\psi})^\top$ as the column

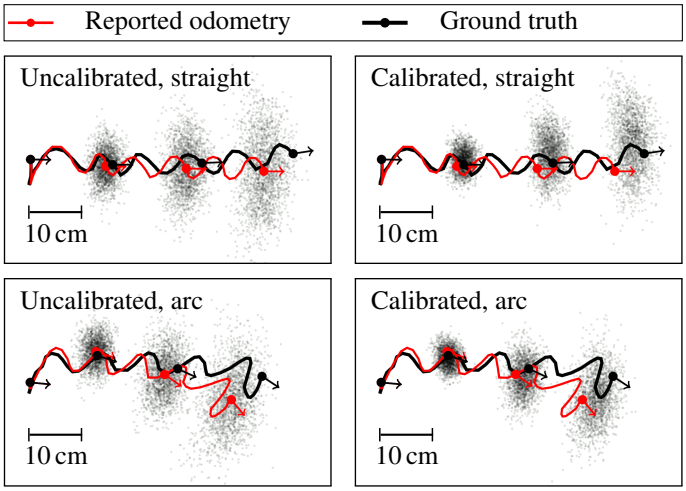


Figure 3.3.: Comparison between an uncalibrated odometry motion model (left) and a calibrated one (right) for walking straight and on an arc with a Nao humanoid. The uncalibrated motion model requires a larger variance such that the particles can account for the systematic drift. 2000 particles were used in each iteration for this visualization.

vector of variances in x, and $\mathbf{X}_{x^2} \in \mathbb{R}^{N\times 1}$ as the measurements on the variances such that $X_{x^2,i} = (U_i\mathbf{M}_x - X_{x,i})^2$. We can now solve the linear equation

$$\mathbf{U}^2\boldsymbol{\Sigma}_x = \mathbf{X}_{x^2} \tag{3.14}$$

and the corresponding ones for $\boldsymbol{\Sigma}_y$ and $\boldsymbol{\Sigma}_\psi$. With the solutions $\mathbf{M} = \left(\mathbf{M}_x\,\mathbf{M}_y\,\mathbf{M}_\psi\right)$ and $\boldsymbol{\Sigma} = \left(\boldsymbol{\Sigma}_x\boldsymbol{\Sigma}_y\boldsymbol{\Sigma}_\psi\right)$ we have now calibrated the systematic drift and noise parameters of the motion model in Eq. (3.11).

Figure 3.3 shows the difference between a calibrated and an uncalibrated motion model for a humanoid walking straight and on an arc. Note that the calibration was obtained with short sequences of sample motions in all directions, not only with the two illustrated motions. The calibrated motion model properly captures the drift since the particle distribution closely represents the ground truth. Compared to that, the uncalibrated motion model requires larger variances, and thus more particles, to account for systematic drift.

3.3. Observation Model

In our localization approach, we update the belief about the humanoid's 6D state based on three different sources of sensor information contained in one observation $\mathbf{o}_t$. First, range measurements $\mathbf{r}_t$ provided by a laser range finder or depth camera are integrated. Second, we regard the height $\widetilde{z}_t$ of the humanoid's torso above the current ground plane as a measurement of its joint encoders and also integrate the angles for roll $\widetilde{\varphi}_t$ and pitch $\widetilde{\theta}_t$

as estimated by an inertial measurement unit (IMU). Since all these measurements are conditionally independent, the observation model decomposes to the product

$$\begin{aligned} p(\mathbf{o}_t \mid m, \mathbf{x}_t) =& p(\mathbf{r}_t, \widetilde{z}_t, \widetilde{\varphi}_t, \widetilde{\theta}_t \mid m, \mathbf{x}_t) \\ =& p(\mathbf{r}_t \mid m, \mathbf{x}_t)\; p(\widetilde{z}_t \mid m, \mathbf{x}_t)\; p(\widetilde{\varphi}_t \mid \mathbf{x}_t)\; p(\widetilde{\theta}_t \mid \mathbf{x}_t)\,. \end{aligned} \tag{3.15}$$

The individual factors of the sensor model are described in detail in the following.

3.3.1. 3D Environment Representation

For humanoid navigation in non-planar, possibly multi-level environments, a full 3D occupancy grid map is necessary since the map needs to encode both occupied and free volumes for arbitrary structures. Here, we make use of the efficient 3D environment representation developed in Chapter 2. This enables our system to use map resolutions as small as 1 cm for a complete 3D indoor map. In this work, we assume a volumetric 3D representation of the environment as given.

3.3.2. Range Measurements

To integrate range measurements, usually the endpoint model (also called likelihood field) or ray casting (also called beam model) are used for K beams $r_{t,k}$ of a range measurement $\mathbf{r}_t$ that consists of a complete scan from a laser or a depth image from a camera.

In the endpoint model (Thrun, 2001), the likelihood of a single range measurement $r_{t,k}$ depends on the distance $D(r_{t,k})$ of the corresponding hypothetical beam endpoint to the closest obstacle represented in the map, i.e.,

$$p(r_{t,k} \mid m, \mathbf{x}_t) = \phi(D(r_{t,k}), \sigma_r)\;, \tag{3.16}$$

with the Gaussian distribution

$$\phi(d, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{d^2}{2\sigma^2}\right). \tag{3.17}$$

Here, σ_r is the standard deviation of the sensor noise. For a given 3D map, the distances D can be pre-computed for all 3D cells with a Euclidean distance transform (Lau et al., 2013). Since the lookup can then be implemented as a single query, the endpoint model is a popular and efficient choice for laser-based localization (Kümmerle et al., 2008). However, it neglects the beam characteristics of range sensors, which cannot pass obstacles between the sensor origin and its endpoint. This potentially leads to a less accurate localization especially in three-dimensional environment structures.

In the ray casting model, a ray is cast from the origin of the sensor in the current beam direction until it hits an obstacle. This returned distance $\widetilde{d}(r_{t,k})$ is then compared with the actually measured beam distance $d(r_{t,k})$:

$$p(r_{t,k} \mid m, \mathbf{x}_t) = \phi\left(\widetilde{d}(r_{t,k}) - d(r_{t,k}), \sigma_r\right) . \tag{3.18}$$

While the ray casting operation is computationally demanding particularly in 3D, it is better informed about three-dimensional structures, which can result in an improved localization accuracy. In Section 3.5, we will compare the performance of ray casting with the endpoint model. For efficiency, we rely on a parallelized ray-casting implementation in our 3D mapping framework OctoMap (see Chapter 2).

We extend the single probabilities $p(r_{t,k} \mid m, \mathbf{x}_t)$ to a full probabilistic model as detailed by Thrun et al. (2005, Chapter 6), including terms for maximum range measurements to account for sensor errors and terms for random measurements to account for unmodeled objects in the sensor beam. The full sensor model then represents a mixture between the Gaussian distribution $p(r_{t,k} \mid m, \mathbf{x}_t)$ and uniform distributions for maximum-range measurements as well as random measurements.

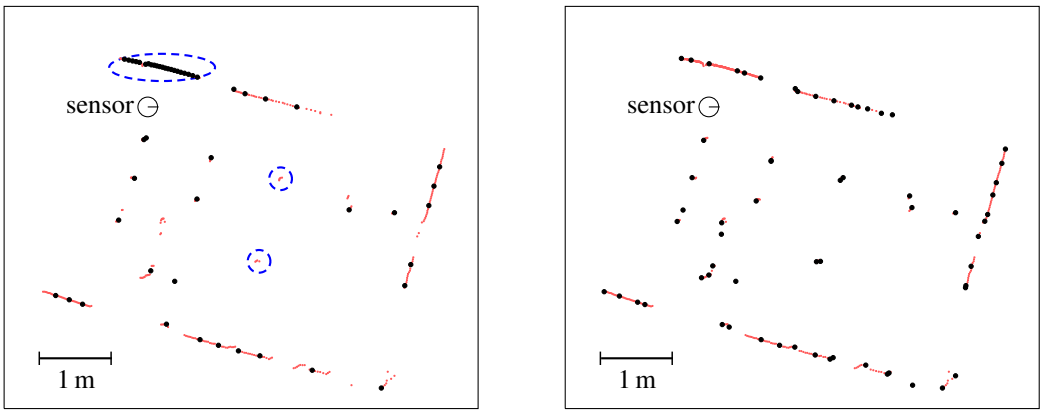
Finally, the integration of a full scan $\mathbf{r}_t$ is computed as the product of the beam likelihoods

$$p(\mathbf{r}_t \mid m, \mathbf{x}_t) = \prod_{k=1}^{K} p(r_{t,k} \mid m, \mathbf{x}_t) \tag{3.19}$$

with the common assumption of conditional independence between the beams. In practice, the independence can be assured by using only a subset of all the available range measurements. A common practice is to use every n-th beam, e.g., at intervals of 5°. While this angular subsampling is straightforward to implement, it exhibits a number of problems, as highlighted in Figure 3.4a. First, endpoints on close obstacles are spaced close to each other while obstacles further away only have few points on them. This results in an overconfidence on close-range readings, since they are no longer conditionally independent. Second, thin obstacles that are observed only with few endpoints but could provide unique features for localization can be missed. To overcome these problems, we implement a subsampling of the endpoints in the Cartesian space. All points falling into one cell of an equally-spaced grid are approximated by their centroid. As can be seen in Figure 3.4b, our Cartesian space subsampling strategy results in regularly-spaced endpoints that represent the actual scan much closer with the same number of points and the assumption of conditional independence between the points is better justified. The same technique can be applied to 3D point clouds, e.g., from depth or stereo cameras.

3.3.3. Roll, Pitch, and Height Measurements

Furthermore, we need to integrate the torso height $\widetilde{z}_t$ above the ground plane as computed from the values of the joint encoders and the roll $\widetilde{\varphi}_t$ and pitch $\widetilde{\theta}_t$ provided by the IMU.



(a) Angular subsampling **(b)** Cartesian space subsampling

Figure 3.4.: Comparison of different techniques for selecting a subset of endpoints (black dots) from a 2D laser scan (light red dots): Angular subsampling takes every n-th point (a), whereas our method subsamples the end points based on their distance in Cartesian space (b). Highlighted by dashed ellipses are potential problems in angular subsampling: Close obstacles receive a higher weighting by more endpoints, and thin obstacles can be missed.

Here, we evaluate the difference between the quantities predicted according to the motion model and their measured values. Similar to the endpoint model for range measurements, their likelihoods are computed according to the Gaussian density function in Eq. (3.17):

$$p(\widetilde{z}_t \mid m, \mathbf{x}_t) = \phi(z_{t,\text{ground}} - \widetilde{z}_t, \sigma_z) \tag{3.20}$$

$$p(\widetilde{\varphi}_t \mid \mathbf{x}_t) = \phi(\varphi_t - \widetilde{\varphi}_t, \sigma_\varphi) \tag{3.21}$$

$$p(\widetilde{\theta}_t \mid \mathbf{x}_t) = \phi(\theta_t - \widetilde{\theta}_t, \sigma_\theta)\,, \tag{3.22}$$

where σ_z, σ_ϕ, and σ_θ are given by the noise characteristics of the joint encoders and the IMU, and $z_{t,\text{ground}}$ is computed from the height difference between z_t and the closest ground level in the map.

Finally, the complete measurement update step of the localization can be combined to the product of Eq. (3.19)–(3.22) according to Eq. (3.15) by choosing appropriate weighting factors for the individual likelihoods.

3.4. Global Localization in Multi-Level Environments

When the robot has no initial guess about its pose, it needs to estimate the pose globally. In MCL, this is done by initially distributing the hypotheses over possible robot poses in the whole environment, in our case over all floor levels. To efficiently draw robot pose hypotheses, we sample x, y, and ψ uniformly within free areas of the map and z from the probability distribution given by Eq. (3.20). For each sampled 2D position (x, y),

Figure 3.5.: Nao humanoid with a laser range finder in Experimental Environment I. A volumetric OctoMap representation of the environment is shown in Figure 2.10.

we hereby consider the initial height measurement $\widetilde{z}_0$ at all represented ground levels. Similarly, roll and pitch are sampled according to Eq. (3.21) and (3.22) for an initial IMU measurement $(\widetilde{\varphi}_0, \widetilde{\theta}_0)$.

Obviously, global localization requires more particles than pose tracking. However, once the initial particle cloud has converged, the robot's pose can be tracked using fewer particles.

Overconfidence in the estimated pose can lead to the algorithm losing track of the localization, which is hard to recover when there are no particles left near the actual true pose. A common solution to this issue is to randomly insert global particles into the tracking estimation from time to time, thus giving the algorithm a chance to recover.

3.5. Evaluation

In a set of localization experiments, we evaluated various aspects of our localization approach with two different Nao humanoids. The experimental environments are shown in Figure 3.5 and Figure 3.6. A volumetric 3D map of these environments was manually constructed, as illustrated in Figure 2.10 for example.

As the estimated localization pose of the robot, we used the weighted mean of the particle distribution. To provide ground truth data, we used a *MotionAnalysis Raptor-E* motion capture system in both environments. We initialized the particle filter pose at the ground truth pose returned by the motion capture system for the tracking experiments.

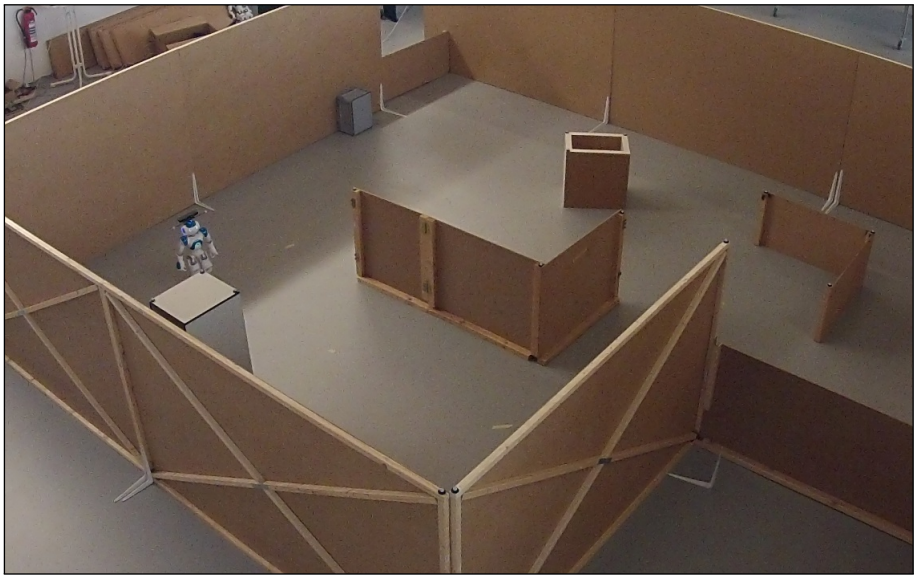


Figure 3.6.: Nao humanoid with an RGBD-camera in Experimental Environment II.

3.5.1. Humanoid Robot Platform

As robotic platforms in the localization experiments, we used two different Nao humanoids which are described in detail in Appendix A. The first set of experiments was performed with a V3 Nao equipped with a laser head, using range data from a Hokuyo URG-04LX laser range finder. The second Nao humanoid has the V4 hardware revision and is equipped with a consumer-grade Asus Xtion Pro Live RGB-D camera, from which we use the range image. Since the error of this sensor grows quadratically in the distance (Khoshelham and Oude Elberink, 2012), we adjusted σ_r in the range measurement models of Eq. (3.16) and Eq. (3.18) to quadratically depend on the measurement range r. To sample rays from the depth camera data, our system separates all end points of the corresponding point cloud into ground and non-ground parts by means of a RANSAC-based plane detection. Our approach uses this information for sampling rays from non-ground points uniformly over the Cartesian space with a higher density than ground points. Thus, we compensate for the fact that beams hitting the floor can provide no information for estimating the translation in the horizontal plane, which is typically more important than height, pitch, or roll.

As further sensors, we used the Hall effect joint encoders for kinematic odometry and the on-board IMU to measure roll and pitch of the humanoid's torso. The measurements of this small and lightweight IMU are quite noisy compared to the IMUs often used in robotics. However, especially while walking, these values can be more accurate than the roll and pitch obtained through kinematics of the measured support leg joint angles, because the robot's feet may not always precisely rest on the ground. In all our experiments, we teleoperated the robot with an omnidirectional velocity input. From that, the default

walking engine (Gouaillier et al., 2010) generated the gait for walking straight, on arcs, sideways, and for turning.

In practice, odometry and other sensor data do not arrive at discrete timesteps but asynchronously and with different update rates. To solve this problem and achieve time synchronization, we update the MCL filter based on range sensor data, interpolating odometry and IMU sensor data between two valid measurements. A second problem stems from the fact that a range image is not generated instantaneously but over a certain amount of time in which the robot may be moving. To this end, we apply temporal uncertainty sampling as introduced by Thompson et al. (2006). For each particle, odometry data is interpolated to a timestamp which is sampled uniformly around the current measurement timestamp in an interval corresponding to the time needed for a complete scan (0.1 s).

Throughout our experiments, we used a Cartesian sampling distance for beam endpoints of 30 cm and integrated new observations only after the robot has traveled a distance of 15 cm or changed its orientation for more than 23° since the previous sensor integration to ensure conditional independence of the sensor integrations.

3.5.2. Run Time Performance

We first evaluated the run times of the range sensor model. It constitutes the most computationally expensive part of MCL, but can be efficiently parallelized. We compared the endpoint model to ray casting in a map of Environment I (Figure 3.5) with 1 cm resolution on a standard desktop CPU (Intel Core i7-2600, 3.4 GHz). The average number of considered end points here was 60, with an average beam length of 1.68 m. Through parallelization with four threads, we achieved average run times as fast as 0.013 ms per particle for the endpoint model, and 0.448 ms per particle for ray casting (single-threaded: 0.024 ms / 1.172 ms). This results in a maximum realtime update rate of 11.2 Hz for 200 particles with ray casting and parallelization. Note that in practice, sensor measurements are only integrated after the robot has moved for a certain distance to ensure conditional independence between updates. As the ray casting time directly depends on the map resolution and number of end points, this can be further optimized depending on the scenario at hand.

3.5.3. Laser-Based Pose Tracking

With the laser range finder, we evaluated the performance of our localization approach in Environment I for different sensor and motion models against ground truth from the motion capture system. Since any Monte Carlo method is susceptible to the effects of pseudo-random number generators, we evaluated the errors as mean and 95% confidence interval over ten differently seeded localization runs of the same recorded sensor data.

To calibrate the odometry motion model using the method described in Section 3.2.2, the humanoid executed a set of sample movements such as walking forward, sideways, or turning on the spot before performing the actual experiments. We then used the data of

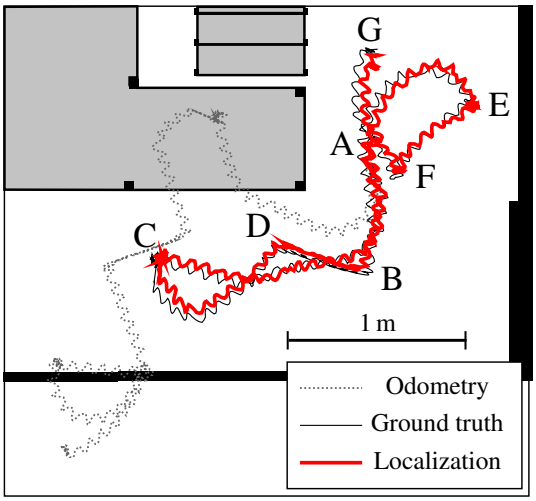


Figure 3.7.: Trajectory estimated with our localization approach compared to ground truth and odometry on the lower level of Environment I. Based on laser range data, 200 particles were used with ray casting and a calibrated odometry motion model. The robot started walking forward at (A), then walked on a right turn at (B) towards (C) where it turned left on the spot and walked on an arc to (D). From there, it walked sideways to (B) and continued forwards over (A) to (E). At (E) it turned on the spot, walked forward to (F), turned again, and walked backwards to the goal at (G).

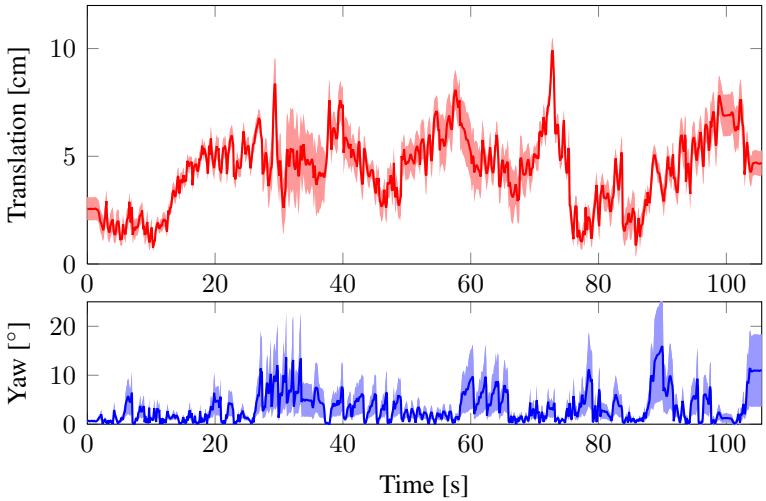


Figure 3.8.: Mean translational and yaw error with 95% confidence interval for $N = 10$ runs with 200 particles in the scenario of Figure 3.7.

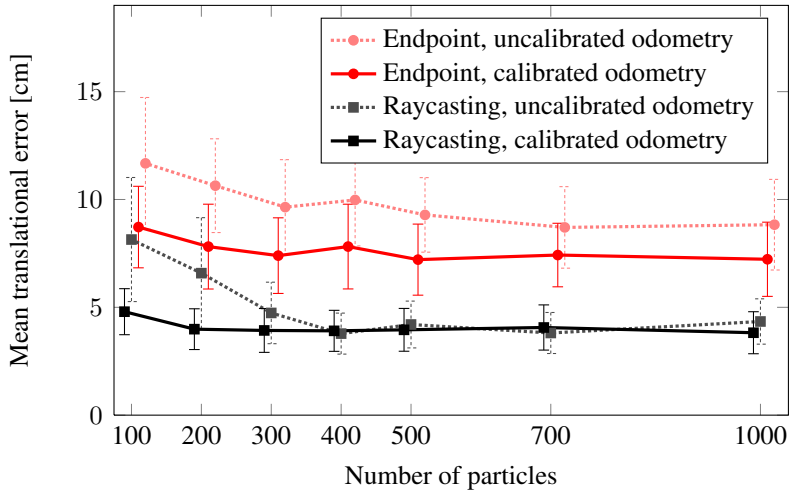


Figure 3.9.: Mean translational error with 95% confidence interval for laser-based localization on the lower level of Environment I ($N = 10$ runs each). Raycasting leads to a significantly lower error. A calibrated odometry motion model requires less particles to achieve the same performance as an uncalibrated one.

the incremental odometry motions and the ground truth from motion capture to obtain the systematic drift and noise parameters of the motion model. In the uncalibrated model, the drift is accounted for with larger noise parameters.

The complete trajectory for the evaluation of the localization performance has a length of approximately 5 m and includes parts of walking forward, on arcs, sideways, backwards, and turning on the spot. Figure 3.7 shows a single trajectory, whereas Figure 3.8 shows the error over time for ten runs when using 200 particles. The mean translational error for all trajectories was 3.9 cm and the mean yaw error 1.7°. Roll and pitch errors are not shown but generally below 3°. Pose estimation based on odometry quickly diverges while our localization approach is able to track the ground truth closely through all walking motions. Figure 3.9 shows the aggregated results for different numbers of particles. Note that the angular errors behave similarly but are not shown for clarity. As can be seen, the endpoint model, while computationally more efficient, is not able to capture the full 3D structure of the environment and results in a significantly higher localization error (t-test, 99% confidence). A calibrated odometry motion model generally improves the localization performance when using only few particles. This improvement is statistically significant up to 200 particles (t-test, 95%). Hence, we used ray casting and the calibrated motion model in the next experiments.

Estimating only x, y, and ψ in a 2D map with ray casting and odometry calibration leads to a higher average error of 4.6 cm, whereas our method results in 3.9 cm when using 200 particles. While this difference is not statistically significant, the individual deviations can be much higher, potentially leading to collisions with obstacles when navigating with an inaccurate pose estimate. In all trajectories, the maximum error was 14.5 cm (transla-

tion) / 22° (yaw) for 2D localization and 9.5 cm / 9° for our 3D localization, both using 200 particles with ray casting and calibrated odometry. This demonstrates that the humanoid's swaying motion of up to 5° in each direction needs to be considered and indeed a full 6D pose estimation results in the highest accuracy.

A video demonstrating our laser-based localization approach is available online at http://www.youtube.com/watch?v=uiIi2rSKWAU.

3.5.4. Laser-Based Global Localization

Figure 3.10 shows the evolution of the particle distribution for global localization with 50 000 particles. Initially, the complete area of Environment I was covered with pose hypotheses. After three sensor updates, the particles converged to the ground truth. Throughout 10 runs in the same setting, our global localization approach always converged to a pose within 10 cm of the ground truth requiring at most four sensor updates, which demonstrates the reliability of our approach for global localization.

3.5.5. Comparing Laser-Based With Depth Camera-Based Localization

In this experiment, we evaluated the localization performance when using the consumer-level Asus Xtion depth camera instead of the Hokuyo laser range finder. Both Nao robots are nearly identical except for the installed range sensors (see Section 3.5.1 and Appendix A for details). We calibrated the motion model parameters for both robots as explained in Section 3.5.3. Both robots were teleoperated to follow the same path of approximately 8 m length through Environment II (Figure 3.6), while they were tracked with the motion capture system. We compared ten localization runs using 60 beams of each sensor on average for a localization update. As can be seen in Figure 3.11, the mean error increases when using the depth camera due to its narrow horizontal field of view and increasing noise for longer measurements. The difference is statistically significant for 100 particles (t-test, 99%). Maximum errors over all 10 localization runs were 13.4 cm (translation) / 7° (yaw) when using the laser, and 21.4 cm / 8° when using the depth camera. As visible in Figure 3.12, the error for depth camera based localization is highest when the robot has only few obstacles to observe nearby, e.g., between locations (A) and (B) (10 – 30 s) or when approaching (D) (82 – 100 s). Nevertheless, both sensor configurations are able to accurately track the humanoid's motions while odometry quickly diverges.

3.6. Related Work

In the last few years, many approaches for tracking the pose of humanoid robots in the two-dimensional space have been proposed. In these approaches, the robot's pose is represented by the 2D position and yaw angle. Hereby, many authors rely on vision information

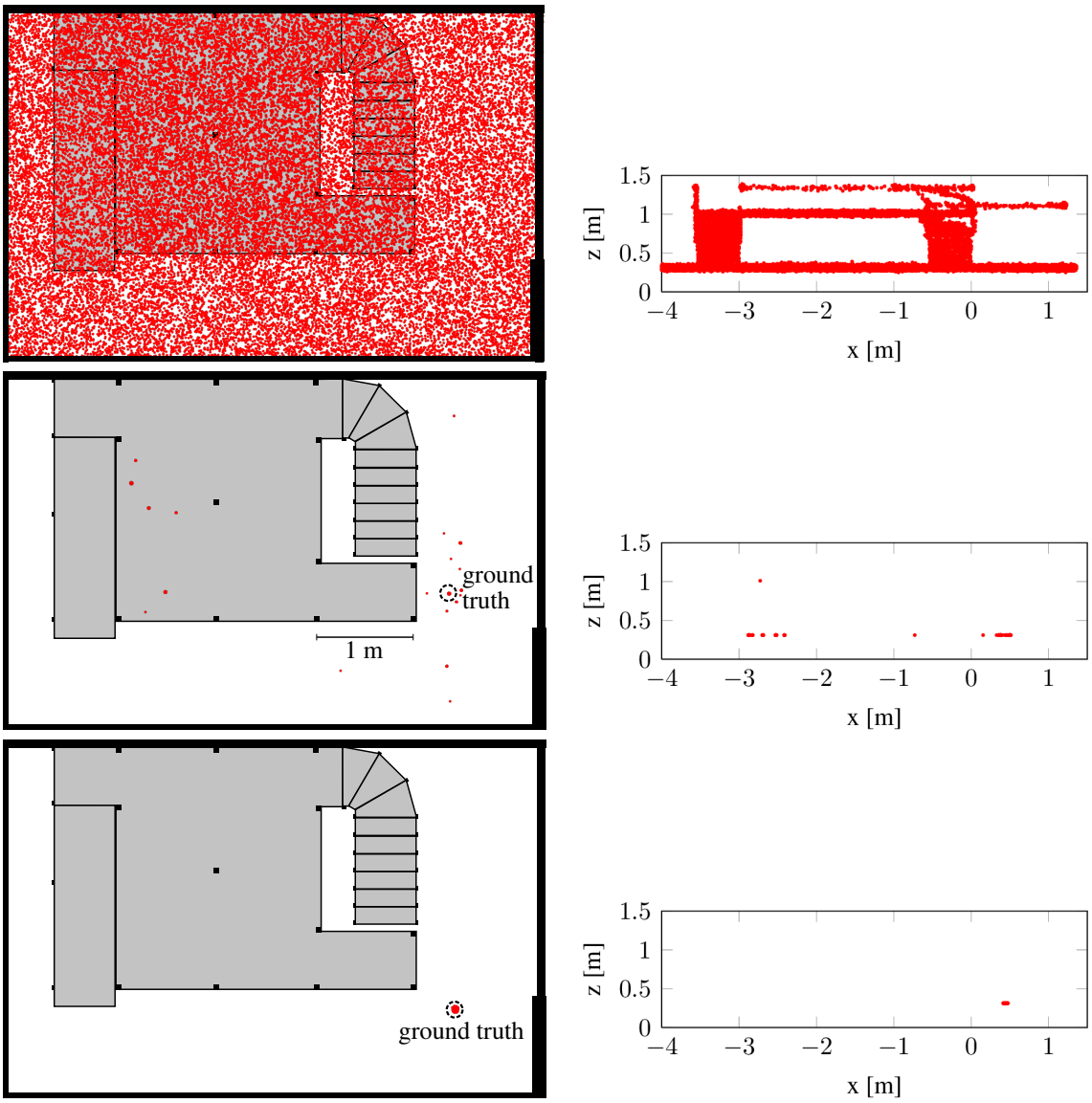


Figure 3.10.: Particle distribution over time for global localization with 50 000 particles in Environment I. The left column shows a projection onto the xy-plane, while the right column shows a side view. Initially (top row), particles were distributed uniformly over all surfaces including the ramp and staircase. The first sensor update (middle row) resulted in a few clusters, one on the upper level at $z = 1$ m. After three sensor updates, the particles converged to the ground truth (bottom row).

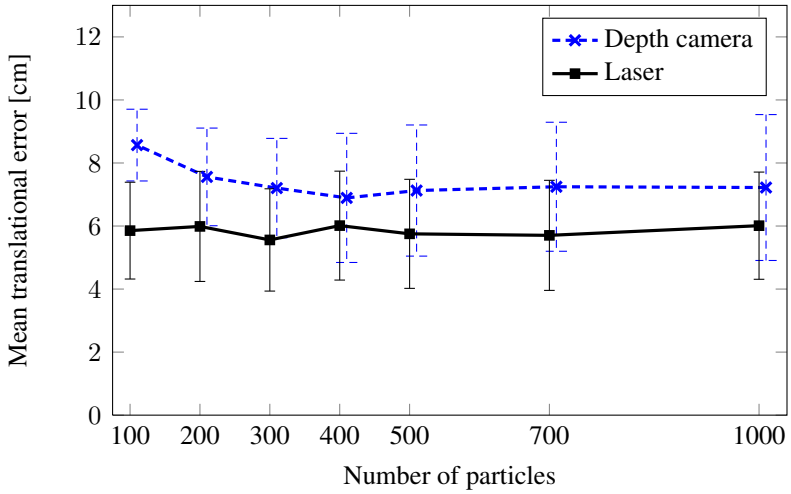
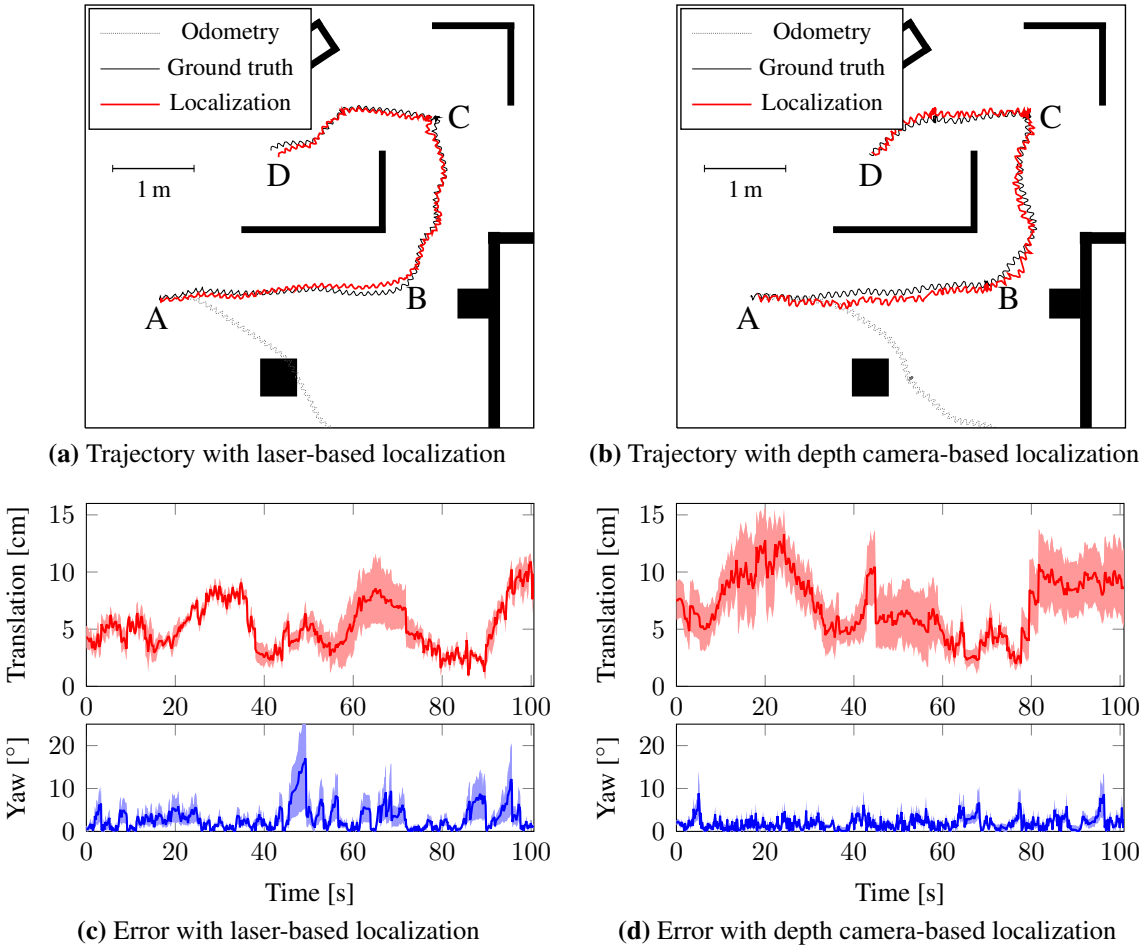


Figure 3.11.: Mean translational error with 95% confidence interval for laser-based and depth camera-based localization in Environment II ($N = 10$ runs each).

from monocular or stereo cameras due to their light weight. Bennewitz et al. (2006) use SIFT features of a humanoid's onboard monocular camera and match them to a recorded map of features for localization. For vision-based localization, motion blur due to the shaky walking motion of humanoids is often a problem. To this end, Ido et al. (2009) determine stable phases of the walking gait and obtain images only then, resulting in sharp observations. They compare camera images to a pre-recorded reference sequence for 2D localization. Oßwald et al. (2010) apply reinforcement learning such that the humanoid learns a reliable and efficient navigation strategy. It trades off walking towards the goal against standing still to reduce its uncertainty by integrating sharp and clear images. The authors integrate observations as SURF features from a humanoid's onboard camera in an unscented Kalman filter. Pretto et al. (2009) introduced a variant of SURF that is robust to motion blur and track these visual features over time to estimate the robot's odometry. Cupec et al. (2005) detect objects with given shapes and colors in the local environment of the humanoid and determine its pose relative to these objects. Seara and Schmidt (2004) use stereo vision and known landmarks for estimating the humanoid's foot positions in the 2D plane with a Kalman filter while walking. By actively controlling the gaze, the authors reduce the error of the estimated poses.

More recently, techniques relying on small and lightweight laser range finders have also been developed. Stachniss et al. (2008) presented an approach to learn accurate 2D grid maps of large environments with a humanoid equipped with a Hokuyo laser scanner. Such a map was subsequently used by Faber et al. (2009) for localization of a humanoid tour guide robot. Similarly, Tellez et al. (2008) developed a navigation system for such a 2D environment representation. Their SLAM approach builds 2D maps from odometry and laser range finders located in the humanoid's feet.



(a) Trajectory with laser-based localization

(b) Trajectory with depth camera-based localization

(c) Error with laser-based localization

(d) Error with depth camera-based localization

Figure 3.12.: Comparison of the trajectories and localization error over time for laser-based and depth camera-based localization with 300 particles in Environment II ($N = 10$ runs each). The robots started at (A), walked forward to (B), then on a curve to (C). There, they turned on the spot and continued forward to the goal (D).

Since a 2D map is often not sufficient for humanoid navigation, several methods rely on 2.5D grid maps that additionally store a height value for each cell. Thompson and Kagami (2005) developed an early particle filter localization approach based on stereo data. In the sensor model, the stereo data was projected into 2D and matched to the known elevation map. This enabled the authors to track the humanoid H7's 2D pose with orientation while walking. The authors later extended their work to estimate the 6D pose of HRP-2 (Thompson et al., 2006). Here, they used a Hokuyo URG-04LX laser range finder in the 2.5D elevation map. However, they assume that the robot is only walking on flat ground, constraining height, roll, and pitch within fixed thresholds.

In further approaches, several authors rely on odometry instead of a global pose estimate of the robot. For example, Chestnutt et al. (2009) and Nishiwaki et al. (2012) estimate the robot's motion using only kinematic odometry. Based on this pose estimate, the robot builds a local 2.5D height map from a tilting laser range finder located in its hip. Gutmann et al. (2008) build a local height map for obstacle avoidance from stereo data. To avoid problems resulting from the accumulated error due to odometry drift, old data is discarded after a short period of time in these approaches. In recent work, Ahn et al. (2012) propose to fuse kinematic and visual odometry with IMU data in order to estimate the pose of a humanoid walking with a heel-toe motion. Relying only on kinematic odometry for this motion is problematic, since the feet do not rest on the ground during the transition phase, but rather roll over their rear and front boundaries, the heel and toe. While the authors' approach improves the odometry estimate, it is still subject to drift over time. With such local tracking based on odometry, it is also not possible to globally determine the robot's initial position in a given map.

Michel et al. (2007) localize a humanoid with respect to a close object. The authors apply a model-based approach to track the 6D pose of a manually initialized object relative to the camera. Stasse et al. (2008) and Alcantarilla et al. (2013) proposed an approach to simultaneously localizing the robot and mapping the environment. The authors combine vision and motion information to estimate the pose and velocities of the camera as well as visual feature positions in 3D while the robot is walking on a small circle. Similarly, Kwak et al. (2009) discussed grid-based particle filter SLAM for the humanoid HRP-2. In their approach, they build a grid map of 10 cm resolution with point clouds from the robot's stereo vision cameras.

Finally, there exist navigation systems for humanoid robots which use external sensors to track the robot's pose during experiments (Chestnutt et al., 2009; Michel et al., 2006; Stilman et al., 2007). However, external sensing is usually not practical to employ outside a lab setting.

In contrast to all of these approaches, we presented a system which is able to accurately determine the complete 6D pose of a humanoid robot in a 3D representation of a complex, multi-level environment using only on-board sensors. The use of range sensors leads to a high accuracy of the estimated pose.

Based on our approach, Maier et al. (2012) implemented a navigation system that localizes a humanoid in a known environment model with a depth camera. The robot then

builds a local, three-dimensional obstacle map with the approach presented in Chapter 2. During navigation, the robot can avoid dynamically appearing obstacles and plan around them.

3.7. Conclusion

We presented a probabilistic localization approach for humanoid robots based on a full 3D representation of the environment. Our system is able to deal with all challenges occurring during humanoid robot navigation. This includes highly inaccurate odometry information, inherently noisy sensor data, and the violation of the flat world assumption. We apply Monte Carlo localization to globally determine and track a humanoid's 6D pose, consisting of the 3D position and the three rotation angles. Hereby, we integrate range data from a 2D laser or an RGBD-camera, as well as attitude estimates from an IMU and measurements from the joint encoders.

We thoroughly evaluated and discussed the presented methods. As we showed in a series of real-world experiments with different Nao humanoids, our method is able to globally estimate the 6D pose of the humanoid's torso and then accurately track it while walking. Ray casting with a 2D laser range finder resulted in the highest accuracy, whereas the usage of a consumer-level depth camera slightly decreased the accuracy. Our presented method of odometry calibration lead to an improved performance, particularly when using only few particles. Our approach and the released implementation is generally applicable to robots equipped with range sensors in a given 3D model of the environment.

Chapter 4

Autonomously Climbing Stairs

Climbing stairs is an essential capability of humanoid robots since staircases are commonly present in all kinds of man-made environments. This chapter presents techniques for all aspects of the task. To perceive staircases, we extend and evaluate two methods for extracting planes from 3D range data: scan-line grouping and two-point random sampling. Based on the reconstructed staircase model, the robot then executes a series of whole-body motions learned from human demonstrations through kinesthetic teaching. An accurate pose estimate is critical while climbing stairs to avoid falling down. To this end, we extend the 6D localization approach from the previous chapter to rely on detected edges in monocular camera images in addition to range and proprioceptive data. All observations are integrated in an extended particle filter using improved proposals, leading to highly accurate pose estimates. Using our methods, a Nao humanoid is able to reliably ascend a complex staircase with ten steps including a winding part.

Man-made environments can contain different kinds of terrain that pose challenging obstacles for robots. Staircases are a common occurrence in domestic environments containing multiple levels, and industrial settings commonly contain ramps or even ladders. While these environment features can easily be overcome by humans, they pose serious challenges and fall hazards for robots. This constitutes a major road block for the general adoption of assistive robots in environments designed for humans.

The human-like body plan of humanoid robots promises an adoption to these environment features, as these robots can step over different types of terrain with their bipedal locomotion. In this chapter, we focus on the ability of humanoid robots to climb staircases, as stairs are a common feature in domestic environments. We hereby consider all aspects of the task, from an initial perception of stairs in sensor data to whole-body motion generation and localization while climbing stairs.

In a first step, the robot acquires an accurate model of the staircase that is then used to determine the sequence of stepping motions, but also to aid the localization. The primary feature of staircases is a set of planar surfaces. To detect these, we adapt, evaluate, and

compare two approaches to plane extraction from range data. These approaches have been successfully applied to humanoid navigation in the presence of stairs in the past. The first approach was presented by Gutmann et al. (2008). It is based on scan-line grouping and first extracts line segments from neighboring scan lines in a range image. The line segments are successively combined to plane segments if they lie on the same plane. This algorithm is highly efficient since it relies on the initially extracted line segments rather than repeatedly processing the 3D points. The second approach was developed by Kida et al. (2004) and relies on two-point random sampling. This method directly operates on the 3D point cloud. The key idea here is to estimate main directions in the environment by randomly sampling points and determining their difference vectors. Afterwards, plane segments orthogonal to the main directions are found with a clustering technique. We propose several improvements to the two-point random sampling method to increase the robustness in complex environments containing multiple planes and clutter. From the planes detected by the two algorithms, we reconstruct the staircase model by intersecting the planes and determining the outline of the contained measurements.

For ascending or descending the detected staircase, a humanoid then has to execute a series of whole-body motions in order to balance its body. Small steps or perturbations from the ground plane could be traversed with compliant motions in a walking controller, but larger steps usually require more elaborate motions to avoid falling over. Here, we apply kinesthetic teaching to learn the motion for climbing a single step from a human demonstration. For climbing a complete staircase, the robot then regularly re-aligns with the next step based on its localization estimate to account for motion drift.

In the previous chapter, we introduced a probabilistic localization method based on range data, IMU, and joint encoders. Since our localization approach estimates the 6D torso pose, it is also applicable to multiple levels and when climbing any kind of terrain. However, even a localization error of only a few centimeters poses a fall hazard during the critical process of stair climbing since the robot might miss a step, or inadvertently bump into steps or a handrail. This hazard is even more pronounced when executing a fixed set of whole-body motions, as it is the case in our approach. Thus, we propose to augment the localization estimate with vision observations of the robot's direct vicinity. To this end, we present a vision observation model based on chamfer matching between an edge model of the environment and a set of observed edges in camera images. The vision observations are directly integrated into the observation model of the particle filter. By calculating an improved proposal distribution based on range and vision observations, the particle filter generates highly focused particle sets that lead to an accurate pose estimate.

We evaluated our methods with a Nao humanoid equipped with a laser range finder. The experimental environment, shown in Figure 3.5, contains a challenging staircase with ten steps and a winding part. As the experiments demonstrate, both methods for plane detection show comparably good results and lead to accurate 3D models of the staircase. Our improved localization approach leads to a highly robust navigation behavior on the stairs, with a success rate of 100% for climbing the complete staircase autonomously with the Nao.

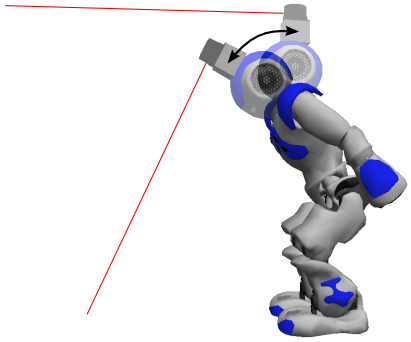


Figure 4.1.: The Nao humanoid acquires a 3D point cloud by tilting its head with the 2D laser range finder on top. The figure illustrates the minimum and maximum head tilt angles.

4.1. Stair Reconstruction from Range Measurements

This section discusses the perception of staircases based on 3D range measurements. We here focus on the two approaches scan-line grouping and two-point random sampling. Other approaches for plane segmentation are based on expectation maximization (EM) (Thrun et al., 2003; Triebel et al., 2005) or RANSAC (Rusu et al., 2009), but are not covered in detail in this work. EM for plane segmentation is time-consuming to compute for large point clouds and thus not practical in our scenario. RANSAC, in our experience, tends to over-simplify complex planar structures. For example, multiple small steps are often merged into one sloped plane.

4.1.1. Data Acquisition

We assume that the humanoid robot is equipped with a sensor for 3D range data to detect the staircase. This sensor could be a tilting laser range finder, a stereo camera pair, or an RGBD-camera with active illumination such as the Microsoft Kinect or Asus Xtion.

In our experiments, we use the laser-equipped Nao humanoid described in Appendix A. The robot is employed in the environment shown in Figure 3.5, which contains two levels, a winding staircase with ten steps, and a ramp. By tilting its head and combining the 2D scan lines over time, the robot obtains 3D point cloud data of the environment. This process is illustrated in Figure 4.1. Note that the sensor placement and head joint limits do not allow to observe the area directly in front of the robot's feet. Figure 4.2 illustrates an example 3D scan with the robot standing close to the first step of the staircase. Whereas the first step and parts of the second one are outside the field of view, the higher steps suffer from partial occlusions. Therefore, in practice the robot has to acquire a new scan every two steps to be able to build an accurate model of the staircase. Figure 4.3 visualizes the 3D scan data of three single steps from the side in an orthographic projection. As can be seen, the data is noisy with errors up to 5 cm, which makes plane fitting a challenging

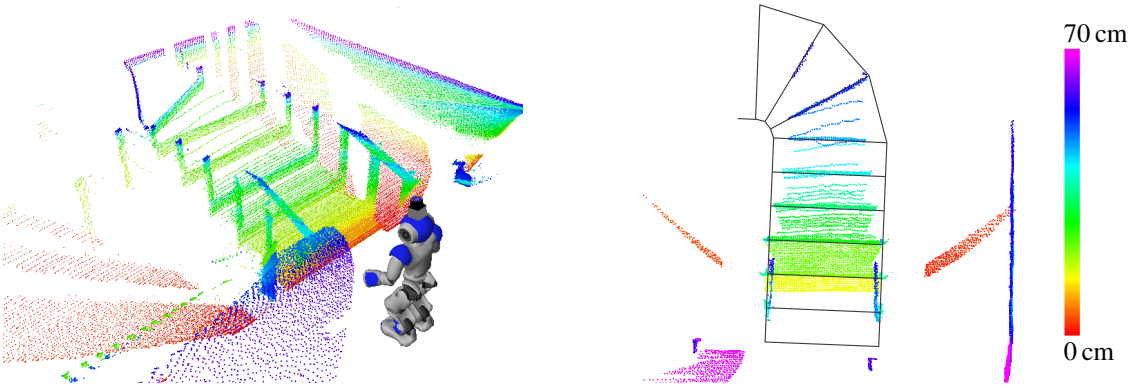


Figure 4.2.: 3D point cloud acquired by a Nao humanoid with its head-mounted laser range finder. For better visibility, height is indicated by color. The right figure shows a projection from the top with a manually constructed model of the staircase for comparison.

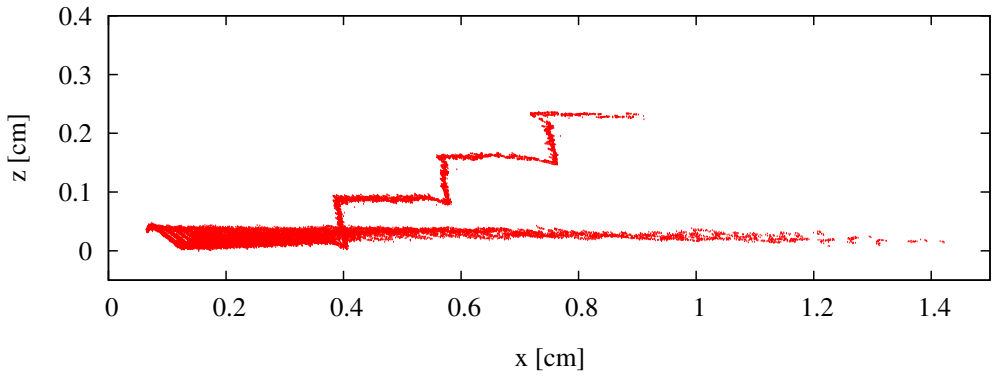


Figure 4.3.: Side view of the 3D point cloud corresponding to three steps of a staircase. Each step is 7 cm high. The data suffers from noise which poses challenges for detecting planar surfaces.

task. The noise originates from the inaccuracy of the laser scanner itself (Kneip et al., 2009) as well as from the estimation of the sensor pose from forward kinematics based on the joint angles while tilting the head.

4.1.2. Scan-Line Grouping

A fast and efficient method for extracting planes from a range image is to first fit straight line segments to each image row and then perform region growing using the lines as primitives to form planar regions. The approach was originally presented by Jiang and Bunke (1994) and performed best in a comparison with other earlier range segmentation techniques (Hoover et al., 1996). Since then the algorithm has been extended and improved in order to deal with range data containing varying levels of noise, such as range images obtained from stereo vision (Gutmann et al., 2008). The method was successfully applied on Sony's QRIO robot, allowing the humanoid to recognize floor, stairs, obstacles, and tables, and to plan and navigate its path on an obstacle course (Gutmann et al., 2005).

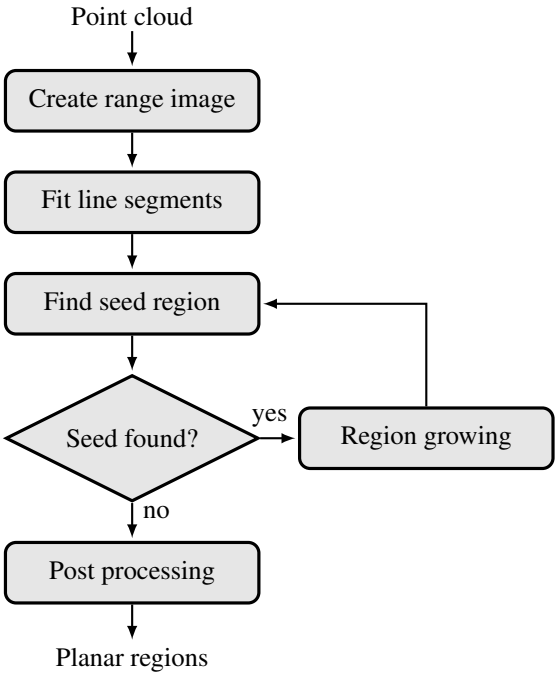


Figure 4.4.: Flow chart of the scan-line grouping algorithm, based on Gutmann et al. (2008).

For the general case of 3D point clouds, we extended the scan-line grouping approach with an initial step that creates a range image from the point clouds. Figure 4.4 shows a flow chart of the complete algorithm.

In the first step, the range scans are assembled into a range image where each pixel holds a 3D point in a coordinate system aligned with the image plane. The advantage of such an image representation is that the contours of regions can be found by looking at the 3D points of neighboring pixels. As the laser readings are collected by tilting the head of the robot, a range image can be assembled by sorting the individual range scans by the pitch angle of the neck joint. In order to obtain more homogeneous data, scans with a similar pitch angle (difference less than 0.18°)[1] are discarded. Figure 4.5 shows the resulting range image from the 3D scan in Figure 4.2.

Note that the creation of a range image becomes more involved as the trajectory of the range finder becomes more complex, e.g., when also yaw and roll of the head are changing. In fact, if the robot was to walk while collecting the range scans, a range image representing the full 3D data might not exist due to the possible occlusion of objects.

In the second step, line segments are fit to each range row of pixels in the image. This is achieved by partitioning the 3D points into groups where a new group is started whenever the distance between neighboring points exceeds a threshold (5 cm). For each group, we fit a straight line by least squares optimization. Using a run distribution test, we evaluate the accuracy of the line: if more than a number (15) of consecutive points all fall on one

[1]Numbers in parentheses indicate the values of parameters used in our experiments in Section 4.4.

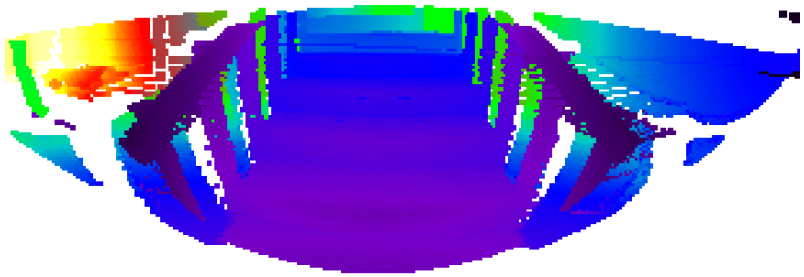


Figure 4.5.: Range image for the 3D scan in Figure 4.2. Depth is encoded by color, where darker colors indicate ranges closer to the sensor.

side of the line, the group is split at the point most distant to the chord connecting start and end point of the group. The algorithm then processes each sub-group recursively. Lines with less than a number of points (5) or shorter than a threshold (5 cm) are discarded.

The extraction of a plane starts by finding three line segments in neighboring image rows that overlap in their start and end columns. The three lines must pass a statistical test before considering them as a seed for a plane: the standard deviation of the plane fit to all points of the three lines (using least squares) must not be larger than a factor (2) of the standard deviation of each line. Once a seed region has been determined, a region growing procedure is performed that adds lines in neighboring rows and columns to the plane as long as the standard deviation of the line points with respect to the plane is less than a factor (2.5) of the standard deviation of the line itself.

When no more seed regions can be found, the scan-line grouping algorithm terminates. In a post-processing step, the found planes are refined. This includes checking whether line segments or single data points better fit to a neighboring plane, the merging of co-linear planes, and plane optimization by discarding border pixels. For a complete description, we refer to the original publication (Gutmann et al., 2008).

An interesting property of scan-line grouping is that the algorithm accesses most input points only when fitting line segments, resulting in an efficient implementation.

4.1.3. Two-Point Random Sampling

Two-point random sampling as introduced by Kida et al. (2004) relies on the assumption that, in man-made environments, most surfaces are aligned with respect to a small set of main directions. To exploit this information, the two-point random sampling algorithm first determines the main directions of the point cloud by sampling. Then, the algorithm cuts the data into slices perpendicular to each main direction and searches for plane segments within each slice. Note that we cannot simply assume planes perpendicular to the z normal because the scan may be tilted due to small errors in the robot's pose, and because the front faces of the steps are also important for the extraction.

Basic Algorithm

Given a set of points $\mathbf{p}_1, \ldots, \mathbf{p}_N$, the algorithm extracts plane segments as follows.

1. Sample a set $\mathcal{V}$ of normalized difference vectors ($|\mathcal{V}| = 10000$):

$$\mathcal{V} \leftarrow \left\{ \frac{\mathbf{p}_i - \mathbf{p}_j}{\|\mathbf{p}_i - \mathbf{p}_j\|} \middle| 1 \leq i, j \leq N, i \neq j \right\} \tag{4.1}$$

   The elements of $\mathcal{V}$ can be interpreted as points on the unit sphere. Points $\mathbf{p}_i, \mathbf{p}_j$ originating from different surfaces in the environment generate randomly distributed points on the unit sphere. If $\mathbf{p}_i$ and $\mathbf{p}_j$ both originate from a common flat surface in the environment, then the corresponding point in $\mathcal{V}$ is located on the unit circle with the same normal vector as the surface. Hence, all points drawn from surfaces with similar normal vectors produce rings with a high point density on the unit sphere. By considering the point density, planar surfaces can be distinguished from the noise occurring in the data.

2. Find a ring-shaped cluster $\mathcal{R} = \{\mathbf{v}_1, \ldots, \mathbf{v}_M\} \subseteq \mathcal{V}$.

3. Determine the normal vector of $\mathcal{R}$ by optimizing:

$$\mathbf{n} \leftarrow \arg\max_{\mathbf{n}} \sum_{k=1}^{M} \begin{cases} 1 & \text{if } \mathbf{n}^\top \mathbf{v}_k \approx 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

4. Cut the point cloud into thin slices perpendicular to $\mathbf{n}$.

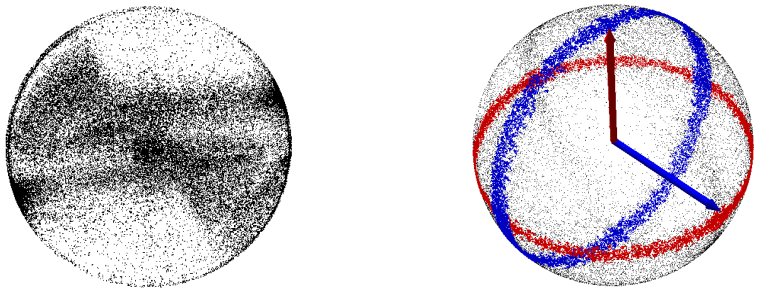
5. For each slice containing a sufficient number of points (20): Cluster the points within the slice to plane segments.

Remove the normal vectors of the cluster $\mathcal{R}$ from $\mathcal{V}$ and repeat Steps 2) to 5) to find the remaining planes.

Improved Version

While the basic algorithm is sufficient for scenes containing a moderate number of planes, the robustness has to be improved for more complex scenes containing many planes or clutter. We propose the following four improvements to increase the robustness.

During the sampling step, the original algorithm draws pairs of points randomly from the whole point cloud. In more complex environments, the probability that both points originate from the same surface is low, so that the resulting rings on the unit sphere are not dense enough to be easily distinguishable from the noise in the data. Additionally, the directions of the difference vectors are biased if the dimensions of the environment are not approximately equal. Drawing the second point $\mathbf{p}_j$ from the nearest neighbors of the first



(a) Uniform sampling **(b)** Improved sampling

Figure 4.6.: Normalized difference vectors $\mathcal{V}$ from the point cloud in Figure 4.2 as points on the unit sphere (black dots). With uniform sampling, there are no distinct main directions. Using our improved sampling considering the local point information, the distinct rings corresponding to the main directions can be clearly identified (indicated by their normal vectors as red and blue arrows).

point $\mathbf{p}_i$ with $d_1 \leq \|\mathbf{p}_i - \mathbf{p}_j\| \leq d_2$ increases the probability that both points originate from the same surface and reduces the bias. The lower boundary reduces the susceptibility to noise. In our scenario, we used the values $d_1 = 2\,\text{cm}$ and $d_2 = 7\,\text{cm}$. Figure 4.6 shows the elements of $\mathcal{V}$ as points on the unit sphere for the point cloud in Figure 4.2. As can be seen, when using uniform sampling, the main directions cannot be reliably estimated. In contrast, our sampling strategy that considers neighborhood information leads to clearly distinguishable rings corresponding to the main direction.

As a next improvement, we combine Step 2) and 3) of the algorithm by applying RANSAC to estimate the main directions. Here, we repeatedly sample two points in $\mathcal{V}$ and determine their normal vector to find the main directions.

In complex environments, the slices that are considered in Step 5) may contain columns and handrails or intersect planes that are perpendicular to the slice. The algorithm might mistake these objects for small plane segments, as they contain a similar number of points. Therefore, we analyze the eigenvalues $\lambda_1 \leq \lambda_2 \leq \lambda_3$ of the covariance matrix of the segmented points. We reject thin and long plane segments, which are indicated by a large difference between the largest and second-largest eigenvalue $\lambda_3 \gg \lambda_2$.

In a final step, we merge neighboring planes that probably originated from the same surface. Due to measurement noise, the points of a single surface may fall into two adjacent slices. As only points within one slice are considered for finding plane segments, these points generate two distinct, parallel plane segments. Hence, we identify and merge planes with similar parameters after the extraction. In particular, planes are merged if their distance is at most 3 cm and their normals have an angular difference of at most 11°.

4.1.4. Reconstruction of Staircases

Based on the planes detected by either scan-line grouping or two-point random sampling, we can reconstruct stairs in a straightforward manner. First, we intersect horizontal and

vertical planes with each other. This yields a set of straight horizontal lines that correspond to the front and back boundary of the step surface. Next, we determine the boundaries of each line segment by projecting the convex hull of points belonging to a plane onto the lines. Finally, the endpoints of parallel lines are connected to form the staircase model. Models from multiple view points can be merged based on a known pose estimate of the robot.

4.2. Learning to Climb a Single Stair

After detecting and reconstructing the staircase model as described in the previous section, the robot can use that model to ascend or descend the staircase. It hereby needs to maintain its balance and avoid hitting the steps or a handrail unintentionally, since this would result in a fall. We here follow the idea that the stair climbing motions are very similar after the robot repositioned itself in front of a single step, and learn the whole-body motion with kinesthetic teaching. This process is supported by an estimation of the robot's center of mass to guarantee statically stable motions. An alternative approach would be to plan stable motions for the whole body as discussed in Chapter 6. However, planning long trajectories for systems with many degrees of freedom is not yet feasible for online navigation.

In our experimental environment, the individual steps have a height of 7 cm, which is comparable to steps in the real world when scaling the robot to human size. Since the steps are still rather challenging to climb with the Nao's body plan, it has to execute a full body motion in order to keep its balance, i.e., it also has to move its torso, head, and arms. Due to the added laser sensor at the top of the head, the robot has a relatively high center of gravity to balance.

For kinesthetic teaching, we make use of *Choregraphe*, a graphical tool developed by Aldebaran Robotics for the Nao humanoid (Pot et al., 2009). In particular, we remove the stiffness from the humanoid's joints, either of single groups or complete chains. A human then demonstrates the motion by manually moving the robot into different body configurations. While the joints are passive, the Hall effect sensors yield information about the pose, which we then use to estimate the robot's center of mass with the detailed weight distribution of each link. With this, we can record statically stable keyframe poses of the stair climbing motion. The continuous trajectory of joint angles between the keyframes is obtained with a Bézier spline interpolation, which yields a smooth motion. Finally, we adjust the timings in a post-processing step to optimize the execution time.

The resulting motion, shown in Figure 4.7, enables the robot to climb stairs of the given height that was used when learning the motion. Steps with a smaller height can be climbed by adjusting the downwards motion of the leg with inverse kinematics.

From the motion learned for climbing with one leg, we obtain a corresponding motion for the other side with a dedicated mirroring operation of the joint angles. That means that angles for the right half of the body are applied to the left and vice versa, while joint

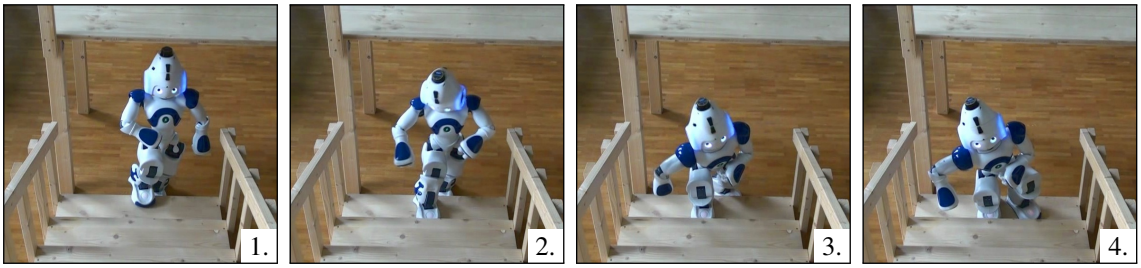


Figure 4.7.: Nao climbing a single step of 7 cm height with a learned whole-body motion.

angles of shared joints such as the neck are simply inverted. Alternating the climbing motion for the left and the right leg distributes the strain on the servos more evenly, which enables the robot to climb more steps before it needs to interrupt due to overheating.

For this learned open-loop motion, there is only a small error margin. The robot has to be at most 1 cm away from the step, otherwise it risks falling backwards after reaching the level of the next step. Similarly, the robot has to avoid bumping the stairs' handrail with its arms so it should position itself at the center of the stairs. To deal with these challenges and improve the localization accuracy, we augment the laser-based localization introduced in Chapter 3 with visual information while climbing stairs.

4.3. Highly Accurate Localization Using Improved Proposals With Vision Observations

Our range-based localization, introduced in Chapter 3, typically results in a pose error below a few centimeters. While climbing stairs or other types of challenging terrain, even this relatively small error constitutes a fall hazard since the robot might miss a step or collide with an obstacle. Additionally, odometry estimates may become highly unreliable and noisy while executing whole-body motions. To this end, we propose to augment the localization estimate with vision observations in an improved proposals particle filter. Vision observations yield information about the robot's direct vicinity, such as the critical area directly in front of its feet. This area could be unobservable from head-mounted range sensors, or be within a dead spot commonly observed with stereo and RGBD-sensors at close ranges due to their camera baseline or properties of the active illumination projector. Our vision observation model is based on chamfer matching between a known edge model of the environment and detected edges in camera images. The edge model of the staircase can be learned through a reconstruction as detailed above, or can be provided as a CAD model. Edges are typically present in all kinds of structured environments and can be detected in monocular images with high accuracy. The vision observations are then fused with range data in an extended MCL algorithm using an improved proposal distribution.

4.3.1. Improved Proposal Distribution

During challenging whole-body motions such as stair climbing, odometry information in MCL (see Chapter 3) may become highly unreliable and noisy, leading to a flat proposal distribution of the motion model. In contrast, the observation likelihood is peaked so only a small number of particles have high weights and cover the meaningful areas of the target distribution. Hence, a large number of particles is required to sufficiently represent the posterior distribution in MCL. In order to achieve more focused particle sets, which require fewer particles to accurately represent the posterior, we therefore use an improved proposal.

According to Doucet et al. (2000), the optimal proposal distribution in terms of minimizing the variance of the importance weights is

$$\pi\left(\mathbf{x}_{0:t} \mid m, \mathbf{u}_{1:t}, \mathbf{o}_{1:t}\right) := p(\mathbf{x}_t \mid m, \mathbf{x}_{t-1}^{[i]}, \mathbf{o}_t, \mathbf{u}_{t-1}) \tag{4.3}$$

$$= \frac{p(\mathbf{o}_t \mid m, \mathbf{x}_t)\; p(\mathbf{x}_t \mid \mathbf{x}_{t-1}^{[i]}, \mathbf{u}_{t-1})}{p(\mathbf{o}_t \mid m, \mathbf{x}_{t-1}^{[i]}, \mathbf{u}_{t-1})}\,. \tag{4.4}$$

Computing this proposal analytically requires to evaluate the denominator

$$\eta^{[i]} := p(\mathbf{o}_t \mid m, \mathbf{x}_{t-1}^{[i]}, \mathbf{u}_{t-1}) \tag{4.5}$$

$$= \int p(\mathbf{o}_t \mid m, \mathbf{x}_t)\; p(\mathbf{x}_t \mid \mathbf{x}_{t-1}^{[i]}, \mathbf{u}_{t-1})\; \mathrm{d}\mathbf{x}_t\,. \tag{4.6}$$

However, there is no closed-form solution for the integral in the general case.

Depending on the context, different solutions were proposed to approximate the integral in Eq. (4.6) or the proposal distribution itself. In the context of SLAM with Rao-Blackwellized particle filters and landmark observations, Montemerlo et al. (2003) introduced an algorithm that approximates an improved proposal by linearizing the observation models. However, in our case it is not possible to linearize the observation model for laser and vision measurements because of the unpredictable shape of the observation likelihoods. Grisetti et al. (2007) introduced improved proposals in the context of grid-based SLAM and proposed to approximate the integral in Eq. (4.6) as a finite sum:

$$\eta^{[i]} \simeq \sum_{j=1}^{K} p(\mathbf{o}_t \mid m, \mathbf{x}_j^{[i]})\; p(\mathbf{x}_j^{[i]} \mid \mathbf{x}_{t-1}^{[i]}, \mathbf{u}_{t-1})\,, \tag{4.7}$$

where $\{\mathbf{x}_1^{[i]}, \ldots, \mathbf{x}_K^{[i]}\}$ is a set of sample points drawn around the particle pose. This sampling technique can be used to efficiently approximate the proposal if the sampled points cover meaningful regions of the proposal and these regions are sufficiently small. The proposal distribution typically only has one mode, which we determine by scan matching the point cloud of the range sensor in the 3D environment model. We assume that the

meaningful regions of the proposal distribution will be in the vicinity of the mode, thus we can cover the meaningful area of the proposal distribution by drawing a low number of samples from a uniform distribution within a fixed radius around the computed mode and finally approximate the distribution by fitting a Gaussian to the weighted sample points.

4.3.2. Improved Proposals for Range and Vision Observations

For an improved accuracy while climbing stairs, we augment range measurements $\mathbf{r}_t$ with camera images $\mathcal{C}_t$ in one observation $\mathbf{o}_t$. By substituting accordingly in Eq. (4.3), we obtain

$$p(\mathbf{x}_t \mid m, \mathbf{x}_{t-1}^{[i]}, \mathbf{o}_t, \mathbf{u}_{t-1}) = \frac{1}{\eta^{[i]}}\, p(\mathbf{r}_t, \mathcal{C}_t \mid m, \mathbf{x}_t)\, p(\mathbf{x}_t \mid \mathbf{x}_{t-1}^{[i]}, \mathbf{u}_{t-1})\,. \tag{4.8}$$

We evaluate this distribution in points sampled in the x,y, and yaw dimensions. According to Eq. (4.7), this results in an approximation of $\eta^{[i]}$:

$$\begin{aligned}\eta^{[i]} &:= p(\mathbf{r}_t, \mathcal{C}_t \mid m, \mathbf{x}_{t-1}^{[i]}, \mathbf{u}_{t-1}) \\ &\simeq \sum_{j=1}^{K} \Big(p(\mathbf{r}_t, \mathcal{C}_t \mid m, \mathbf{x}_j^{[i]})\, p(\mathbf{x}_j^{[i]} \mid \mathbf{x}_{t-1}^{[i]}, \mathbf{u}_{t-1}) \Big)\,.\end{aligned} \tag{4.9}$$

Based on MCL introduced in the previous chapter, the improved proposals MCL algorithm performs the following steps for each of the particles:

1. Transform the pose of the particle according to the motion model. Starting from this initial pose estimate, a scan matcher improves the particle pose by fitting the current range measurement in the given 3D environment model. The scan matcher uses a gradient descent approach and RPROP (Riedmiller and Braun, 1993) for fast convergence. This step is necessary because the proposal distribution can only be evaluated for a limited number of sample points, which must be chosen carefully so that they cover the meaningful area of the proposal distribution. Analogous to Grisetti et al. (2007), we assume that the target distribution typically has only one mode and that the proposal distribution can be approximated by sampling within a fixed radius around the mode returned by the scan matcher.

2. The algorithm needs to estimate the improved proposal distribution from which it draws the new particle poses. As the most likely pose returned by the scan matcher already provides a good estimate of the robot's pose, we assume that the meaningful regions of the proposal distribution will be in the vicinity. Hence, we sample $\{\mathbf{x}_1^{[i]}, \ldots, \mathbf{x}_K^{[i]}\}$ from a uniform distribution within a fixed range around the pose returned by the scan matcher. The sampling range must be chosen large enough to cover the meaningful area of the target distribution, in our case ± 5 cm in x and y direction, and $\pm 4^\circ$ for the yaw.

3. At each point $\mathbf{x}_j^{[i]}$, the algorithm evaluates the observation likelihood $p(\mathbf{r}_t, \mathcal{C}_t \mid m, \mathbf{x}_j^{[i]})$ based on the current range measurement $\mathbf{r}_t$ and the set of camera images $\mathcal{C}_t$. This evaluation of the proposal distribution can only be performed at sample points. We assume that the range measurement and vision measurements are conditionally independent, which results in

$$p(\mathbf{r}_t, C_t \mid m, \mathbf{x}_j^{[i]}) = p(\mathbf{r}_t \mid m, \mathbf{x}_j^{[i]})\; p(\mathcal{C}_t \mid m, \mathbf{x}_j^{[i]})\,. \tag{4.10}$$

   We obtain the range observations $p(\mathbf{r}_t \mid m, \mathbf{x}_j^{[i]})$ according to the ray casting model (Section 3.3.2) and the vision observations $p(\mathcal{C}_t \mid m, \mathbf{x}_j^{[i]})$ according to chamfer matching, described in detail in the next section.

4. From the sample values of the proposal, we recover a continuous distribution by fitting a multivariate Gaussian distribution. From this distribution the algorithm then draws the new particle poses in the resampling step.

5. Finally, the importance weights of the newly sampled particles are computed according to

$$w_t^{[i]} \propto w_{t-1}^{[i]}\, \eta^{[i]}\; p(\tilde{z}_t \mid m, \mathbf{x}_t^{[i]})\; p(\widetilde{\varphi}_t \mid \mathbf{x}_t^{[i]})\; p(\widetilde{\theta}_t \mid \mathbf{x}_t^{[i]})\,, \tag{4.11}$$

   where $\widetilde{\varphi}_t$ and $\widetilde{\theta}_t$ are the current roll and pitch angles from the inertial measurement unit in the robot's chest, $\widetilde{h}_t$ is the torso height estimated from the joint encoder values, and $\eta^{[i]}$ is the sum defined in Eq. (4.9) for particle i. The individual observation models are defined in Eq. (3.20)–(3.22).

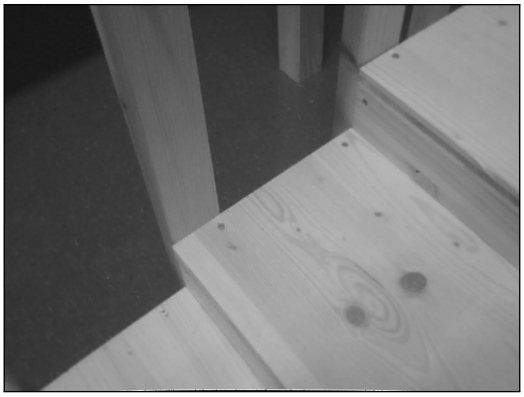
4.3.3. Observation Model for Vision Data

In order to integrate vision observations into the particle filter, we need to define an observation model $p(\mathcal{C}_t \mid m, \mathbf{x}_t)$. This new observation model defines the likelihood of capturing the scene in a set of images $\mathcal{C}_t = \{c_{t,1}, c_{t,2}, \dots\}$ given a 3D edge model of the environment m and the estimated pose $\mathbf{x}_t$ of the robot. We assume that the individual image observations are conditionally independent

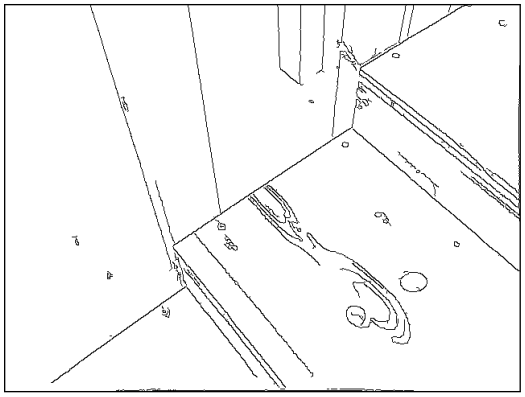
$$p(\mathcal{C}_t \mid m, \mathbf{x}_t) = \prod_{c_t \in \mathcal{C}_t} p(c_t \mid m, \mathbf{x}_t)\,, \tag{4.12}$$

which is a reasonable assumption when there is only little or no overlap between them.

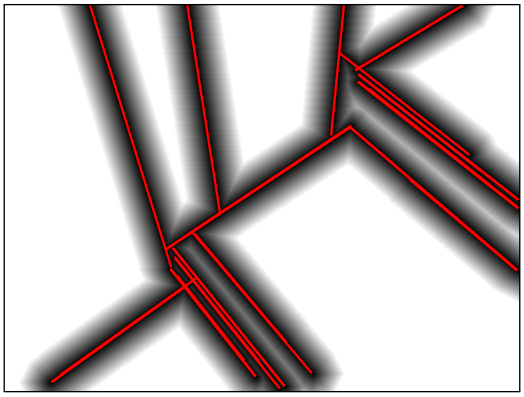
Our approach for estimating the observation likelihood is based on chamfer matching (Barrow et al., 1977) and relies on a consistent matching of the given edge model of the staircase to lines detected in the camera images. The detection process is illustrated for an example camera image in Figure 4.8. First, the algorithm applies the Canny edge detection algorithm (Canny, 1986) and a probabilistic Hough transform (Matas et al., 2000) in order to extract straight line segments. Afterwards, a distance transformation is applied



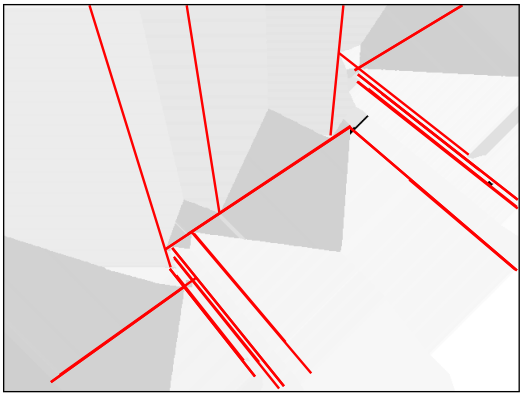
(a) Raw camera image.



(b) Canny edge detection algorithm applied to camera image.



(c) Distance transform: Gray values represent the Euclidean distance to the nearest line.



(d) Orientation transform: Gray values represent the orientation of the nearest line.

Figure 4.8.: Individual steps of the edge detection for chamfer matching: Starting with the raw camera image (a), the Canny algorithm marks edges (b) and the Hough filter extracts straight line segments shown as red lines in (c)–(d). Convolving these lines yield the distance (c) and orientation (d) transformation arrays that are used for matching edges in the observation model.

to the detected lines so that the value of each pixel indicates the Euclidean distance between the pixel and the nearest detected line. In a similar way, we compute an orientation transformation that maps each pixel to the orientation of the nearest detected line.

The algorithm then proceeds by computing the pose of the camera sensor by means of the kinematic chain from the robot's torso to the sensor and assuming that the robot's torso is located at pose $\mathbf{x}_t$. Then, the edges of the given staircase model are projected onto the hypothetical camera image, clipping them to the image rectangle if necessary. To compute the cost function for the matching, the algorithm iterates over all visible pixels $\mathbf{q} = (u, v)$ of the model edges l in the projected image:

$$cost = \sum_{l \in m} \sum_{\mathbf{q} \in l} \left(\alpha \, \mathrm{dist}(\mathbf{q}, \mathrm{d}(\mathbf{q})) + \beta \angle(\mathrm{d}(\mathbf{q}), l) \right). \tag{4.13}$$

Here, $\mathrm{d}(\mathbf{q})$ denotes the detected line nearest to pixel $\mathbf{q}$, $\mathrm{dist}(\mathbf{q}, \mathrm{d}(\mathbf{q}))$ denotes the Euclidean distance between $\mathbf{q}$ and the nearest detected line from the computed distance transform, and $\angle(\mathrm{d}(\mathbf{q}), l)$ is the absolute angle between the nearest detected line and the projected model line from the orientation transform. α and β are constant weighting factors to trade off distance against orientational costs. In the ideal case, all projected model edges are covered by detected lines in the actual camera image, which results in a cost value of 0. This proposed cost function is robust to additional detected edges in the camera image, e.g., due to noise or different textures because the algorithm only iterates over model edges in the map. For certain viewing angles, however, the robot's body occludes parts of the camera image. In these cases, the likelihoods could be wrongly estimated due to increased costs from the occlusion. To solve this problem, we mask out all parts of model edges that are occluded in the projection based on the robot's 3D mesh model and its current body pose estimation.

Since the costs in Eq. (4.13) are within the semi-infinite interval $[0, \infty)$, we model the observation likelihood with an exponential distribution, resulting in

$$p(c_t \mid m, \mathbf{x}_t) = \lambda \, \exp(-\lambda \, cost). \tag{4.14}$$

The distribution parameter λ was determined experimentally.

4.4. Evaluation

We evaluated all steps from perceiving a staircase in range data to ascending the staircase based on an improved localization estimate. As experimental platform, the laser-equipped Nao humanoid described in Appendix A was used.

4.4.1. Stair Reconstruction Results

First, we evaluated the segmentation of planes from the 3D scan data acquired by the Nao. As discussed before, the input data contains both random noise and systematic errors caused by the measurement process. This noise equally affects the performance of all algorithms.

For our experimental comparison, we used datasets acquired while the robot was standing at a distance of 70 cm in front of the staircase, and while standing on the first, third, fifth, and seventh step. Each dataset for a step consist of approximately 132 000 points.

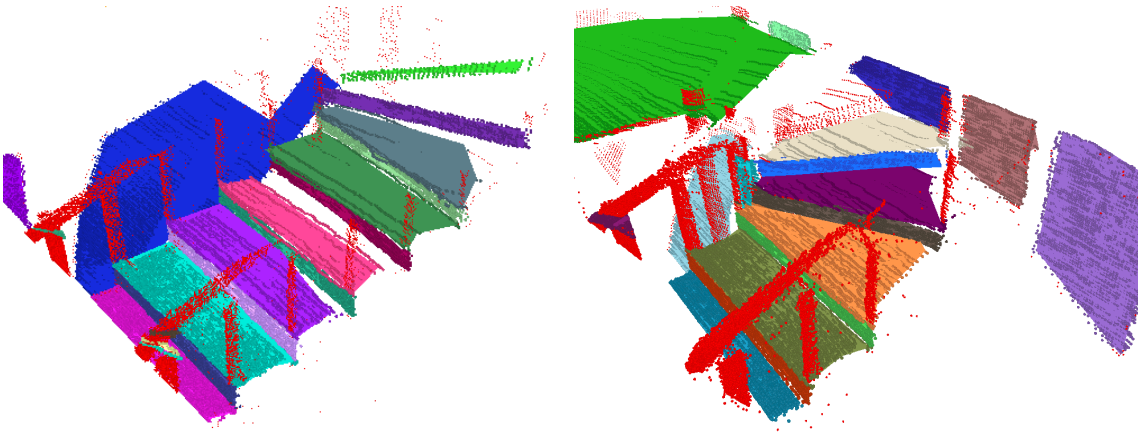
We found that scan-line grouping as well as the improved version of the two-point algorithm reliably extract most plane segments of the staircase within the robot's field of view (see Figure 4.9). Since scan-line grouping can find planes with arbitrary orientations, the method in general extracts more plane segments than the two-point algorithm. In most cases, the method also precisely finds the edges of planes due to the refinements carried out in the post-processing step.

On the other hand, the two-point algorithm is also able to detect smaller plane segments as long as they are aligned to one of the main directions. Only small plane segments not parallel to any other planes generate sparse rings on the sphere of the unit difference vectors, making such planes hard to distinguish from random noise. Accordingly, the two-point method is less likely to extract small planes with less frequent orientations such as the vertical faces of the winding part of our staircase (Figure 4.9d). However, these steps can still be reconstructed from the more reliable detection of the horizontal faces.

For a quantitative evaluation of the two segmentation algorithms, we measured the runtime on a standard desktop CPU (Intel Core 2 Duo, 3 GHz) and compared the reconstructed dimensions of the steps to the measured ground truth values of the staircase. These consist of the side lengths of the steps and the angles between adjacent faces. Table 4.1 shows a summary of the results in terms of average error and standard deviation. The two-point random sampling algorithm aligns all planes with respect to a small number of common main directions, thus the angular deviation between parallel planes tends to be lower than in models generated by scan-line grouping. Both methods lead to stair models that are accurate enough to be climbed autonomously. Regarding computational complexity, the scan-line grouping algorithm is significantly more efficient as it mainly operates on line segments extracted from the range image, whereas sampling-based methods operate on the whole point cloud and need a large number of iterations to find all plane segments with sufficiently high probability.

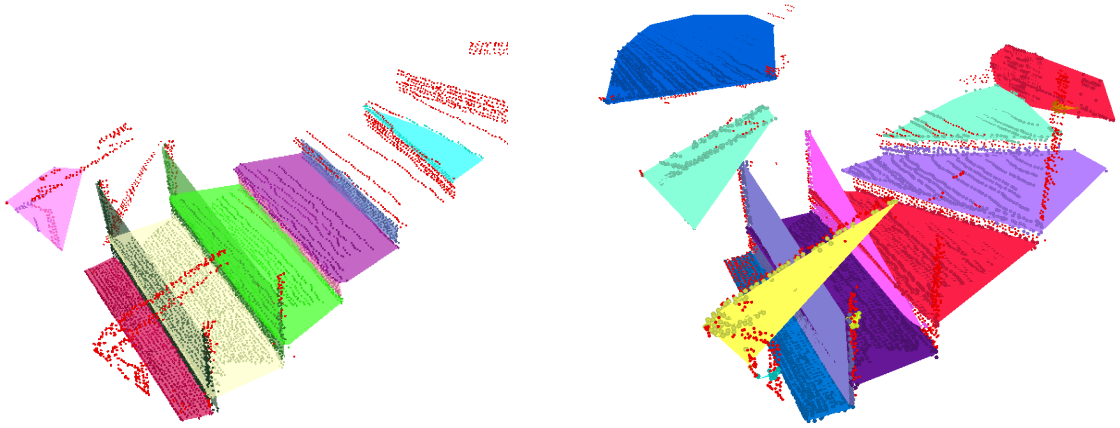
However, it is worth mentioning that two-point random sampling can also be used to extract plane segments from accumulated point clouds obtained from multiple viewpoints whereas scan-line grouping only operates on one range image from one viewpoint at a time.

In order to climb the complete staircase, the robot requires a reconstructed 3D model of the whole staircase. This is obtained by merging the reconstruction results from different view points. Whenever a new point cloud is segmented and its steps are merged into the



(a) Scan-line grouping, lower staircase

(b) Scan-line grouping, upper staircase



(c) Two-point random sampling, lower staircase

(d) Two-point random sampling, upper staircase

Figure 4.9.: Comparison of segmentation results with scan-line grouping (a-b) and two-point random sampling (c-d) for different parts of the staircase. For a clearer visualization, each segmented plane is displayed by the convex hull of the points in the plane with a different color. Remaining points not belonging to any plane are indicated in red.

		Scan-line Grouping	Two-point random sampling
Modeling error [cm]	Step height (7 cm)	0.42 ± 0.31	0.68 ± 0.54
	Step width (60 cm)	3.40 ± 1.95	2.25 ± 1.97
	Step depth (18 cm)	1.17 ± 0.67	0.90 ± 0.61
Angular error [°]	Parallel planes	2.22 ± 2.17	1.14 ± 1.13
	90° angles	4.97 ± 2.13	3.12 ± 1.47
Runtime [s]		0.025 ± 0.001	3.102 ± 1.043

Table 4.1.: Comparison of error and runtime (mean and standard deviation) between scan-line grouping and two-point random sampling

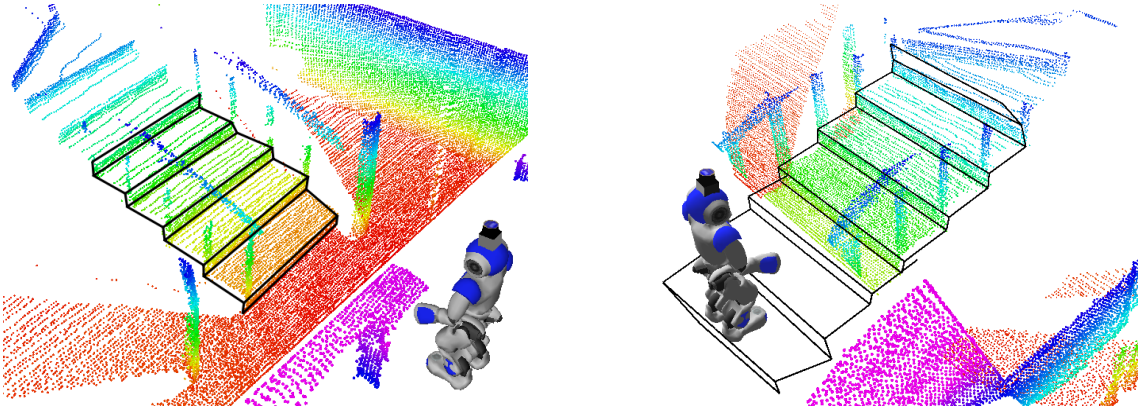


Figure 4.10.: Staircase model reconstructed from a 3D point cloud after the first scan (left), and combined from a subsequent scan on the first step (right).

model, the newly extracted steps replace existing ones that have the same height level. The result of such a process can be seen in Figure 4.10.

4.4.2. Improved Proposals Localization for Stair Climbing

We evaluated the performance of our extended localization approach that also integrates monocular vision data in the particle filter. For this evaluation, the Nao humanoid had to ascend all ten steps of the winding staircase shown in Figure 3.5. In addition to the volumetric 3D map for range data integration, the robot here made use of the known edge model of the staircase for image matching. For the purpose of this evaluation, we provided the staircase edge model as a manually designed CAD model. Each step of the staircase is 7 cm high. Based on the localization estimate, the robot aligns with the step in front of it and then proceeds to climb the step with the learned whole-body motion, as described in Section 4.2. Once the humanoid reaches the top of the step, it integrates a sensor measurement for pose estimation and then corrects for motion drift by aligning

with the next step. For integrating vision data into our improved proposals approach, the robot assumes an upright posture and acquires three images with its bottom-facing camera, looking to the left, center, and right. This combination allows the humanoid to capture most of the area in front of its feet, despite the narrow field of view of the internal camera.

Measurement Integration for a Single Step

Figure 4.11 shows an illustrating example of integrating vision data according to the method presented in Section 4.3. Here, the robot was standing on the second step and looked to the left, center, and right. Figure 4.12 shows the resulting observation likelihoods in the horizontal plane for these camera observations and Figure 4.13 (left) shows the likelihood of the corresponding laser observation with ray casting. The coordinate system is identical for all distributions and is centered at the odometry pose. The distributions of the left and right camera image are both focused and have similar modes while the distribution of the center camera image is spread in horizontal direction due to high uncertainty in lateral direction. The final improved proposal and fitted Gaussian distribution resulting from combining the distributions for vision and laser is shown in Figure 4.13 (right). As can be seen, the pose estimate is highly focused. Figure 4.11 shows the edge model projected from the most likely estimated pose after localizing on this step. The projected lines on the left and right camera image closely fit the corresponding edges of the staircase; the errors in lateral direction and in the orientation are only small. The error in forward direction is slightly higher, which is caused by the strong lines in the wood texture parallel to the stair edge.

Localization Accuracy

As a performance measure for the quality of the localization, we evaluated the pose returned by the localization approaches with and without integrating visual observations and compared it to the ground truth from the motion capture system on each single step of the staircase. Here, we aggregated the results over the number of successful steps the robot climbed in six runs ($N = 23$ for laser-only localization, $N = 60$ for laser with vision).

The mean and standard deviations of the error between the estimated pose and the ground truth from the motion capture system are displayed in Table 4.2. Our combined approach using improved proposals with chamfer matching decreased the translational error from 2.6 cm to 1.1 cm and the rotational error from 1.3° to 0.6°. Both improvements are statistically significant (t-test, 95% confidence for translation, 99.9% for rotation). In many cases, the accuracy of our approach is higher than the map resolution used for laser-based MCL (1 cm), which additionally demonstrates the advantages of integrating visual information.

Using only laser-based MCL, the robot succeeded in climbing all ten steps of the staircase in only one out of six runs and fell five times. During these six experiments,

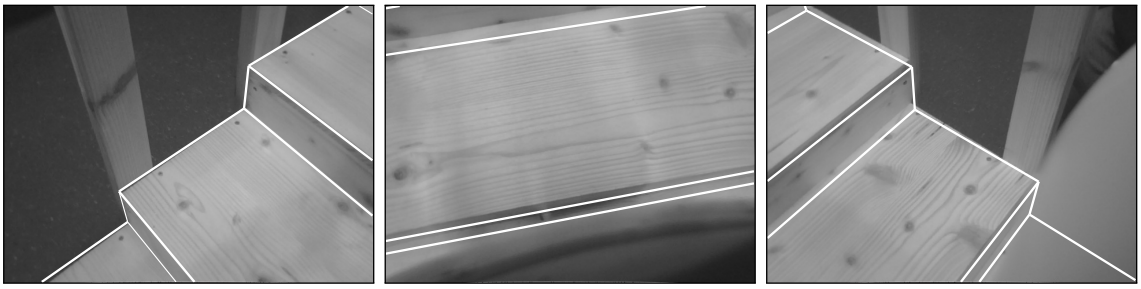


Figure 4.11.: Camera view of the robot when looking to the left, center, and right on the staircase. Overlaid in white is the projected stair model from the best particle pose after integrating all observations. The matching model demonstrates the high accuracy of the localization.

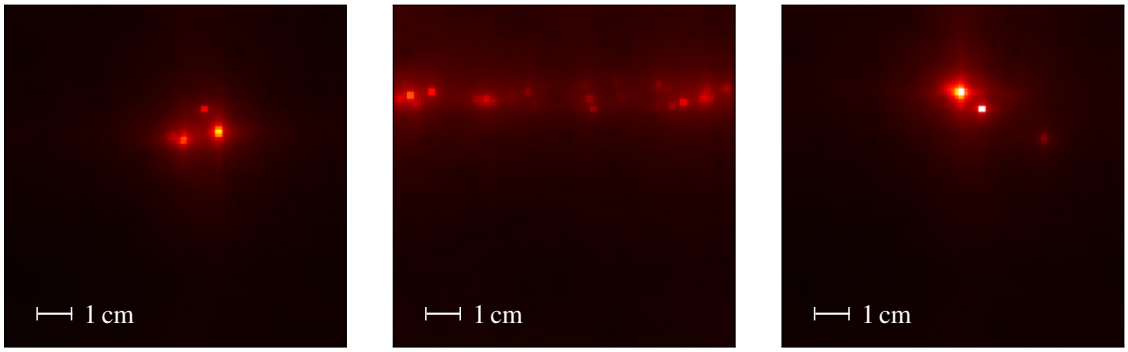


Figure 4.12.: Observation likelihood $p(c_t \mid m, \mathbf{x}_j)$ for left, center, and right camera image of Figure 4.11 as returned by chamfer matching. The likelihood is shown as a projection in the horizontal plane, whereas brighter areas correspond to a higher likelihood.

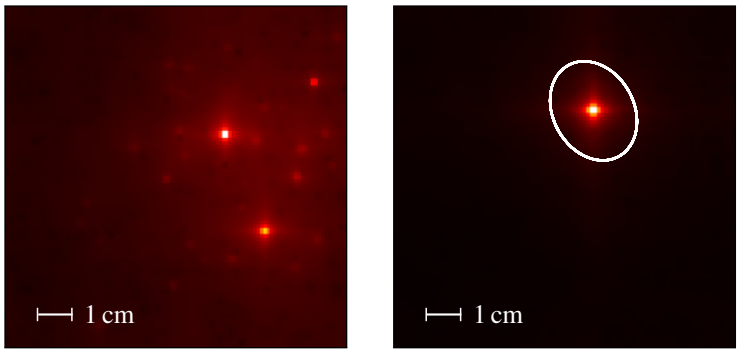


Figure 4.13.: *Left*: Observation likelihood $p(\mathbf{r}_t \mid m, \mathbf{x}_j)$ of the laser data. *Right*: Final improved proposal distribution by combining laser and vision observations (Figure 4.12) with the 95% confidence ellipse of the Gaussian approximation (white). Brighter areas correspond to a higher likelihood.

Error	Laser only	Improved proposals w. chamfer matching
Translation [cm]	2.6 ± 0.6	1.1 ± 0.1
Yaw [°]	1.3 ± 0.4	0.6 ± 0.2

Table 4.2.: Mean and 95% confidence interval of the error between estimated pose and ground truth after localizing on each step.

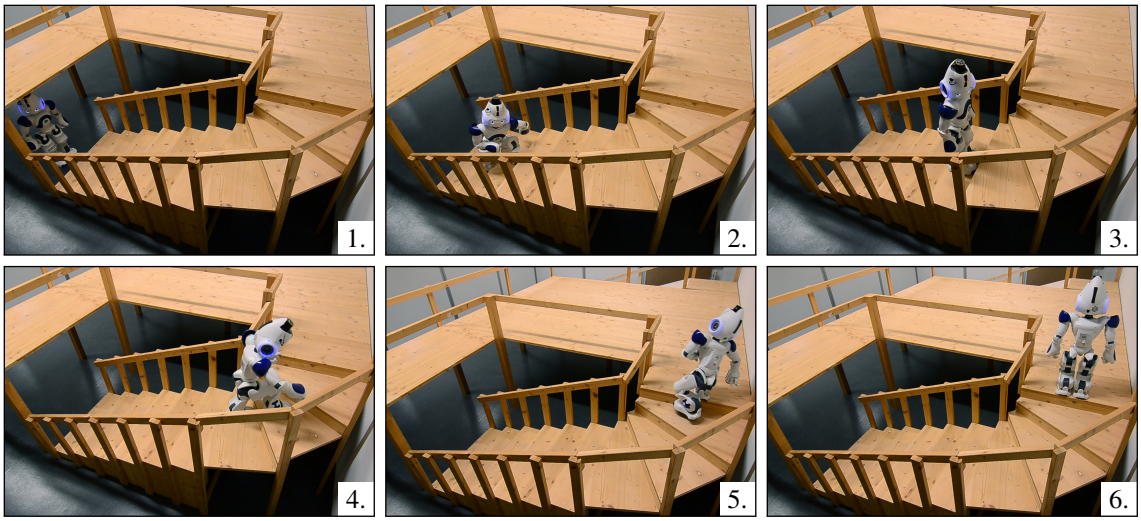


Figure 4.14.: Nao humanoid climbing a winding staircase using our approach of combining vision and laser range data for localization to correct for motion noise.

the robot successfully climbed 23 steps in total. This corresponds to a success rate of 82% for climbing a single step. In contrast to that, our improved proposals localization lead to a more accurate pose estimation. The robot always positioned itself precisely in front of the next step and reliably executed the stair climbing motion. Over the course of ten experiments, the robot had a success rate of 100%, i.e., it successfully climbed the whole staircase in all ten runs. Figure 4.14 shows the robot ascending the winding staircase using our approach. A video of the complete sequence is available online at http://www.youtube.com/watch?v=U9ll8y3Svkw.

Our approach is robust against unmodeled objects and texture on the surfaces, as the vision observation model only tests if the predicted edges are present in the camera images. The robot can detect self-occlusions and exclude the affected regions from the observation model. However, the localization performance may degrade if other unmodeled objects occlude significant parts of the predicted model edges, or if the texture contains strong lines running parallel to the modeled edges.

Run Time Evaluation

The run times (mean and 95% confidence interval) of integrating measurements on each step of the staircase are 0.078 s ± 0.07 s for laser-only localization and 18.8 s ± 0.2 s for improved proposals with additional vision data. Sensor measurements are processed and integrated on an external Desktop PC (Intel Core 2 Duo, 3 GHz), communicating with the Nao over network. The improved proposals algorithm needs significantly more time because the observation model is computationally expensive and the robot has to turn its head slowly to acquire images from three different camera perspectives without motion blur. Sensor measurements are already processed while the robot turns its head to record the next image, and the run time includes capturing and transmitting the camera images over network (12.9 s ± 0.2 s). All sensor data integrations only have to be done once per step. As a comparison, executing the complete whole-body motion of climbing one step requires 17 seconds with the Nao.

4.5. Related Work

Several approaches in the past did not use a 3D model at all for traversing single steps or small staircases. For example, Nishiwaki et al. (2002) constructed toe joints for the humanoid H7. These joints enabled a wider range of motions and more efficient torque usage in the knee joints, both of which are useful for climbing stairs. The robot was consequently able to climb single steps with 25 cm height after a manual positioning in front of them. Honda's ASIMO executed a fixed sequence of footsteps that was locally adapted based on data from force sensors (Hirai et al., 1998). Such methods are mostly reactive and unsuited to traverse complex staircases spanning over multiple steps or even containing winding parts like in our scenario. In contrast to previous work that used specialized hardware for the stair climbing task such as 6D force/torque sensors or flexible toe joints, we rely on a standard, low-cost humanoid robot platform.

For localization, Michel et al. (2007) proposed to visually track single objects that are in the camera's field of view. Their approach requires a known 3D model, to which the detected edges are matched in a GPU-accelerated process. After a manual pose initialization, the robot can determine its pose relative to the tracked object while walking. Their HRP-2 humanoid localized itself with respect so a staircase of three steps, which it subsequently ascended. Cupec et al. (2005) developed a vision-based approach to track obstacles. Their obstacle detection relies on the assumption that obstacles and floor are clearly distinguishable in the images to simplify edge detection. The authors presented experiments in which the humanoid robot Johnnie stepped over small obstacles and climbed up two steps.

Okada et al. (2001) proposed a technique to reconstruct single steps from 3D stereo data. The authors used a randomized 3D Hough transform to extract planes. Their humanoid autonomously climbed a single step. In a later work (Okada et al., 2005), the au-

thors extended the step detection with a local 3D footstep planner. Gutmann et al. (2004; 2008) use the efficient scan-line grouping algorithm discussed in this chapter to extract models of steps given stereo data. Using such models, the humanoid QRIO was able to climb up and down staircases consisting of four steps. Kida et al. (2004) introduced the two-point random sampling algorithm which we extended and applied in this chapter. From stereo vision data, the authors were able to segment the ground plane and detect three single steps of a staircase.

Chestnutt et al. (2009) applied the two-point random sampling approach to identify planes corresponding to steps or flat obstacles based on 3D laser point cloud data. To speed up the process, they restricted the detection to planes $\pm 30^{\circ}$ from horizontal and used lookup templates specific to the environment when sampling. Their humanoid robot equipped with a pivoting laser scanner was able to climb up and down two successive steps of a small staircase.

Furthermore, there are more general approaches for segmenting 3D range data. Hähnel et al. (2003) apply a region-growing technique to identify planes in 3D point clouds. The main focus was here to generate compact models of buildings. EM-based procedures to extract planes from 3D range data as proposed, for example, by Thrun et al. (2003) and Triebel et al. (2005) are rather time-consuming since the data points need to be processed multiple times during the estimation of the model components and also for the estimation of the number of components.

Rusu et al. (2009) proposed to divide the point cloud into different cells, apply RANSAC to find polygons locally, and merge them afterwards. The resulting polygonal models were used for navigation with the ground-based RHex robot. RANSAC, however, tends to over-simplify complex planar structures in our experience. Multiple small steps are often merged into one sloped plane which is inappropriate for safely climbing stairs with a humanoid. Klasing et al. (2008) presented an approach for clustering surfaces from 3D point clouds, which they later extended to realtime segmentation from a continuous stream of sensor data (Klasing et al., 2009). Their approach first estimates the surface normals from a local point neighborhood and then clusters the points based on their distance and normal direction. While the segmentation results are impressive, they are quite sensitive to a carefully tuned set of parameters as stated by the authors.

The recent work of Ben-Tzvi et al. (2010) follows a different approach. The authors first generate a depth image from the 3D point cloud and reconstruct contours of objects. Afterwards, they segment planes based on these contours. The technique works well for scenes with a low number of planes, however, it is unclear how it scales to more complex scenes.

4.6. Conclusion

In this chapter, we first provided methods for humanoid robots with 3D sensing capabilities to perceive staircases in complex environments. We compared and extended two

plane segmentation approaches, both leading to accurate 3D models despite noisy sensor data, a high number of faces, and clutter in the environment. The learned stair model is slightly more accurate with two-point random sampling, making this approach more useful in practice. Based on this learned model, a Nao humanoid was able to climb a challenging staircase with ten steps and a winding part using whole-body motions. These motions were learned through kinesthetic teaching from human demonstrations.

We furthermore presented an extension to the 3D localization approach from Chapter 3. Our extension is based on informed proposals in the particle filter and yields highly accurate pose estimates, which is required for the critical task of climbing stairs. Our approach uses monocular vision in addition to range data to estimate the robot's 6D pose in a given 3D model of the environment. We developed a new observation model based on globally consistent chamfer matching between the previously learned edge model of the 3D world and lines detected in camera images. In our evaluation with a Nao humanoid, we showed that our approach leads to a highly accurate localization of the robot and thus to robust navigation capabilities. The presented method based on improved proposals and chamfer matching can be generally applied to robot localization whenever lines detected in images can be used to refine an initial pose estimate. For example, current RGBD-sensors based on active illumination such as the Microsoft Kinect or Asus Xtion yield no depth data at close ranges. Localization with these sensors could be enhanced with our approach by relying on edge detection whenever there is no range data available.

Chapter 5

Footstep Planning

In this chapter, we present a framework for search-based footstep planning in cluttered environments and evaluate different planning approaches. We first introduce and compare the anytime planners ARA and R*, which return plans with provable suboptimality fast and improve them as time allows. Since a robot often has to replan in a real-world setting, we then present efficient incremental replanning using AD*. We furthermore provide an approach to efficiently plan longer trajectories. Our adaptive level-of-detail planning approach uses fast 2D planning in open spaces and detailed footstep planning close to obstacles, which results in efficient planning without sacrificing the solution quality. We evaluated the presented planning approaches in different environments and furthermore demonstrated the applicability to humanoid navigation with a real Nao humanoid.*

Compared to wheeled robots, the greater flexibility of humanoid robots allows for unique capabilities and a human-like locomotion. This not only enables humanoids to climb stairs as discussed in the previous chapter, but also to step over or onto various other objects that pose obstacles to wheeled robots. However, the human-like body plan of humanoid robots requires a high number of degrees of freedom to be controlled, which makes planning and control a demanding challenge. This is mostly due to the high complexity of motion planning, which is a PSPACE-complete problem in the general case (Canny, 1988; Reif, 1979).

Whole-body planning is not yet practical for navigation over long distances (Hauser et al., 2005, 2007). In the next chapter, we will consider whole-body planning for manipulation instead. For humanoid navigation, however, planning can be performed for a sequence of collision-free footsteps, as originally proposed by Kuffner et al. (2001). From this sequence, a trajectory of the desired zero moment point (Vukobratovic and Borovac, 2004) can be generated that represents a dynamically stable motion. Finally, a pattern generator computes the trajectory for the center of mass (Kajita et al., 2003) and thus a sequence of joint angles to perform a stable walking motion on the initially planned footsteps. Many locomotion approaches for humanoids adhere to this layered architecture,

which enables us to rely on existing walking controllers for the execution and thus plan for humanoid navigation in the lower-dimensional space of footsteps.

Previous methods for footstep planning are either based on A* search (Chestnutt et al., 2003, 2005, 2007) or randomized planners (Liu et al., 2012; Perrin et al., 2012a). While A* search will find the optimal path for a given footstep parameterization and state discretization, its planning performance highly depends on the quality of the heuristic function that guides the search. In particular, obstacles posing local minima in the search space can force A* to expand many states during the search. Randomized methods find results fast but do not have any guarantees on the solution quality of the final path, which may be far from optimal. Furthermore, they often depend on smoothing the final path in a post-processing step (Hauser et al., 2005). Randomized planners are probabilistically complete, which means they will find a solution if it exists with a probability approaching 1 as the running time approaches infinity (Kuffner and Lavalle, 2000). A* search, on the other hand, is guaranteed to terminate with a solution if it exists and will return a failure otherwise.

In this chapter, we introduce and compare several search-based approaches for footstep planning. These methods are all based on a variant of the A* search and thus yield goal-directed paths suitable for navigation that can be cost-optimal in the best case. While it is desired to fully plan the path from start to goal to decide on its feasibility, it is often not necessary to initially find the optimal path since it can be further improved while the robot is walking towards the goal. To achieve fast planning results, we use anytime planners that have a guarantee on their solution suboptimality. The resulting path costs are guaranteed to be no more than a factor w higher than the optimal result from A*. This behavior will be particularly beneficial when obstacles pose local minima in the search space, which is often the case when planning footsteps that can pass over obstacles.

We first adapt the Anytime Repairing A* (ARA*) algorithm for footstep planning and evaluate various heuristics for the planner. ARA* was originally proposed by Likhachev et al. (2004) and runs a series of weighted A* (wA*) searches, thereby efficiently reusing previous information. wA* inflates the heuristic of A* by a factor w and guarantees a solution cost suboptimality of at most w.

As a second planner, which depends less on the heuristic function, we propose to use R* (Likhachev and Stentz, 2008), an A* search variant that combines wA* in a local deterministic search with a randomized component, thereby trying to speed up the search. In contrast to purely randomized approaches such as rapidly-exploring random trees, R* provides probabilistic guarantees on the sub-optimality of the solution.

While executing a plan in the real world, a humanoid might deviate from its initially planned path due to joint backlash and foot slippage. Furthermore, the environment is often not static and obstacles can appear or disappear. This forces the robot to regularly replan during execution. Instead of planning from scratch, we propose to employ the Anytime Dynamic A* algorithm (Likhachev et al., 2005). This algorithm combines the advantages of ARA* with D* Lite, thus enabling efficient replanning.

When planning long distances for navigation, the robot may encounter large open spaces where no obstacles are present. In these areas, a 2D plan is often sufficient and more efficient to plan than footsteps. To efficiently plan for long distances, we propose an adaptive level-of-detail planning approach which combines coarse global path planning with detailed motion planning in areas requiring complex navigation capabilities. Compared to only using the coarse planner, we can find more efficient paths since the difficult regions can be passed instead of choosing detours around them. Compared to using only the detailed footstep planner, we can significantly reduce the computation time using our approach.

We present our extensions to search-based footstep planning and provide a comprehensive comparison of the planners to existing approaches. We thoroughly discuss the performance for different situations and heuristic functions. As the experiments demonstrate, we are able to plan even long footstep paths in short planning times. Due to search-based planning, the paths are directed towards the goal and have provable suboptimality bounds. Our implementation is available open source and is based on the Search-Based Planning Library (Likhachev, 2010). For the scope of this work and the purpose of evaluating the applicability of the planning approaches for humanoids, we assume a two-dimensional environment representation. Obstacles are thus planar and the robot can usually step over them.

5.1. Planning Framework

As a basis for all search methods, we first describe the basic principles underlying our footstep planning framework.

5.1.1. States, Transition Model, and Lattice Graph

Assuming a planar environment model, the robot's state for the planner is described by the global position and orientation of its stance foot $s = (x, y, \theta, l)$, where (x, y) denotes the position of the foot origin, θ its orientation, and $l \in \{\text{left}, \text{right}\}$ indicates the left or right leg. For walking, the robot has to alternate between the left and right stance leg. Thus, a transition between two states can be parameterized by the displacement of the moving foot relative to the stance foot, as illustrated in Figure 5.1. We denote this transition as a footstep action $a = (\Delta x, \Delta y, \Delta\theta)$ relative to the position of the stance foot. $\mathcal{A}$ denotes the set of all footstep transitions available to the robot, which is usually constrained by the range of motion of the legs. We hereby assume that $\mathcal{A}$ contains only the actions for one foot and the actions for the other side are symmetric.

When the footstep planner expands a state s, it determines the successor states s' by applying all available actions $a \in \mathcal{A}$. Depending on the leg of the stance foot, the planner mirrors the action in $\mathcal{A}$ accordingly. After determining all successor states s', the planner checks them for collisions with obstacles and discards invalid states.

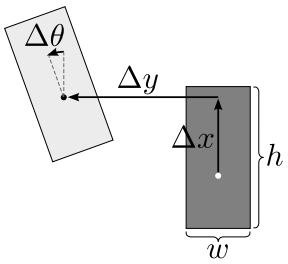


Figure 5.1.: Footstep transition model from the right stance foot to the left foot.

The costs of a state transition from s to s' are given by the transition costs $c(s,s')$. To obtain results that minimize the execution time to reach the goal, we model the costs to correspond to the execution time of one step. We hereby assume that there are constant costs for each step β that relate to the walking controller frequency, and costs that correspond to the distance covered by the step. Compared to that, the effort of changing the orientation can be neglected. Finally, a term dependent on the distance of the expanded state to the closest obstacle $D(x',y')$ makes the planner prefer paths further away from obstacles, while still being able to step close to obstacles when required. For $s = (x,y,\theta,l)$ and $s' = (x',y',\theta',l')$, this results in transition costs of

$$c(s,s') = \alpha \left\| (x,y)^\top - (x',y')^\top \right\| + \beta + \gamma D(x',y') \,. \tag{5.1}$$

With the weighting factors α, β, and γ, the different costs can be traded off with respect to the robotic platform and walking controller at hand, e.g., to prefer paths with fewer steps.

The iterative construction of states connected by footstep actions builds a sparsely-connected lattice graph. The footstep actions hereby correspond to the motion primitives used in search-based planning for wheeled-robots (Likhachev and Ferguson, 2009; Pivtoraiko and Kelly, 2005). One difference in footstep planning is that different footstep actions have to be applied to the left and right stance leg. In case only shallow obstacles in 2D are considered, the collision check only needs to be performed for the end foot location, as humanoids can step over these obstacles. In 3D, the humanoid's body needs to be tested for collisions over the complete foot trajectory instead.

The footstep set $\mathcal{A}$ is usually a discrete finite set of stepping motions derived from the kinematic range of the humanoid's leg, although there are also approaches that adaptively extend this set (Chestnutt et al., 2007). With this discrete set, the goal location — given as a pair of footsteps or a single state — may not be exactly reached or require too many intermediate steps to reach exactly. Thus, we dynamically extend the footstep set with a transition to the goal if it is reachable from the current state according to the humanoid's kinematic range. In essence, this creates an absorbing goal region around the actual goal state.

5.1.2. Environment Model and Collision Checking

As environment representation, we use a 2D grid map consisting of equally-sized grid cells that are marked as either free or occupied. In addition, occupied cells also contain traversability information, enabling the robot to differentiate between shallow obstacles that can be stepped over (such as uneven floor or clutter) and obstacles that have to be avoided with a larger clearance (such as walls or furniture).

In order to validate a possible footstep, the planner needs to check if the footstep collides with an obstacle in the environment. Because this validation needs to be performed for all successors of an expanded state, it should be as efficient as possible. We assume that the humanoid's footprint is rectangular, or can be approximated by a rectangular bounding box. By applying the method proposed by Sprunk et al. (2011), we can efficiently check rectangular shapes with arbitrary orientation against collisions. The method recursively compares the rectangle incircle and circumcircle against a pre-computed distance map of the obstacles and aborts if the foot is sufficiently away to be safe or close enough to definitely collide. Otherwise, the remaining parts are recursively subdivided.

5.1.3. Plan Execution

After planning a sequence of footsteps to reach its goal, a humanoid needs to execute this sequence by means of a walking controller, which computes the joint angle trajectories for the body. Small errors in the execution, e.g., due to joint backlash or a sliding of the foot as it touches the ground, can easily add up over multiple footsteps. In the end, this usually prevents the robot from executing footsteps open-loop. To correct the drift during execution, we require a planned sequence to consist of global states s_i instead of incremental footstep actions $a_i \in \mathcal{A}$ and additionally rely on the localization approach developed in Chapter 3 to provide a pose estimate. Before executing a step, we first compute the current position and orientation of the stance foot in the world based on the estimated pose of the torso frame. From this we compute the relative stepping action to the next footstep location in the world, which the robot then executes. By executing these computed, relative steps instead of the initially planned actions, the robot can correct small deviations as long as the computed footstep is within its range of motion. If the robot deviated too much from its initially planned path, the computed step is outside its stepping range and it needs to replan.

5.2. A* Search

On the lattice graph of all states, we can now apply search methods such as A* (Hart et al., 1968). The costs to transition between states are given from the transition model in Eq. (5.1) and only valid states remain after collision checking. In A* and related

methods, the search is guided towards the goal by a heuristic h that can include different kinds of information about the environment.

A* search expands states according to the evaluation function

$$f(s) = g(s) + h(s), \tag{5.2}$$

where $g(s)$ are the current costs of the best path from the start to the current state s and the heuristic $h(s)$ provides the estimated costs to the goal from state s. A* thus expands states first that are estimated to reach the goal with the least costs. To this end, the implementation uses a priority queue.

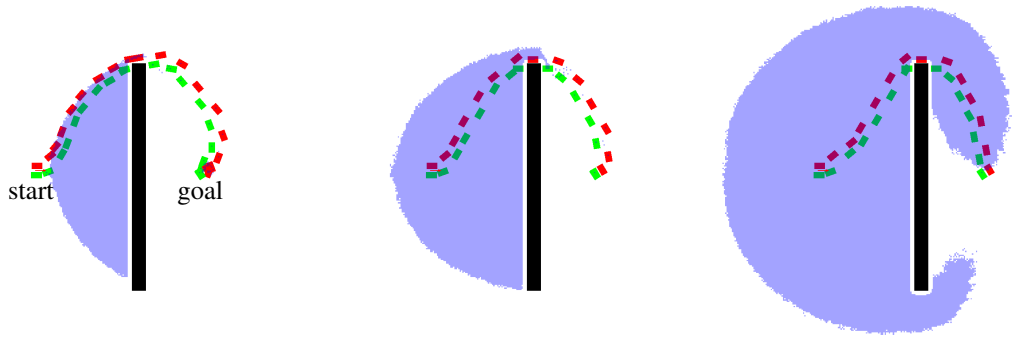
5.2.1. Heuristics

Usual requirements on the heuristic function h are that it is non-negative, admissible, and consistent for all states s, s':

$$h(s) \geq 0 \tag{5.3}$$
$$h(s) \leq c(s, s_{\text{goal}}) \tag{5.4}$$
$$h(s) \leq c(s, s') + h(s'), \tag{5.5}$$

where $c(s, s')$ denotes the true minimum costs of a traversal from state s to s'. A heuristic is admissible when it never overestimates the costs to reach the goal state s_{goal}. We may also use $h(s, s')$ to denote the heuristic costs between any two states. With a non-negative and admissible heuristic, A* is guaranteed to find the optimal path and expand no more states than any other optimal algorithm using the same heuristic (Hart et al., 1968). A consistent heuristic according to Eq. (5.5) is beneficial for an efficient implementation because it enables A* to expand each state at most once.

A common heuristic is the straight-line Euclidean distance to the goal, because it is admissible and easy to implement. At the same time, it is not much informed about the environment, particularly of any obstacles blocking the way. For footstep planning, an alternative heuristic can be obtained based on the costs of an optimal path in the lower-dimensional 2D grid space, usually pre-planned with A* or Dijkstra search. While this heuristic is more informed about the environment, it is potentially inadmissible because it does not reflect the humanoid's capability of stepping over obstacles (Chestnutt et al., 2005). Note that using A* for the 2D grid search will consider less states for a single search between a start and goal location, but Dijkstra search can be beneficial for pre-computing all heuristic values to a certain goal on a grid.



(a) $w = 10$ **(b)** $w = 2$ **(c)** $w = 1$ (A*)

Figure 5.2.: wA* footstep planning iterations around an obstacle for different heuristic inflation weights w with the Euclidean distance heuristic. Note that ARA* leads to identical paths and state expansions as wA*. Blue areas denote the expanded states. An inflated heuristic results in fewer expanded states and faster planning times at the cost of suboptimal paths.

5.2.2. Weighted A* Search

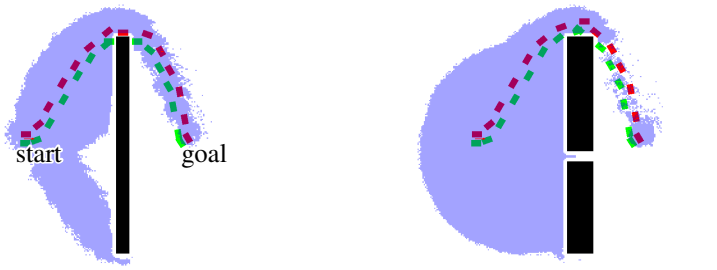
In our framework, the weighted A* (wA*) search forms the basis for the anytime algorithms ARA* and R*. wA* inflates the heuristic h with a factor $w \geq 1$, which means that the evaluation function becomes

$$f(s) = g(s) + w\,h(s)\,. \tag{5.6}$$

This makes the search more greedy towards the goal. While the resulting paths are suboptimal, they can be found faster and using less memory since fewer states are expanded. With the weight w, we can trade off the solution quality for planning efficiency. Furthermore, with an admissible heuristic $h(s)$, the suboptimality of the solution is bounded: The resulting paths are guaranteed to cost no more than w times the cost of an optimal path (Pearl, 1984). The actual path costs in practice are often much lower than the upper bound. For $w = 1$, the search uses the original heuristic values and thus corresponds to a regular A* search. As illustration, Figure 5.2 shows the footstep plans with wA* for different weights w around an obstacle.

5.3. Footstep Planning With ARA*

For navigation planning, it often takes much more time to execute a planned path than to plan it in the first place. It is also often desired to obtain a fast initial solution. With an anytime planner, we can obtain a fast initial solution that is suboptimal and then improve it as time allows. To this end, we rely on the Anytime Repairing A* (ARA*) search, which runs a series of wA* searches while efficiently reusing previous information (Likhachev et al., 2004). An initially large w causes the search to find a non-optimal initial solution



(a) initial $w = 10$, final $w = 1$ **(b)** initial $w = 10$, final $w = 1$

Figure 5.3.: ARA* planning around a local minimum with the 2D Dijkstra path as informed heuristic. In (a), planning succeeded after 1.8 s whereas in (b) the heuristic wrongly lead into a non-traversable narrow passage, requiring 15.0 s for the result and expanding seven times more states.

quickly. Then, as time allows, the search reruns with incrementally lower values for w. Given enough time, ARA* finally searches with $w = 1$ and finds the optimal path. If the planner runs out of time before, the cost of the best solution found by wA* is guaranteed to be no worse than w times the optimal solution cost. Iteratively running independent wA* searches would be computationally expensive, however, since much more states would have to be expanded than with regular A*. Instead, ARA* reuses the state information throughout the wA* searches and adapts their g-values for the evaluation function. In this way, ARA* expands a state in a new wA* iteration only if it became inconsistent due to the lowered weight w.

5.3.1. Heuristic Influence on ARA*

As any heuristic search, the efficiency of ARA* depends on the quality of the used heuristic. The more informed it is of the environment, the fewer states need to be expanded. With the admissible Euclidean heuristic, obstacles that block the straight path to the goal create local minima. As illustrated in Figure 5.2, these obstacles force the planner to expand large areas of the state space as w decreases and the optimal solution is sought. Note that, for identical values of w, the resulting paths and state expansions are identical for a single iteration of wA* and ARA*.

In contrast, a planner using the 2D Dijkstra path to the goal as heuristic is better informed about obstacles (Chestnutt et al., 2005). The planner then expands states on paths around the obstacles instead of creating local minima, as shown in Figure 5.3a. This directly results in more efficient planning times. However, the Dijkstra heuristic is potentially inadmissible and may even result in inefficient or no solutions at all in the case of footstep planning. Aside from that, it can even be susceptible to other types of local minima, which again result in a bad planner performance. As an example, consider Figure 5.3b with a narrow passage through the obstacle. The 2D Dijkstra heuristic will lead the planner to expand states through this passage even though it is not traversable with

Algorithm 2: Single iteration of R* according to Likhachev and Stentz (2008)

Input: Search graph Γ
Output: Γ with one state s expanded

```
select unexpanded state s ∈ Γ (priority to states not labeled as AVOID)
if path of edge bp(s) → s not computed yet then
    try to compute path bp(s) → s with wA*
    if failed then
        label s as AVOID
    else
        update g(s) based on cost of found path and g(bp(s))
        if g(s) > wh(s_start, s) then label s as AVOID
else                                                  // expand s by growing Γ
    let SUCCS(s) be k randomly chosen states at distance Δ from s
    if goal within distance Δ from s then add goal to SUCCS(s)
    foreach s' ∈ SUCCS(s) do
        add s' and edge s → s' to Γ
        bp(s') ← s                                    // set backpointer to predecessor s
```

stepping motions. Inflating the obstacles by the robot's circumcircle for computing the 2D heuristic path would prevent this specific problem, but essentially degrades the planning process to planning footsteps in the local neighborhood of a 2D path. In that case, however, the robot can no longer step through any kind of clutter which results in non-optimal paths. To keep this capability without giving up solutions, only the foot incircle can be used as the maximum obstacle inflation radius for the heuristic.

In case a location is surrounded by planar obstacles or clutter, there does not even exist a valid Dijkstra path as heuristic. Shallow steps present an example for this case, where each edge must be treated as an obstacle.

To summarize, although ARA* can be more efficient and better suited for footstep planning than A* due to its anytime properties, it is just like A* highly dependent on a well-designed heuristic function. Designing this heuristic becomes a challenging problem in itself for complex environments.

5.4. Footstep Planning With R*

The randomized A* search (R*) aims to be less susceptible to local minima without a strong dependency on the heuristic function. The search avoids local minima by running a series of short-range, fast wA* searches towards randomly chosen sub-goals in the state space. The exploration of the search space with random sub-goals relates to randomized

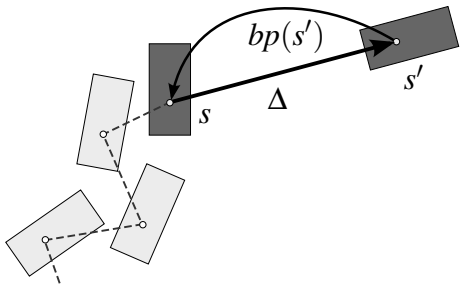


Figure 5.4.: R* expansion of a state $s \in \Gamma$ (dark gray) to which a local footstep path (dashed, light gray) already exists. k successors are determined in a random direction at distance Δ. If a successor s' is collision-free, it is added to Γ with the edge $s \rightarrow s'$ and the pointer to its predecessor $bp(s') = s$ is stored. At the next expansion of s', the search tries to find a local footstep path between s and s'.

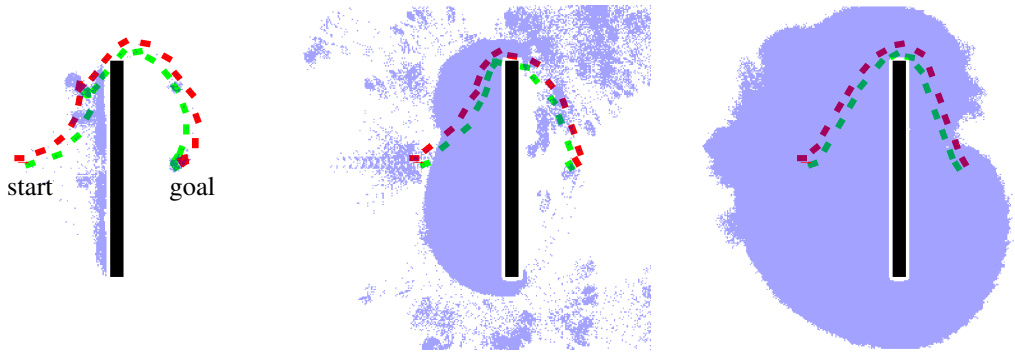
planners such as RRT. But contrary to them, R* aims to minimize the solution cost and provides probabilistic guarantees of the solution suboptimality (Likhachev and Stentz, 2008). In the context of navigation planning, R* is able to provide better solutions than RRT.

The R* algorithm, listed in Algorithm 2, iteratively constructs a graph Γ of sparsely placed states in the same state space as the dense search. At each iteration, the algorithm either computes a footstep plan for an existing edge in Γ (line 2–8) or generates further random successor states and edges in Γ (line 10–14). Hereby, k random states at a given distance Δ are added as well as any goal state within Δ.

Each edge in Γ corresponds to a path in the original, dense search graph. The algorithm plans these local footstep paths with wA*. R* hereby first tries to find all easy local paths. If a local path requires too many expansions, it is labeled as AVOID and the plan is postponed until it is required to be computed in order to meet the suboptimality bounds. During all expansions, R* maintains probabilistic guarantees on the suboptimality of the solution by the heuristic inflation factor w of the local wA* search. Finally, we obtain an anytime capability in R* by iteratively decreasing w throughout multiple runs, as long as there is still planning time remaining. Note that, compared to ARA*, R* currently offers no efficient plan re-usage over the single runs of wA*.

Generating the random nodes in Γ ensures the exploration of the search space. For footstep planning with humanoid robots, we randomly sample a direction and place a state s' at distance Δ from the current state, as illustrated in Figure 5.4. The leg variable of the state (left or right) is also chosen at random, whereas its orientation is given by the initial random direction to favor forward walking. If s' is collision-free, it is added to Γ along with the edge $s \rightarrow s'$.

Figure 5.5 shows an example run of R* using the admissible but uninformed Euclidean heuristic in the scenario with a local minimum that we have considered before. At the beginning, R* sparsely samples the state space towards the goal and around the obstacle. As the heuristic inflation w approaches 1, the state space expansion grows more dense in



(a) $w = 10$ **(b)** $w = 2$ **(c)** $w = 1$

Figure 5.5.: R* footstep planning iterations around an obstacle for different heuristic inflation weights w with the Euclidean distance heuristic. Blue areas denote the newly expanded states for each iteration.

order to find the optimal path. Only then the expansion resembles the wA* and ARA* expansions shown in Figure 5.2.

5.5. Anytime Incremental Replanning With AD*

The anytime planning approaches ARA* and R* described in the previous sections assume a static environment. Whenever parts of the environment change or the robot deviates from its initially planned path, these approaches need to plan a completely new path. This, however, is impractical for a robot navigating in the real world. Particularly in domestic environments there are humans in the vicinity of the robot, who pose dynamic obstacles. A robot is also unlikely to exactly execute an initially planned path since accumulated drift from foot slippage and joint backlash leads to deviations. All of this requires a robot to regularly re-plan.

Incremental replanning allows the robot to efficiently update an existing motion plan instead of planning from scratch. The most notable implementations are the algorithms Dynamic A* (D*) (Stentz, 1994) and D* Lite (Koenig and Likhachev, 2002), both of which work largely similar. The incremental search of D* Lite is hereby orders of magnitude more efficient than repeatedly planning a new path from scratch with A* (Koenig and Likhachev, 2002). In this section, we will first describe D* Lite for incremental replanning and then outline the Anytime D* (AD*) algorithm, which extends D* Lite with the anytime capabilities of ARA*.

5.5.1. D* Lite

For efficient replanning from changing starting states, D* Lite searches in the reverse order from the goal state s_{goal} to the current state s_{start}. This enables us to continually

update the start state based on new localization estimates, while most parts of the plan remain unchanged. In the case of footstep planning, s_{start} is the location of the current stance foot.

The initial search corresponds to a regular backwards A* search as there is no previous information available for replanning. Afterwards, the costs of all edges between states that have changed have to be updated. These changes can be the result of an updated start state or changes in the obstacle configuration of the environment. D* Lite maintains the estimated costs $g(s)$, whereas $g^*(s)$ denotes the costs of the optimal path from s to s_{goal}. Only when all $g(s)$ are consistent with $g^*(s)$, the optimal path based on the updated information can be extracted. Thus, upon every change of cost values, $g(s)$ needs to be made consistent with $g^*(s)$ again. Instead of recomputing all $g(s)$ on every change, D* Lite changes only relevant states. To this end, the algorithm uses a one-step cost lookahead

$$rhs(s) = \begin{cases} 0 & \text{if } s = s_{\text{goal}} \\ \min\limits_{s' \in SUCCS(s)} (c(s,s') + g(s')) & \text{otherwise,} \end{cases} \tag{5.7}$$

where $SUCCS(s)$ denotes the successor states of s by applying all valid state transitions. We can now detect s to be consistent if $g(s) = rhs(s)$ holds.

When a state is detected to be inconsistent, it is inserted into the priority queue of D* Lite. When expanding a state from the queue, D* Lite updates its cost values and thus makes it consistent again. To focus the state expansion, D* Lite uses a combination of the heuristic h and rhs to order its priority queue.

As for A* search, the heuristic for D* Lite needs to be non-negative, admissible, and consistent. The requirement for consistency, however, is stricter than Eq. (5.5) with $h(s) = h(s, s_{\text{goal}})$ by considering the heuristic values between all states s, s', s''

$$h(s,s'') \leq h(s,s') + h(s',s'') \tag{5.8}$$

instead of just to the goal state. This allows for an efficient cost update of states in the priority queue while avoiding expensive reordering operations.

To achieve this, D* Lite maintains only lower bounds of the optimal path costs in the priority queue. Whenever it detects changed transition costs between two states s and s', the algorithm stores the estimated cost change $h(s,s')$. This cost change is identical for all states in the priority queue with respect to maintaining the lower bounds. Thus, it is sufficient to only add it to states that are inserted into the queue later on, instead of adjusting all states in the queue.

5.5.2. Anytime Dynamic A*

By combining the anytime properties of ARA* with the incremental replanning of D* Lite, the AD* algorithm can cope with both dynamic environments and complex planning problems (Likhachev et al., 2005). As ARA*, it inflates the heuristic with a factor w

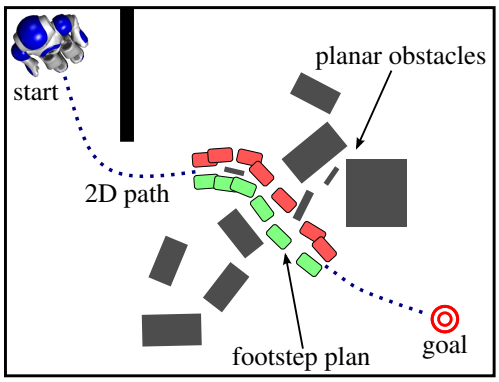


Figure 5.6.: Illustration of adaptive level-of-detail planning for a humanoid. Planning a coarse 2D path in open spaces and a detailed footstep plan only where needed enables fast and efficient results.

and incrementally decreases w as the planning time allows. Whenever cost changes are detected in the environment, the affected states are marked as inconsistent and will be re-evaluated similar to D* Lite.

AD* differentiates between states that are under- and overconsistent due to a change in the environment, i.e., $g(s) < rhs(s)$ or $g(s) > rhs(s)$. For overconsistent states, the key in the priority queue uses the heuristic inflated by w. For underconsistent states, on the other hand, the heuristic may not be inflated to maintain lower bounds and to propagate the updated costs to neighboring states. The key for these states thus uses the uninflated heuristic value.

Note that incrementally repairing state values may be too expensive when large parts of the environment change due to the increased bookkeeping. Thus, it can be beneficial to plan from scratch when the number of changed states exceeds a threshold.

5.6. Adaptive Level-of-Detail Planning

While planning footsteps with an anytime planner is well suited to enable a humanoid robot to reach its navigation goal, it may still require too much planning time to generate solutions over long distances. In many environments, it is also not necessary to plan the complete path at the high granularity of footsteps, since only parts of the environment contain obstacles while the other areas are free space. In these free areas, a 2D path can be quickly computed on a grid representation and can be safely followed by a humanoid with a corresponding walking controller. Close to obstacles, however, footstep planning offers a greater flexibility and results in more efficient plans since the robot can step close to or over obstacles. Thus, as illustrated in Figure 5.6, we propose to adapt the level-of-detail of the planner based on the environment. We can hence combine fast 2D planning in open spaces with footstep planning in areas containing obstacles. Further use cases could be to employ different modes of locomotion depending on the environment (Hauser

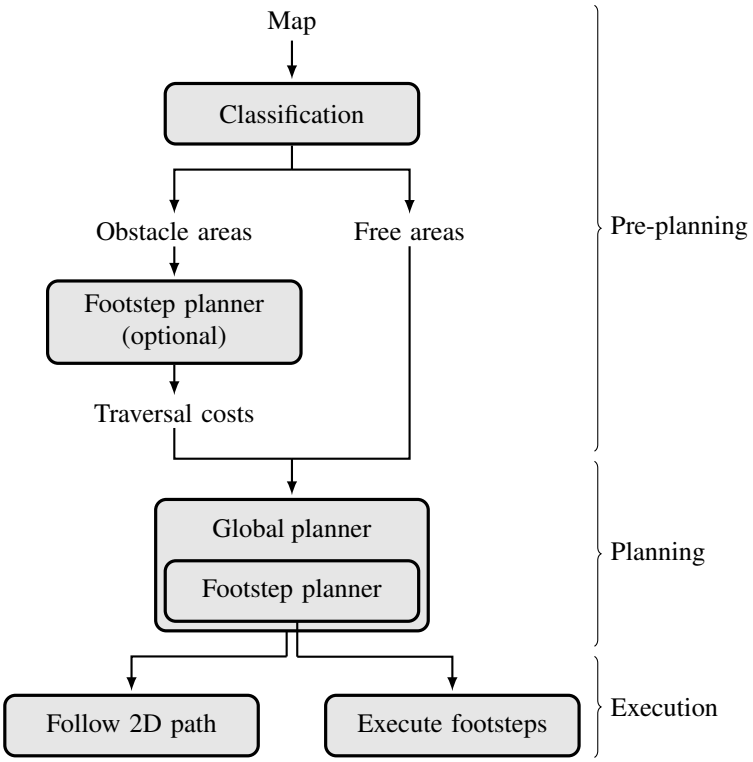


Figure 5.7.: Overview of adaptive level-of-detail planning.

et al., 2007) or to use stair climbing motions as in the previous chapter whenever needed in a global plan.

Figure 5.7 shows an overview of our method. An initial classification of the environment yields free areas and obstacle areas, the latter also containing the areas close to obstacles. With a footstep planner, we then estimate the traversal costs of each obstacle region and store them in the map. This step is optional and can be replaced by a learned heuristic. During planning, we employ a regular 2D grid search in the free areas and in addition use the pre-planned obstacle traversal costs as shortcuts across obstacles. As soon as the search reaches the goal, the segments crossing an obstacle region are converted into footsteps by using a footstep planner from the entry to the exit point of the obstacle region. Finally, the robot uses a walking controller to execute the planned footsteps and a velocity-based controller to follow the 2D path. This interleaved approach results in efficient plans using the level of detail required for each type of region in the environment. Note that, opposed to our interleaved approach, a hierarchical planner would first plan a 2D path and then footsteps along that path, which would result in detours around the obstacles instead of crossing them.

5.6.1. Classification and Segmentation

We first classify the environment into regions of different complexity, each of which is suitable for a certain kind of planner. In this work, we restrict the classification to wide areas in which safe navigation can be performed by planning 2D paths and areas containing narrow passages or planar obstacles in which more detailed planning is necessary.

To classify the environment, we segment the map $\mathcal{M}$ based on the humanoid's circumcircle, which we enlarge to account for the swaying walking motion. This corresponds to the clearance needed when walking along a 2D path and neglecting more detailed motion planning. Using a distance map D containing the Euclidean distance to the closest obstacle for each cell (x,y), we determine connected areas larger than the clearance radius r. In the corresponding regions, 2D path planning can be applied to generate efficient paths avoiding obstacles. We denote this free area as $\mathcal{M}^f \subseteq \mathcal{M}$ and compute it according to

$$\mathcal{M}^f = \{(x,y) \in \mathcal{M} \mid D(x,y) > r\} \ , \tag{5.9}$$

followed by a segmentation into individual disjunct regions

$$\mathcal{M}^f = \mathcal{M}^f_1 \cup \mathcal{M}^f_2 \cup \cdots \cup \mathcal{M}^f_n \ . \tag{5.10}$$

In standard 2D planning with an enlarged robot circumcircle, the remaining areas $\mathcal{M}^o = \mathcal{M} \setminus \mathcal{M}^f$ would be avoided to not risk any collision. In contrast to that, our approach applies more detailed, complex planning in these regions, hereby taking into account more degrees of freedom to find collision-free paths. This directly results in shorter paths since, for example, obstacles can be closely passed or stepped over.

We also segment $\mathcal{M}^o$ into disjunct regions

$$\mathcal{M}^o = \mathcal{M}^o_1 \cup \mathcal{M}^o_2 \cup \cdots \cup \mathcal{M}^o_m \ , \tag{5.11}$$

which we call *obstacle* or *footstep* regions. Figure 5.8 displays the classification of a typical indoor environment with various planar and non-planar obstacles.

5.6.2. Estimation of Traversability Costs

The contour $C(\mathcal{M}^o_i)$ of an obstacle area $\mathcal{M}^o_i$ contains the 2D map cells of the free area $\mathcal{M}^f$ that lie on the border to $\mathcal{M}^o_i$. To determine the actual traversability of all $\mathcal{M}^o_i$ and to estimate the corresponding path costs, we densely sample pairs of entry and exit points t_j:

$$\mathcal{T}_i = \{t_1, \ldots, t_{n_i}\} \subset C(\mathcal{M}^o_i) \times C(\mathcal{M}^o_i) \, . \tag{5.12}$$

Since $C(\mathcal{M}^o_i) \subset \mathcal{M}^f$, all entry and exit points can be reached with the fast planner operating only in $\mathcal{M}^f$.

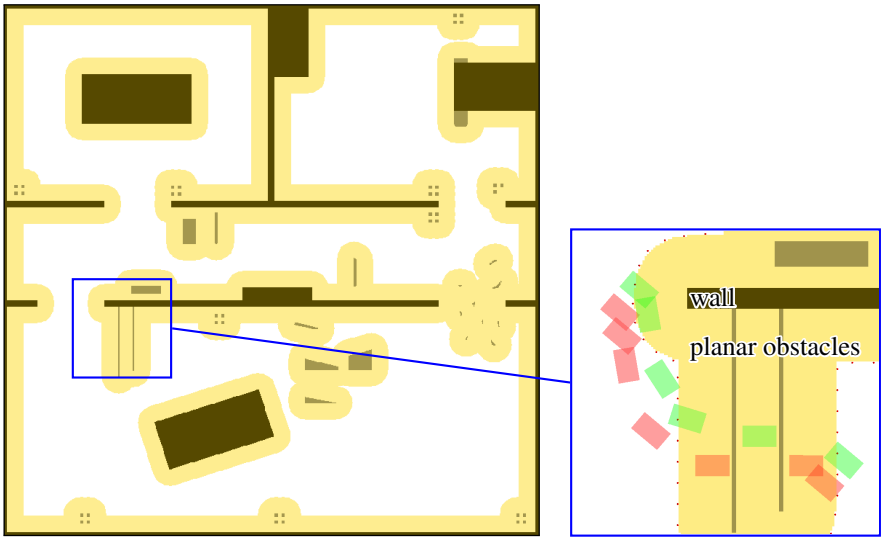


Figure 5.8.: Environment classification for adaptive level-of-detail planning: General obstacles are black, planar obstacles gray. The white area was classified to be suitable for 2D planning. In the yellow areas, a footstep planner is used to plan efficient, collision-free paths. The close-up on the right shows sampled contour points (red dots) and an exemplary footstep path between two points.

Our system then applies a footstep planner for all pairs in $\mathcal{T} = \bigcup_i \mathcal{T}_i$. The footstep planner is hereby aware of the environment semantics, i.e., it can step only over planar obstacles and needs a certain clearance to all other obstacles such as walls. At this stage, the start and goal orientations for footstep planning are given by the straight-line connection between the corresponding entry and exit points.

If a footstep path is found, we connect the entry and exit point in the environment representation with the resulting path costs. These yield an estimate of traversing the area $\mathcal{M}_i^o$. Note that this value is only an estimate for the global planning stage later, since then the pose orientation may be different from the one used for precomputation. By computing footstep paths between all pairs of states in $\mathcal{T}_i$ and connecting them, we obtain the estimated costs of traversing the obstacle region $\mathcal{M}_i^o$.

The preplanning process is carried out once for a given map. The estimation is reasonably fast because the footstep plans are comparably short and they can be easily parallelized as all plans are independent. To avoid the precomputation in mostly dynamic environments, a heuristic can be derived from once learned traversal costs. It estimates the costs based on the average traversal costs normalized by the straight-line distance while discarding any connection across an impassable, non-planar obstacle.

5.6.3. Global Planning

The global planner in our framework computes the path between the start and goal state using an augmented map. This map contains labeled regions $\mathcal{M}_i^f$ and $\mathcal{M}_i^o$ (free or obsta-

cles), annotated obstacles (planar or not) from $\mathcal{M}$, as well as transition costs for each of the transitions in $\mathcal{T}_i$ of each obstacle region $\mathcal{M}_i^o$. For global planning, we allow transitions from a state to its eight-connected neighborhood on the 2D grid and, if it is an entry point for a transition in $\mathcal{T}$, to all corresponding exit points. The costs of the obstacle traversal transitions are given by the precomputed estimates.

It is important to have a unified cost metric for the different planners. In our case, the footstep planning costs correspond to the time it takes the robot to execute the footsteps. Hence, we scale the costs of 2D paths in the map so that they are normalized with respect to the footstep planner. Accordingly, the costs for a 2D path of a certain length are the same as walking on a footstep plan of the same distance between start and goal. By adjusting the scaling factor, it is possible to give preference to one plan over the other, e.g., when single footsteps are executed slower than a fast path following behavior.

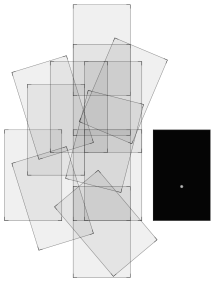
Global planning now corresponds to a search that proceeds by expanding states in $\mathcal{M}^f$ and $\mathcal{T}$. As before, we can here apply any kind of heuristic search such as A*, ARA*, or AD* when incremental replanning is required. When the planner found a path to the goal, this path consists of a sequence of 2D path segments and obstacle region traversals. Then, a footstep plan for each segment crossing an obstacle region is generated as discussed in the previous sections. Opposed to the preplanned estimate, this footstep plan now uses the correct orientation of the entry and exit poses which is given by the connecting 2D path segments. Should the start or goal state be within an obstacle region, we first check whether it is accessible by placing footsteps. Our approach then plans footsteps to the surrounding contour points, from which the global planner can continue with its 2D search.

This global planning strategy is highly efficient and yields cost-optimal paths in terms of the given cost metric when using A*. Only due to the sampling of transitions, the paths might differ from the optimal footstep paths found by globally planning footstep actions. Opposed to other techniques, our approach considers the costs at a globally unified scale and the different plan segments are not planned independently, but are considered in an interleaved fashion.

During execution, the robot can follow the 2D paths easily by walking with an omnidirectional controller to sub-goals on the path. As soon as a footstep plan segment is reached, the robot then switches to the footstep controller.

5.7. Evaluation

In our experiments, we evaluated and compared the performance of the different planning approaches. Unless noted otherwise, we used the footstep parameterization shown in Figure 5.9 which corresponds to a full-size humanoid such as HRP-2 or ASIMO. We chose a lattice graph resolution (discretization) of 1 cm for x/y and 5° for θ. The planners use this discretization to check equality between expanded states on the lattice. The occupancy maps for collision checks also have a resolution of 1 cm but we preserve angles to be con-



Parameter	Value / Range
w	10 cm
h	16 cm
Δx	$[-10, 22]$ cm
Δy	$[12, 28]$ cm
$\Delta\theta$	$[-0.23, 40.0]^\circ$

Figure 5.9.: Footstep parameterization for a large humanoid such as HRP-2 or ASIMO according to the model in Figure 5.1, consisting of 14 different steps.

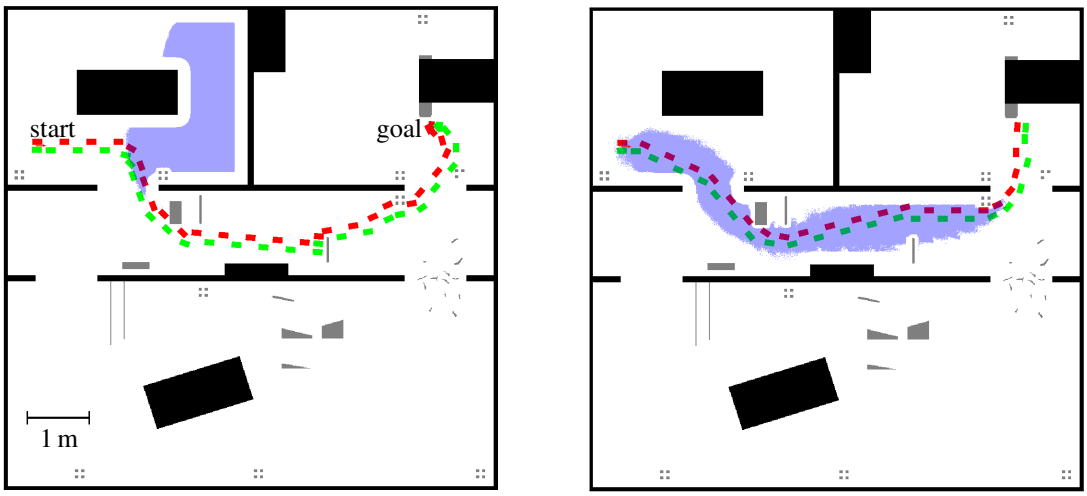
tinuous in the collision check. All planning times are given for a single core of a desktop CPU (Intel Core i7, 3.4 GHz).

5.7.1. Anytime Footstep Planning

We first evaluated anytime footstep planning in static environments and compared different heuristics for the planners A*, ARA*, and R*. In our setting, the robot needs to maintain a clearance of 15 cm to obstacles marked as walls. We set the obstacle inflation radius for the 2D Dijkstra heuristic to the foot incircle radius of 5 cm, as explained in Section 5.3. For R*, we experimentally determined the parameters $\Delta = 1.5$ m, $k = 20$, and the expansion limit for easy-to-find paths as 500 as best suited. For comparison, planning a distance of 1.5 m in free space requires 100 – 200 A* expansions with our footstep parameterization.

Planning in an Indoor Environment with Limited Time

In a first experiment, we evaluated the algorithms qualitatively in an indoor environment containing several rooms with shallow obstacles that can be stepped over and obstacles that have to be avoided such as walls. To obtain near-realtime plans suitable for navigation, we set the time limit for the planners to five seconds and the initial heuristic inflation weight $w = 10$. The results for a first scenario are shown in Figure 5.10. With the Euclidean distance heuristic, ARA* fails to find a plan within the given time. It requires 43 seconds for a first solution due to expansions being misled into local minima. In this planning scenario, the 2D Dijkstra path heuristic is well suited due to the connectivity of the rooms through the hallway and relatively few planar obstacles. This leads ARA* with the Dijkstra heuristic to find a first solution after only 0.7 ms and a nearly optimal path ($w = 1.4$) within five seconds. In comparison, R* with the Euclidean heuristic finds a first solution after 1 s and a solution with $w = 7$ within the time limit of five seconds. It expands fewer states than ARA* with the same heuristic. Since we are mainly interested in using this admissible heuristic, we omit results of R* with the Dijkstra heuristic.

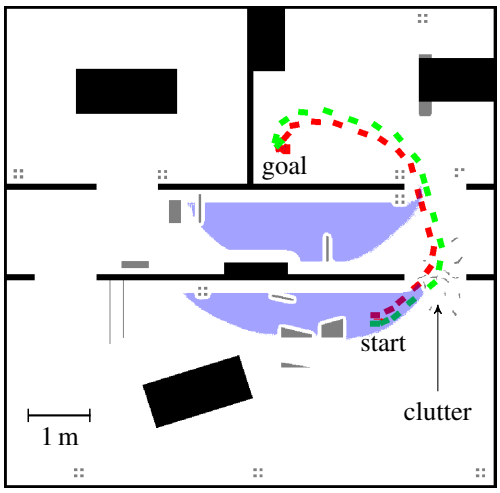


(a) ARA* with Euclidean heuristic after 43 s

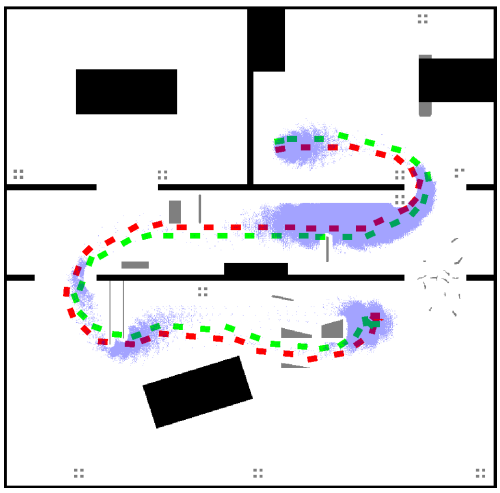
(b) ARA* with Dijkstra heuristic after 5 s

(c) R* with Euclidean heuristic after 5 s

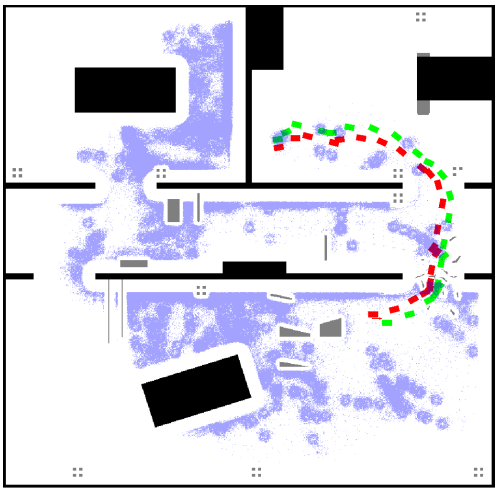
Figure 5.10.: Footstep planning with a time limit of 5 seconds between two rooms of an indoor environment. Gray areas denote shallow obstacles that the robot can step over, black areas denote objects that the robot has to avoid (tables and walls), and blue areas denote the expanded states. ARA* with the Euclidean heuristic fails to find a path within the time limit and requires 43 seconds to find a first solution with $w = 10$. With the Dijkstra heuristic, ARA* finds a near-optimal path within the time limit (final $w = 1.4$). R* finds a path even with the Euclidean heuristic (final $w = 7$).



(a) ARA* with Euclidean heuristic after 92 s



(b) ARA* with Dijkstra heuristic after 5 s



(c) R* with Euclidean heuristic after 5 s

Figure 5.11.: Footstep planning with a time limit of 5 seconds through a cluttered passage. ARA* with the Euclidean heuristic fails to find a path within the time limit and requires 92 seconds to find a first solution with $w = 10$. The inadmissible Dijkstra heuristic results in a detour due to the clutter blocking the heuristic path. R* finds a path even with the Euclidean heuristic (final $w = 8$).

The problem of using the 2D Dijkstra heuristic becomes apparent in a second scenario with different start and goal locations. As shown in Figure 5.11, the optimal path leads through a passage blocked by clutter that the robot can step over. A similar situation arises when there is a door sill that the robot should avoid stepping onto, which poses an obstacle completely blocking the doorway in 2D. Again, ARA* with the Euclidean distance heuristic fails to find a path within reasonable time while R* succeeded within the time limit of five seconds. In this scenario, the Dijkstra heuristic is no longer admissible since the clutter blocks its path through the doorway. As a result, ARA* with this heuristic wrongly expands a non-optimal path despite w reaching 1.4 after five seconds. The reason for this is that due to the heuristic inadmissibility the suboptimality guarantees no longer hold. This scenario demonstrates that the 2D Dijkstra path heuristic can be a poor choice for footstep planning in cluttered environments.

Initial Plan Results in Dense Clutter

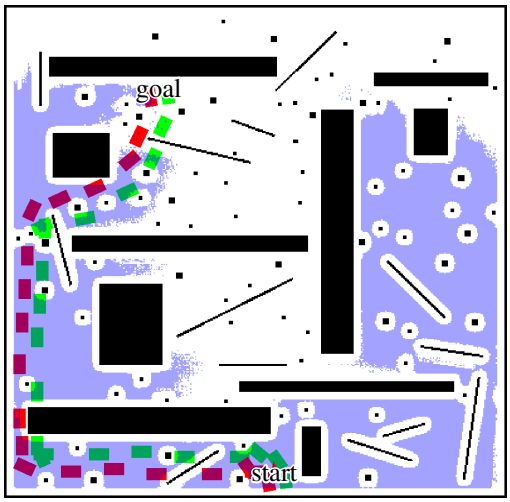
In the next set of experiments, we evaluated how the different planners can cope with densely cluttered environments in which there are only a few valid footstep positions. Here, we analyze the planning time and path quality of the first solution. To this end, we plan footstep paths between 12 random start and goal locations approximately 3.5 m apart in a $4 \times 4\,\mathrm{m}^2$ environment. Figure 5.12 shows the environment with a comparison of the different paths and expanded states for one start and goal location.

Table 5.1 lists the aggregated statistics as mean and standard deviation over the 12 planning attempts with $w = 5$, while Figure 5.13 shows the success rate over time. A* with the Euclidean distance heuristic requires on average 33 s to find the optimal paths containing 37 individual steps on average, which demonstrates the need for anytime suboptimal planning for these complex problems.

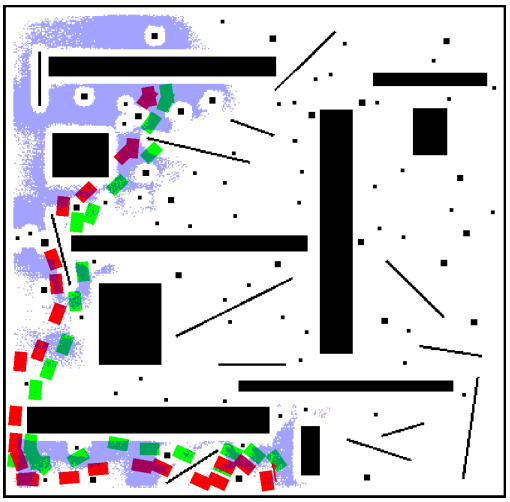
As in the previous experiments, R* succeeds in finding fast results in general. ARA* with the admissible Euclidean heuristic requires more planning time even for initial suboptimal results, while it needs less time with the inadmissible Dijkstra heuristic. The Dijkstra heuristic also leads to some longer paths, since it overestimates in some instances. All anytime planners find paths significantly faster than A*, at the cost of longer paths for the initial solution. The actual suboptimality of the solution costs, as opposed to the upper bound of $w = 5$, was less than 2.0 in all instances. While this initial solution is often already sufficient, both R* and ARA* with the Euclidean heuristic will converge to the same optimal result as A* given enough time.

5.7.2. Incremental Replanning

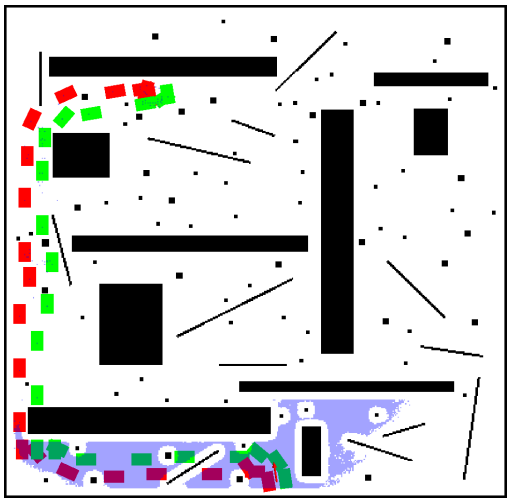
In our evaluation of AD*, we analyzed its incremental replanning capabilities when either the start state or small parts of the environment change. In the cluttered environment, we planned an initial plan to the goal with AD* and the Euclidean distance heuristic. We hereby planned backwards for efficient replanning, using $w = 5.0$ as initial suboptimality



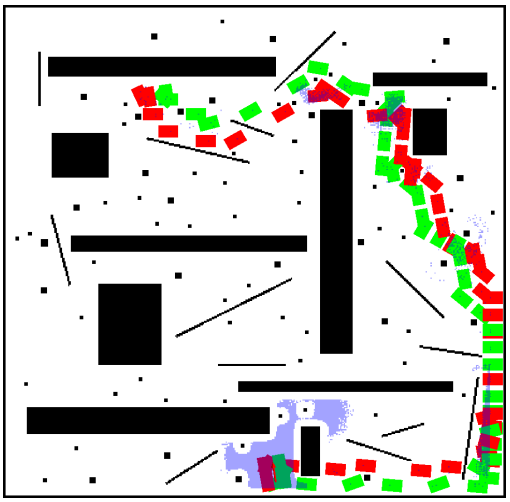
(a) A* (Euclidean heuristic). Optimal result after 11.9 s.



(b) R* (Euclidean heuristic, w=5). First result after 0.4 s.



(c) ARA* (Euclidean heuristic, w=5). First result after 2.7 s.



(d) ARA* (2D Dijkstra heuristic, w=5). First result after 0.7 s.

Figure 5.12.: Footstep planning in a densely cluttered $4 \times 4\,\mathrm{m}^2$ area. Blue areas denote the expanded states. Computing the optimal path with A* takes a long time due to local minima, and requires expanding large areas of the state space (a). Anytime-planners provide fast solutions for realtime navigation with suboptimality bounds. R* explores the state space even with the weakly informed Euclidean distance heuristic in short time (b). ARA* depends more on the heuristic function for fast initial results (c)(d).

Planner	Heuristic	Planning time [s]	Path costs	Suboptimality
R* ($w = 5$)	Euclidean	0.32 ± 0.23	16.45 ± 3.16	1.49 ± 0.25
ARA* ($w = 5$)	Euclidean	2.15 ± 2.21	13.57 ± 1.15	1.23 ± 0.09
ARA* ($w = 5$)	2D Dijkstra	0.56 ± 1.13	20.41 ± 5.08	1.84 ± 0.35
Optimal: A* ($w = 1$)	Euclidean	33.31 ± 15.00	11.06 ± 1.20	1.0

Table 5.1.: Performance comparison for first footstep plan solutions in a densely cluttered environment (mean and standard deviation for 12 different scenarios).

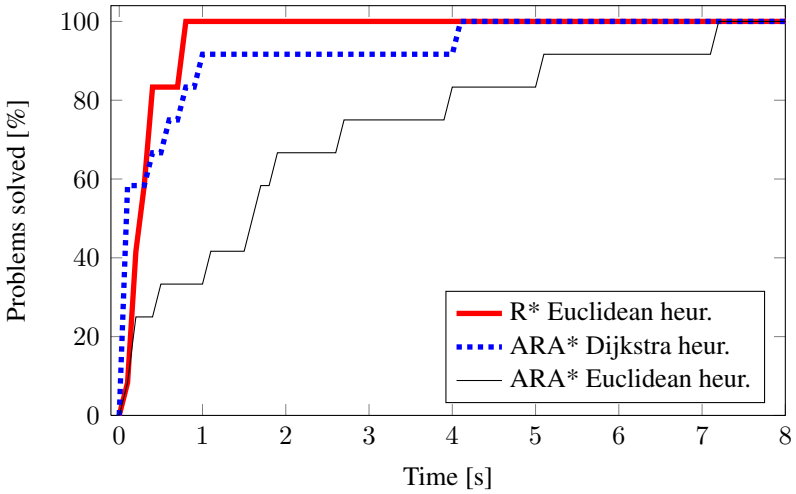
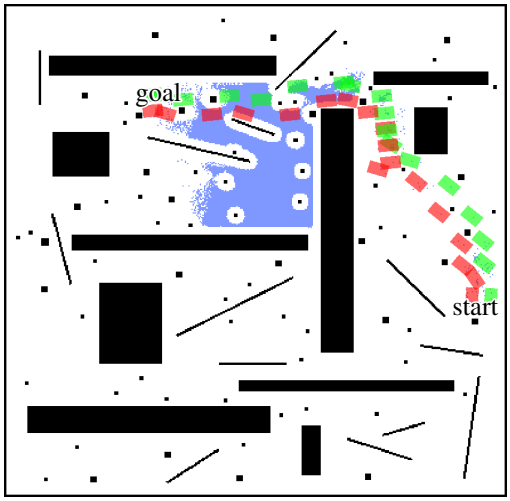


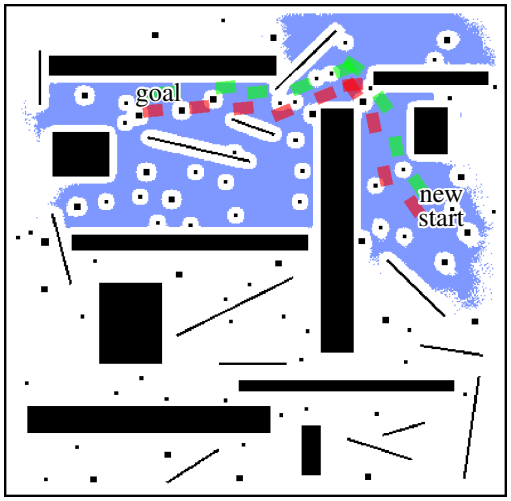
Figure 5.13.: Success rate for 12 random start and goal configurations in a densely cluttered environment. The percentage of solved problems within a certain time is shown for the first solution with $w = 5$.

with a time limit of two seconds to obtain fast planning results for navigation. The resulting plan is shown in Figure 5.14a. The final suboptimality after two seconds was $w = 3.4$. The humanoid then used the time while executing the next steps to incrementally improve the plan to the optimal solution with $w = 1.0$, which succeeded after four planning runs with a time limit of two seconds. We simulated a localization update that corrects for motion drift by shifting the current foot locations by 5 cm, forcing an update of the motion plan. AD* obtained a new optimal plan within only 0.2 seconds, shown in Figure 5.14b. Compared to that, replanning without reusing the previous information reached $w = 2.6$ within the two second limit and required 6.18 seconds to obtain the optimal solution.

Figure 5.15 illustrates replanning in a different situation. The initial, optimal path with AD* here required 11 seconds and expanded 1 729 629 states. A small change of the obstacle positions in the environment then forced the planner to replan. With incremental replanning, AD* was able to obtain an updated optimal solution within 0.7 seconds and newly expanded only 76 663 states. A completely new plan, on the other hand, required

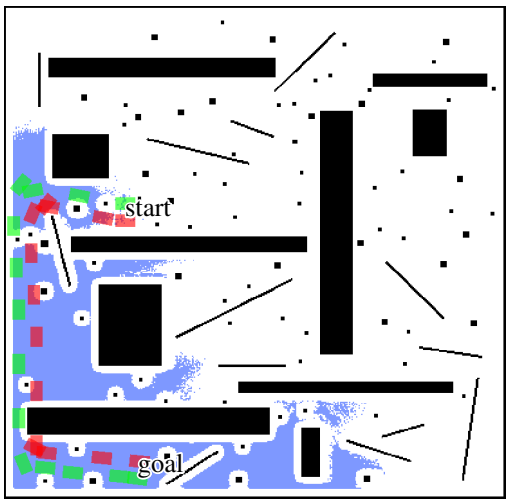


(a) Initial footstep plan with starting w=5 and final $w = 3.4$ after 2 s.

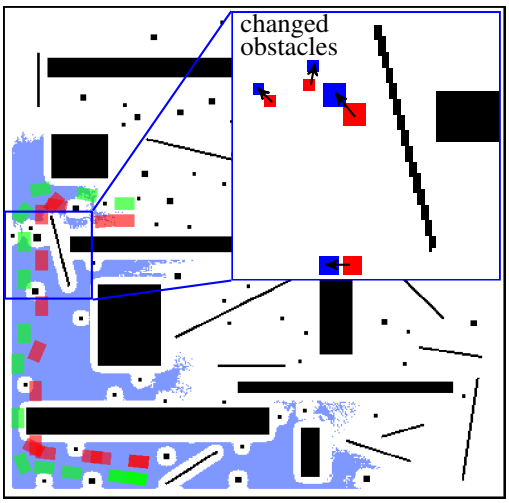


(b) Replanning during execution from a different start configuration within 0.2 s.

Figure 5.14.: Incremental replanning with AD* from an updated start state in a $4 \times 4\,\mathrm{m}^2$ area using the Euclidean heuristic. Blue areas denote the expanded states. After computing an initial plan within two seconds, the robot executed the plan and improved it during execution to obtain the optimal solution. It then had to replan from an updated pose estimate. AD* was able to update the optimal solution within 0.2 s. Completely replanning to the optimal solution would take 6.18 s.



(a) Optimal footstep plan ($w = 1.0$) after 11 s.



(b) Replanning due to a small change in the environment (highlighted) within 0.7 s.

Figure 5.15.: Incremental replanning with AD* due to a change in the $4 \times 4\,\mathrm{m}^2$ environment using the Euclidean heuristic. Blue areas denote the expanded states. After computing the initial, optimal plan within 11 seconds (a), the robot had to update it due to small changes of obstacles positions (b). Incrementally updating the optimal plan with AD* only took 0.7 s, whereas a completely new optimal plan required 10.3 s.

Precomputation			Planning performance	
Density [m]	# plans	time [s]	time [s]	Path costs
0.05	32266	3640	0.44 ± 0.18	12.05 ± 1.11
0.1	7820	884	0.47 ± 0.20	11.97 ± 1.19
0.2	1892	202	0.41 ± 0.32	11.78 ± 1.08
0.4	516	51	0.49 ± 0.47	12.16 ± 1.16
0.6	236	27	0.35 ± 0.20	12.47 ± 0.68

Table 5.2.: Overview of precomputation effort and performance for adaptive level-of-detail planning.

10.3 seconds and expanded 1 620 117 states. This demonstrates the efficient replanning capabilities of AD* whenever parts of the environment or the starting state change.

5.7.3. Adaptive Level-of-Detail Planning

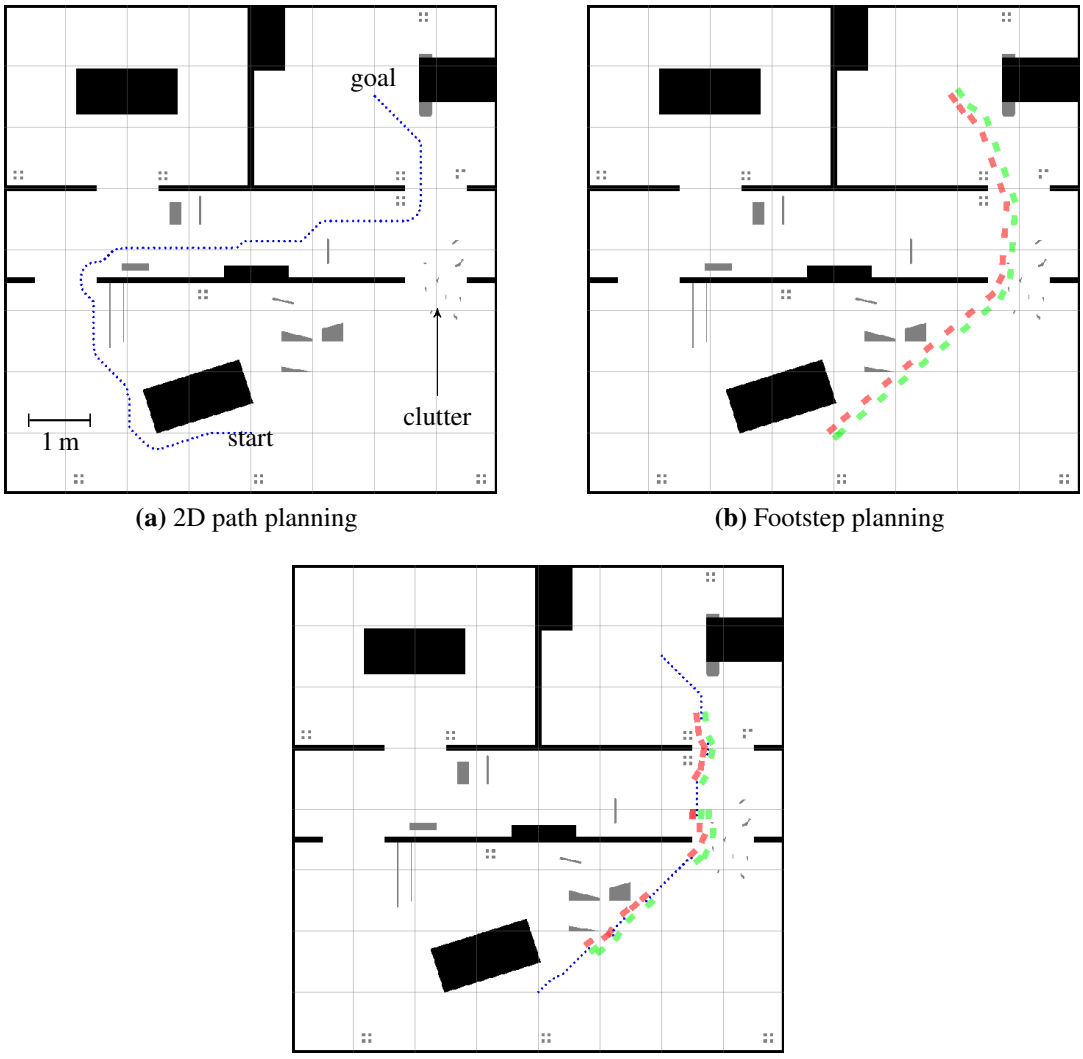
In the next experiment, we evaluated the performance of adaptive level-of-detail planning. While it is possible to use any kind of search-based planner in our approach, we are interested in comparing optimal paths for the purpose of this evaluation. We thus used A* for both footstep planning and 2D grid planning in the open spaces. The footstep planner used the Euclidean distance to the goal as heuristic.

Precomputation

We first evaluated the computational effort of estimating the obstacle traversal costs in the preplanning phase. Table 5.2 displays the average precomputation times for different densities of sampled contour points and the resulting planning performance as average and standard deviation. For evaluation, we used ten different start and goal configurations in the indoor environment shown in Figure 5.8. We exploited parallelization in the classification phase by running four threads on a 3.4 GHz Intel Core i7 CPU. As can be seen, a costly dense estimation is not required to yield acceptable planning performance in terms of time and path costs. Only when there are too few transitions across obstacle areas (more than 0.6 m apart), the performance degrades. We hence used a distance of 0.2 m between the contour points of footstep regions in the next experiment.

Planning

We then compared the planning time and resulting paths of our approach with 2D path planning and footstep planning. A 2D-only path was fastest to compute in less than a second but resulted in a significant detour through the hallway since it could not pass the clutter (Figure 5.16a). The global footstep plan took substantially longer to compute (29 s),



(a) 2D path planning

(b) Footstep planning

(c) Adaptive level-of-detail planning

Figure 5.16.: Comparison of our adaptive level-of-detail planning approach to conventional 2D path planning and footstep planning. Conventional 2D path planning is fast but takes detours around cluttered areas and needs to respect larger safety margins to obstacles (a). A footstep path exploits the stepping capabilities of humanoid robots resulting in a more efficient path, but takes substantially more time to compute (b). Our proposed approach combines the advantages of both methods by using a conventional 2D path planner in open spaces and footsteps planning in more complex regions (c).

Approach	Planning time [s]	Path costs
2D grid planning	0.51 ± 0.11	19.18 ± 5.28
Footstep planning	43.70 ± 25.66	11.78 ± 1.16
Adaptive with precomputation	0.41 ± 0.32	11.78 ± 1.08
Adaptive, no precomputation	0.99 ± 0.54	12.41 ± 1.43

Table 5.3.: Performance comparison between adaptive level-of-detail and regular planning methods.

but resulted in a more efficient path as planar obstacles are stepped over instead of walking around them (Figure 5.16b). Compared to that, our adaptive level-of-detail planning approach combines the advantages of both in that it is as fast as 2D planning and resulted in an efficient path (Figure 5.16c). Footstep planning was only invoked where needed. The path costs in this scenario were only 2% higher compared to the optimal footstep plan, while being 51% less compared to 2D planning.

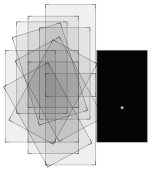
For a statistical evaluation, we employed the three planning approaches between ten different start and goal configurations in the same environment. Each required a path length of approximately 8 m in the optimal case. As evaluation criteria in Table 5.3, we used the planning time and execution costs for each plan (mean and standard deviation). Footstep planning resulted in the best paths, but the optimal solution required a planning time of 43.7 s on average and 94 s at most. Conventional 2D planning was faster, but the resulting paths were significantly longer as they cannot pass close to obstacles or step over objects. In contrast to that, our adaptive approach returned paths that were as efficient as full footstep plans in a short time.

Ideally, the environment is known and the costs to traverse obstacle regions can be estimated in the preprocessing step. When precomputation is not possible, e.g., because the environment is not completely static, a heuristic can be used to estimate the obstacle traversal costs. As can be seen in Table 5.3, this resulted in 5% longer plans on average. The planning time also increased since in some instances more non-relevant footstep segments had to be planned.

5.7.4. Navigation With Footstep Plans

Finally, we present an experiment that demonstrates the execution of footsteps with a real robot. To this end, we used a Nao humanoid with a laser range finder for localization, which is described in Appendix A. The robot relied on the localization approach described in Chapter 3 to estimate its pose at the beginning and continually updated the estimate while walking. As described in Section 5.1.3, the robot uses this pose estimate to correct for motion drift. The footstep parameterization for the Nao is shown in Figure 5.17.

We supplied a 2D map of planar obstacles that the robot should avoid stepping on. From its initial pose estimate, the robot planned a footstep plan with AD* through the obstacles. Figure 5.18 shows the resulting plan and execution, while Figure 5.19 shows individual



Parameter	Value / Range
w	8.8 cm
h	16 cm
Δx	$[-4, 8]$ cm
Δy	$[9, 16]$ cm
$\Delta\theta$	$[-28.6, 28.6]^\circ$

Figure 5.17.: Footstep parameterization for the Nao humanoid according to the model in Figure 5.1, consisting of 12 different steps.

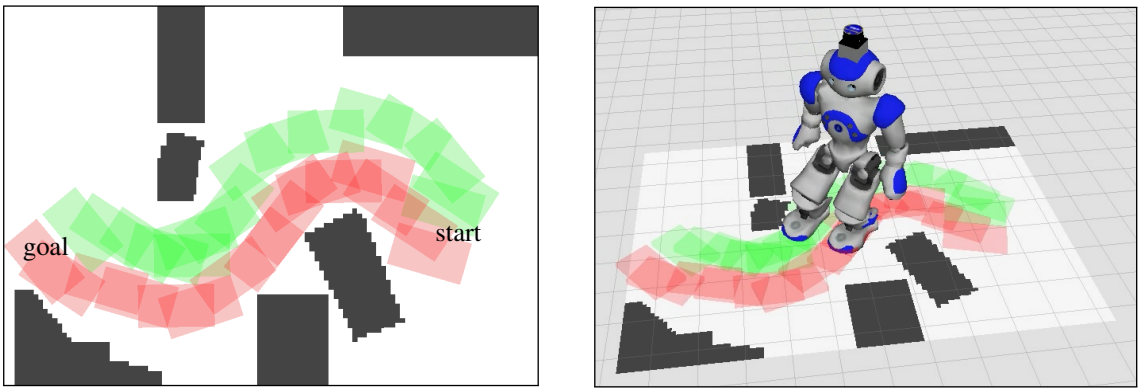


Figure 5.18.: Footstep plan for the Nao humanoid through a narrow passage of obstacles (left) and execution by the robot (right).

steps of the robot executing the plan. A video of the sequence is available online at http://www.youtube.com/watch?v=o0rlrEHN1w4. As can be seen, the humanoid successfully avoided the obstacles while walking to the goal location.

5.8. Related Work

In the last few years, several approaches have been presented that aim to plan motions for a humanoid in order to reach a navigation goal.

The concept of planning footsteps for humanoid navigation was originally introduced by Kuffner et al. (2001). The authors planned a sequence of statically stable footsteps in a planar environment with clutter as obstacles. The sequence was planned using dynamic programming with a greedy heuristic. Chestnutt et al. (2003) proposed to use the A* heuristic search instead with a discrete footstep set and introduced various terrain costs. Their cost metrics encode the relative safety of uneven terrain, the required effort, and the complexity of the motion. In their comparison between A* and best-first search, A* was significantly slower for longer paths but returned better paths around non-traversable obstacles. In a later work (Chestnutt et al., 2005), the authors presented experiments in

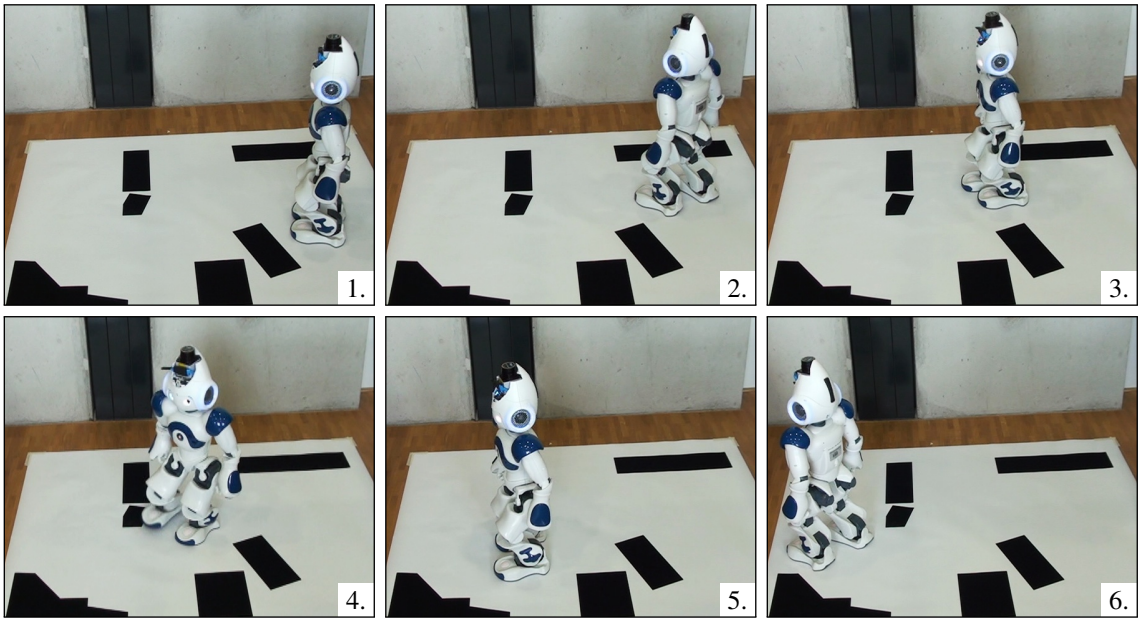


Figure 5.19.: The Nao humanoid executes the footstep plan shown in Figure 5.18, carefully avoiding obstacles.

which an ASIMO humanoid stepped through static and dynamic planar obstacles in a known environment based on pose estimates from an external motion capture system. An extension of the work uses an adaptive action model, which locally adjusts footstep locations when they collide with an obstacle (Chestnutt et al., 2007). This potentially leads to more efficient paths since the robot can locally extend its discrete footstep set as needed. Their approach enabled an HRP-2 humanoid to ascend a small staircase using footstep planning. In all of these approaches, footstep planning was only employed on relatively small, but complex environments due to the long planning times with A*.

Huang et al. (2013) apply the above methods and plan footsteps with A* to optimize an energy-based cost function. The optimized costs for footsteps are a sum of terrain cost features and costs derived from human walking studies.

Gutmann et al. (2005) presented a fast planning approach that runs on the embedded hardware of a small humanoid. In a labeled 2D grid map that also contains the height of obstacles, they plan a 2D path with orientations for a cylindrical approximation of the robot. In an A* search, the authors consider a fixed set of motion primitives such as walking forward or turning by 45°. With this relatively coarse discretization of the environment and the approximation of the robot shape, a humanoid robot cannot step through clutter by stepping over obstacles, but will completely avoid these areas.

Early approaches for planning long distances first applied a 2D path planner such as A* in a grid representation and then followed that path by locally generating sequences of footsteps around that path (Bourgeot et al., 2002; Chestnutt and Kuffner, 2004; Li et al., 2003; Pettré et al., 2003). For example, Bourgeot et al. (2002) proposed to first plan a

2D path in a height map representation that contains flat ground and slopes. Depending on the underlying terrain, different sets of footholds are then used to follow the linear path. Candido et al. (2008) proposed a hierarchical motion planner for humanoids. After an initial decomposition of the environment into regions, a high level planner computes a 2D path. A local planner then computes motion primitives between sub-goals of the global plan. The drawback with these hierarchical methods is that the global planner operates on a lower-dimensional configuration space, and is thus not aware of the local planner's capabilities. Hence, the resulting plans could take detours around obstacles and discontinuous terrain that could otherwise be stepped over with footsteps.

Baudouin et al. (2011) and Perrin et al. (2012a) presented footstep planning using RRT. The authors hereby used a large number of possible stepping motions and planned so-called half steps, which represent the upwards and downwards motion of the feet. Swept volume approximations of the humanoid's lower legs sped up collision checks while allowing motions for stepping over obstacles. In a following work, Perrin et al. (2012b) considered real-time footstep planning with a hybrid bounding box approach. Their approach plans a continuous trajectory for the bounding box of the humanoid's upper body and discrete placements for the bounding boxes of the feet, which correspond to footstep placements. Using the concept of weakly collision-free configurations (Perrin et al., 2011), their approach is able to plan motions for the bounding boxes using traditional randomized planners and to generate dynamic walking trajectories from them. In simulated experiments, the robot HRP-2 successfully stepped through a cluttered environment using the approach.

Hauser et al. (2005) applied probabilistic roadmaps to generate highly complex and computationally expensive whole-body motions, such as climbing a large step or a ladder. Dalibard et al. (2011) introduced the notion of small-space controllability for a two-staged motion planner. A randomized planner first computes a statically stable collision-free trajectory for the humanoid's body without stepping motions. Along this trajectory, a controller then generates a dynamically stable walking gait. The current drawbacks of their method are the relatively long computation times and that the global planner is not able to plan motions for stepping over obstacles, thus planning a detour around them.

While these randomized methods provide fast solutions, there are no bounds on the quality of the solution as our A*-based methods provide. To this end, RRT* provides probabilistic convergence to an optimal solution (Karaman and Frazzoli, 2011). The algorithm was recently extended for anytime motion planning (Karaman et al., 2011), albeit only for a wheeled vehicle so far. The remaining downside of randomized methods is, however, that infeasible queries cannot be detected within a finite running time.

A number of approaches exist that search for a global navigation plan with different levels of detail. Gochev et al. (2011) presented an approach similar to our adaptive level-of-detail planning. In their adaptive dimensionality planning framework, the authors combine planning in a high-dimensional space wherever needed with fast planning in a low-dimensional space. Experiments were performed with a wheeled platform in

three-dimensional space and with a 7 DOF robotic manipulator. A recent work (Gochev et al., 2013) extended the approach with incremental replanning capabilities.

Morales et al. (2004) proposed to classify the configuration space for randomized planners with machine learning in order to apply different roadmap-based planners, each best suited for a certain area of the environment.

Finally, there are approaches which use a combination of planners or dynamically adjust the planning granularity (Steffens et al., 2010; Vahrenkamp et al., 2008; Zickler and Veloso, 2010). These methods adapt the planning level in order to obtain accurate short-term results, and only rough long-term results. This is particularly useful when planning in highly dynamic environments. In order to avoid local minima and decide on the feasibility of the plan in more static scenarios, however, it is necessary to compute a complete plan instead of relying a set of local plans.

5.9. Conclusion

5.9.1. Summary

We presented a framework for search-based footstep planning and evaluated different planning approaches for bipedal humanoid robots. Our framework uses a set of footstep transitions and builds a lattice of states as they are expanded by the planner. Obstacles may pose local minima in the search space, which forces an optimal planner such as A* to expand large areas of the search space. To achieve fast planning times, we proposed to use the anytime planners ARA* and R*, which return initial paths with provable sub-optimality fast and improve them as time allows. In our experiments, we found ARA* with the 2D Dijkstra path heuristic suitable for some planning scenarios, but generally in danger of yielding non-optimal paths due to its inadmissibility. ARA* with the admissible Euclidean distance heuristic required long planning times and expanded many states in the presence of local minima. R*, on the other hand, returns fast solutions even with the Euclidean heuristic by avoiding local minima with its random expansion.

Since a robot operating in the real world often needs to replan due to changes in the environment and deviations from its original path, we presented and evaluated efficient replanning of footsteps using AD*. The method extends the anytime capabilities of ARA* with an efficient plan reuse.

For efficiently planning longer paths, we introduced an adaptive level-of-detail approach, which combines fast but coarse 2D planning in open areas with detailed footstep planning through obstacle regions. This enables fast planning times without sacrificing the solution quality due to planning only in 2D. The method can be generally applied to other environments and different types of locomotion. For example, the robot could combine planning footsteps in the plane with stair climbing motions as described in Chapter 4 whenever the environment is suitable.

In our experiments, we evaluated the presented planning approaches in different settings. Furthermore, we demonstrated the applicability to humanoid navigation by employing AD* footstep planning on a real Nao humanoid. The robot used the localization approach from Chapter 3 to estimate its pose, and thus corrected its motion drift while executing the planned footsteps collision-free.

5.9.2. Future Work

The presented footstep planning framework can be employed in three-dimensional environments by adding new stepping capabilities and by accounting for the actual robot shape in the collision check. This enables a humanoid to step over clutter and to climb onto obstacles, as recently demonstrated by Maier et al. (2013).

By combining this approach with our adaptive level-of-detail planning, expensive 3D planning and collision checks can be avoided unless they are needed. The respective environment classification would regard regions as free, planar, or three-dimensional, which is suitable for 2D planning, planar footstep planning, and three-dimensional footstep planning.

Recent advances in motion planning provided anytime capabilities to the asymptotically optimal RRT* (Karaman et al., 2011) or E-Graphs (Phillips et al., 2013), which exploit experience from previous planning runs to speed up the search. It may be promising to apply these techniques to footstep planning, thus speeding up the search to obtain better results in short planning times.

5.9.3. Impact

Our framework was published as open source implementation for ROS at http://wiki.ros.org/footstep_planner in 2011. Since then it has been used by a number of international research groups and was adopted to different humanoid robots, such as the HRP-2 at the University of Tokyo. Pal Robotics use our framework for the REEM-C humanoid[1], illustrated in Figure 5.20. Within the DARPA robotics challenge, team *ViGIR* by TU Darmstadt, TORC Robotics, and Virginia Tech successfully used our framework to plan footsteps for the ATLAS humanoid[2] through clutter. Figure 5.21 shows an example application.

[1] http://www.pal-robotics.com/robots/reem-c (retrieved October 30, 2013)
[2] http://www.bostondynamics.com/robot_Atlas.html (retrieved November 18, 2013)

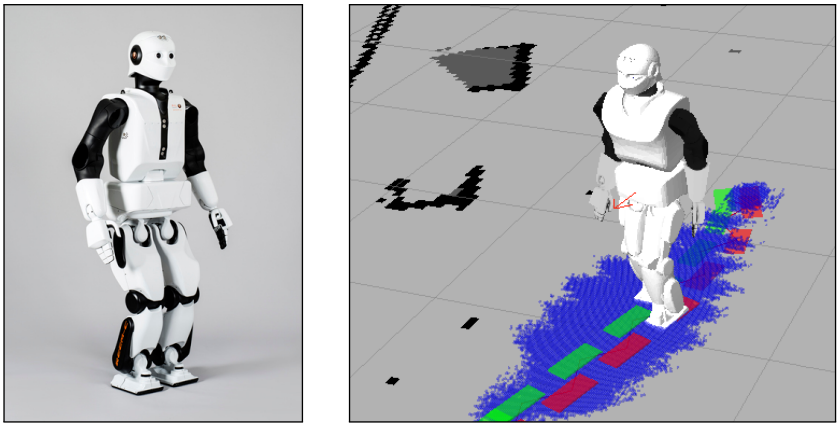


Figure 5.20.: Our footstep planner implementation applied to the Pal Robotics REEM-C humanoid. Pictures courtesy of E. Fernandez / Pal Robotics.

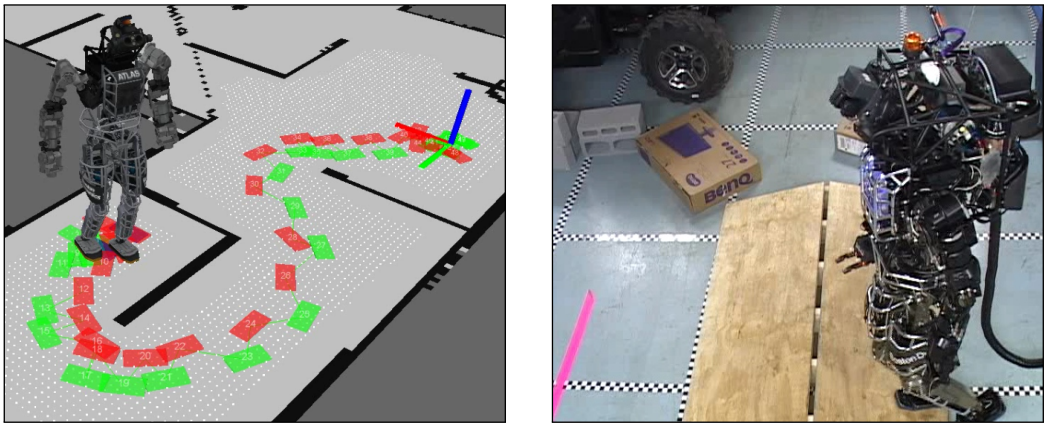


Figure 5.21.: Our footstep planner implementation applied to the Boston Dynamic ATLAS humanoid. Pictures courtesy of A. Stumpf, TU Darmstadt.

Chapter 6

Whole-Body Motion Planning

This chapter describes planning for the whole body of a humanoid robot to perform a manipulation task, such as picking up an object or opening a door. Our approach is based on rapidly-exploring random trees and considers all constraints inherent to the task. The robot must keep its balance, respect its joint limits, and avoid collisions with itself and the environment. Furthermore, when manipulating an articulated object such as a door, the trajectory of the end-effector must follow the constrained motion of the manipulated object in Cartesian space. The planner relies on a database of stable configurations to improve the sampling performance and uses inverse kinematics to enforce constraints for the leg and arm chains. We thoroughly evaluated our planning system in experiments with a Nao humanoid. Whole-body planning greatly expands the robot's reachability. Our approach enables the robot to reach into shelves, to pick up objects, and to open doors and drawers without colliding or losing its balance.

In the previous chapter, we presented navigation planning for humanoids at the level of single footsteps that are placed on traversable areas of the environment. We extend the notion of motion planning for the complete body of the humanoid in this chapter. The goal hereby lies in manipulating the environment, e.g., by picking up small objects or by opening doors and drawers. This capability is also important to navigate in man-made environments, where doors have to be opened regularly to pass between different rooms. Due to their dexterity and human-like body plan, humanoid robots are well-suited for these tasks in environments designed for humans. For complex motions over a large range such as opening a door, a humanoid needs to plan coordinated motions for its lower and upper body parts to successfully open the door without colliding or falling over. Two challenges arise when generating whole-body motions for bipedal humanoid robots in this context. First, motions for a high number of degrees of freedom (DOF) have to be planned, usually for 20 or more joints. Second, a variety of constraints have to be satisfied: The robot must respect its joint limits, avoid self-collisions as well as collisions with obstacles in the environment, and keep its balance. Furthermore, constraints induced

from the objects to be manipulated must be taken into account. For example when opening a door, the robot's end-effector must remain attached to the handle and thus needs to follow its trajectory, in this case an arc.

While we employed search-based methods for footstep planning in the previous chapter, the curse of dimensionality makes a search for all valid joint configurations impractical for the whole body due to the high number of degrees of freedom. Instead, methods that grow random trees in the space of all joint angles, which is called the configuration space, are well-suited for high-DOF systems (LaValle, 2006). Popular approaches are rapidly-growing random trees (RRT) and probabilistic roadmaps (PRM).

We here rely on RRTs and present an extension to the RRT-Connect planner (Kuffner and Lavalle, 2000) that satisfies all required constraints. One focus of our work hereby lies in manipulating articulated objects. Thus, the robot must particularly keep contact with the object handle to manipulate it. The key idea of our approach is to force the hand poses of sampled configurations to follow the trajectory of the object handle. Constraints in the Cartesian or the joint space form a sub-space of all joint configurations, usually referred to as the constraint manifold. Some of these constraints form a manifold of lower dimension than the joint space. This results in a probability of 0 for obtaining a correct configuration by sampling from the manifold intersection, as discussed by Kaiser et al. (2012). To solve this problem, our planner re-projects sampled configurations onto the constraint manifold by adjusting the joint angles of the invalid leg and arm chains with inverse kinematics. A precomputed database of stable double support configurations improves the performance of our planner since these configurations already fulfill most of the constraints.

We present experiments with a Nao humanoid illustrating that our approach can generate solutions to complex whole-body manipulation tasks. We particularly evaluate the manipulation of articulated objects. The experiments demonstrate the extended range of motion for the robot's end-effectors, allowing the Nao robot to successfully reach into shelves, pick up objects, and open doors and drawers.

In contrast to techniques for manipulating articulated objects using wheeled platforms equipped with a manipulator (Chitta et al., 2010; Diankov et al., 2008; Rühr et al., 2012), the planning problem is more complex due to the high number of DOFs and additional constraints regarding the balance of the humanoid system. While Jacobian-based control methods present an alternative approach to generate whole-body motions (Kanoun et al., 2009; Mansard et al., 2007; Yoshida et al., 2006), they usually exhibit problems around obstacles that pose local minima. Randomized sampling-based approaches, on the other hand, have the ability to cope with arbitrary collision environments.

6.1. The RRT-Connect Algorithm

Our system builds upon the RRT-Connect algorithm, which has already demonstrated the ability to efficiently find solutions for planning problems in high-dimensional domains (Kuffner and Lavalle, 2000). The basic idea of RRT-Connect is to grow two search

Algorithm 3: RRT-Connect

Input: Start and goal configurations $\mathbf{q}_{\text{start}}$, $\mathbf{q}_{\text{goal}}$, database of stable configurations DS_DATABASE

Output: A path from $\mathbf{q}_{\text{start}}$ to $\mathbf{q}_{\text{goal}}$ or FAILURE

```
1 T_a.init(q_start)                                  // initialize tree a with start node
2 T_b.init(q_goal)                                   // initialize tree b with goal node
3 for i = 1 to max_iterations do
4     q_rand ← random configuration from DS_DATABASE
5     if extend(T_a, q_rand) ≠ TRAPPED then
          // try to connect the last added node q_new in T_a to T_b
6         if connect(T_b, last(T_a)) =REACHED then
7             return smoothPath(T_a, T_b)
8     swap(T_a, T_b)
9 return FAILURE
```

trees, one from the start and one from the goal configuration. The search trees are iteratively connected by randomly sampling configurations and extending one tree at a time. We first describe RRT-Connect for whole-body motion planning under the constraints of respecting joint limits, avoiding collisions, and maintaining balance. Extensions for manipulation constraints will be described thereafter.

These constraints define a subspace in the search space of all configurations, also referred to as constraint manifold. When randomly sampling configurations during the search, most of them will not lie on the constraint manifold and have to be rejected, e.g., because they cause the robot to lose its balance. This results in a poor sampling coverage of the constrained configuration space. Constraints originating from the robot's geometry and weight distribution are usually not dependent on the task. This allows for computing the constraint manifold — or a sampled approximation — offline. The planner then samples from this precomputed database of configurations to obtain statically stable configurations that are free of self-collisions (Kuffner et al., 2002; Şucan and Chitta, 2012). Other constraints, such as collision avoidance with the environment, are task-specific and need to be validated during planning.

For now, let us assume that we have a database of stable, balanced configurations available. Algorithm 3 lists the RRT-Connect algorithm in pseudocode, with the two functions *extend* and *connect* in Algorithm 4 and 5. Each element of the search trees consists of a vector of the robot's joint angles, called a configuration $\mathbf{q}$. The search space is then called configuration space, as opposed to the Cartesian space in which some of the constraints are described.

As input, the planner takes two collision-free, statically stable double support configurations $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$. While the start configuration usually corresponds to the current

Algorithm 4: Extend step of the RRT

Input: Search tree $\mathcal{T}$, reference configuration $\mathbf{q}_{\text{rand}}$
Output: $\mathcal{T}$ is extended towards $\mathbf{q}_{\text{rand}}$, return status of expansion

```
1  Function extend(T, q_rand)
2      q_near ← findNearestNeighbor(T, q_rand)
3      q_new ← newConfig(q_rand, q_near)      // new configuration at distance ε from q_near
4      if q_new is valid then                 // check constraints and collisions
5          T.addNode(q_new)
6          T.addLink(q_near, q_new)
7          if q_new = q_rand then
8              return REACHED
9          else
10             return ADVANCED
11     else
12         return TRAPPED
```

Algorithm 5: Connect step of the RRT

Input: Search tree $\mathcal{T}$, configuration $\mathbf{q}$
Output: $\mathcal{T}$ is connected to configuration $\mathbf{q}$, return status of expansion

```
1  Function connect(T, q)
2      repeat
3          S ← extend(T, q)
4      until S ≠ ADVANCED
5      return S
```

state of the robot, the goal configuration is generally not known in advance but has to be generated so that it also respects the task constraints. We will describe the goal generation in Section 6.3.

After initialization, two search trees $\mathcal{T}_a$ and $\mathcal{T}_b$ are grown from $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$. At each iteration, the algorithm samples a random statically stable configuration $\mathbf{q}_{\text{rand}}$ from the precomputed constraint manifold database DS_DATABASE in line 4. First, it extends $\mathcal{T}_a$ towards $\mathbf{q}_{\text{rand}}$ as reference configuration.

The extension of a search tree $\mathcal{T}$ towards the reference configuration $\mathbf{q}_{\text{rand}}$ is described in Algorithm 4. The method first finds the nearest neighbor $\mathbf{q}_{\text{near}}$ of $\mathbf{q}_{\text{rand}}$ in $\mathcal{T}$, and then generates a new configuration $\mathbf{q}_{\text{new}}$ at a fixed distance ε from $\mathbf{q}_{\text{near}}$ towards $\mathbf{q}_{\text{rand}}$ in the configuration space (line 3). This process is illustrated in Figure 6.1. The method then checks whether $\mathbf{q}_{\text{new}}$ satisfies all required constraints, adds the new node to $\mathcal{T}$ if that is the case, and returns the status of the tree extension.

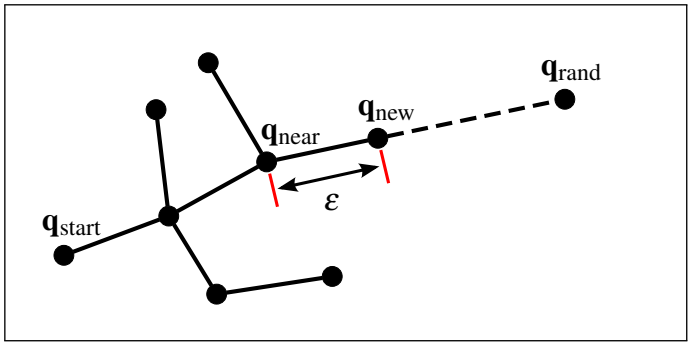


Figure 6.1.: Example of an RRT expansion step. From $\mathbf{q}_{\text{near}}$, $\mathbf{q}_{\text{new}}$ is expanded at distance ε towards $\mathbf{q}_{\text{rand}}$.

The validation in line 4 of Algorithm 4 checks whether the joint angles are within their limits, whether the robot mesh model with configuration $\mathbf{q}_{\text{new}}$ collides with itself or the environment, and whether the configuration keeps the robot balanced.

Should the extend function return TRAPPED due to a constraint being invalidated, Algorithm 3 proceeds by swapping the two trees, i.e., tree $\mathcal{T}_a$ becomes $\mathcal{T}_b$ and vice versa. In the next iteration, a new random configuration is sampled and the other tree $\mathcal{T}_a$ is expanded. Otherwise the extend function found a valid configuration and added it to the search tree. In that case Algorithm 3 tries to connect $\mathcal{T}_b$ towards the newly added node in line 6 by repeatedly extending $\mathcal{T}_b$. If this expansion reaches the newly added node, the algorithm connected the two search trees and thus found a valid path.

This procedure is repeated until a path has been found, i.e., the two trees are connected, or a maximum number of iterations has been reached. When the search was successful, the solution path is smoothed and cut short by removing extraneous nodes while preserving the constraints in line 7 of Algorithm 3. Note that RRT-Connect, like most randomized planners, is probabilistically complete. That means that if a valid path exists, RRT-Connect will find it with a probability approaching 1 as the number of iterations approaches infinity.

6.2. Precomputing Stable Configurations

As introduced by Kuffner et al. (2002), we use a precomputed set of stable whole-body configurations as constraint manifold approximation, from which the planner then draws samples (Şucan and Chitta, 2012). Samples drawn from this database are guaranteed to satisfy the constraints. Our approach builds this database by sampling random configurations of joint angles that are within the robot's joint limits, modifying them to be in double support mode, and finally verifying whether they are stable and free of self-collisions.

For the purpose of this work, we aim at storing stable double support configurations in the database. This requires that both feet of the robot remain on the supporting ground

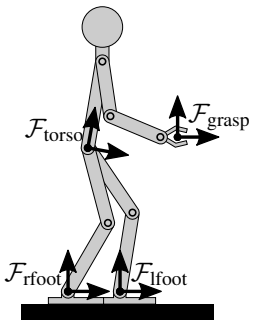


Figure 6.2.: Coordinate frames for whole-body manipulation.

plane and that the posture is balanced such that the robot does not fall. In general, the zero moment point indicates a humanoid's dynamic stability (Vukobratovic and Borovac, 2004). In this work, we use the simplification of static stability, which is a valid approximation for slow motions. Thus, stability can be evaluated by checking whether the robot's center of mass (COM) projected to the ground plane is within the support polygon formed by the two feet. In practice, we scale down the actual support polygon by a small safety margin to avoid the critical boundary areas, thus accounting for inaccurate motion execution and a mismatch between the real robot and its CAD model.

Since the probability of obtaining a double-support configuration just from random sampling is close to zero, we need to re-project the random samples onto the constraint manifold. In our case, this can be achieved by adjusting the leg joint angles such that both feet remain on the same supporting plane a certain distance apart. To do so, our system chooses the frame of the right foot $\mathcal{F}_{\text{rfoot}}$ as the root of the kinematic model and adapts the joint angles of the left leg to reach the desired pose for the left foot. Note that the choice of the foot frame is arbitrary in double support mode. Figure 6.2 illustrates the necessary coordinate frames and their relation.

From forward kinematics of the right leg chain, we can obtain the closest frame common to both legs, the hip or torso frame $\mathcal{F}_{\text{torso}}$. By means of 3D coordinate transformation, we can express the desired coordinate frame of the left foot $\mathcal{F}_{\text{lfoot}}$ with respect to $\mathcal{F}_{\text{torso}}$, and solve that end-effector pose with IK for the left leg chain. We hereby rely on a numerical solution, using the right leg joint values as an initial guess. Finally, if a solution exists, we create a modified whole-body configuration that consists of the IK solution for the left leg and the sampled joint angles for the remainder of the body. We add this configuration to the database if it is free of self-collisions and statically stable. Otherwise it is discarded.

Generally, building this database of statically stable configurations needs to be performed only once for a robot since it is independent of the planning scenario or environment. Sets for left and right single-leg support can be constructed in a similar fashion, without adjusting the other leg.

6.3. Generating the Goal Configuration

When planning whole-body motions for manipulation, the goal configuration not only has to respect all constraints inherent to the planning process, but the end-effector pose for grasping also depends on the object to be manipulated. For example, if a handle has to be grasped for opening a door, then the end-effector has to align with the handle prior to grasping.

To this end, our goal generation first computes a set of valid double support configurations in a similar way as the sampling process for building the database in the previous section. The difference here is that we adapt the arm configuration as explained in the following. As an alternative, we could also sample a configuration from the precomputed database. However, the database is usually sparsely-populated, which is sufficient to provide guiding reference configurations for tree expansion but may not contain enough variety to provide a valid goal configuration in a certain scenario.

We assume that the grasping goal, e.g., for grasping a handle, is given as a 3D world coordinate with one degree of freedom unconstrained around the z-axis of the object handle. The goal denotes the desired location for the tip of the chosen hand $\mathcal{F}_{\text{grasp}}$ for grasping. From a generated double-support configuration, we first compute the grasping goal with respect to the torso frame $\mathcal{F}_{\text{torso}}$. This allows us to compute a 5 DOF solution for the arm chain in closed form with IK. Generally it is also possible to solve IK for a full 6D hand pose. However, allowing one degree of freedom around a handle bar allows a larger set of possible solutions in our application.

The analytic IK computation may result in a number of solutions for arm configurations $\mathbf{q}^{\text{arm}}$. We select one of them by means of an evaluation function

$$\text{eval}(\mathbf{q}^{\text{arm}}) = q^{\text{arm}}[0] \times |\det(J(\mathbf{q}^{\text{arm}}))| \,. \tag{6.1}$$

Here, $\det(J(\mathbf{q}^{\text{arm}}))$ is the determinant of the Jacobian associated with the arm configuration and denotes a measure of manipulability based on the distance from singularities (Yoshikawa, 1985). We combine it with the shoulder pitch angle $q^{\text{arm}}[0]$ to prefer elbow-down configurations which are considered to be more natural than elbow-up configurations. The best IK solution maximizing Eq. (6.1) is then assigned to the arm joints of the initially sampled double support configuration to obtain a whole-body goal posture that can reach the given grasping goal.

Figure 6.3 shows a set of example goal configurations for a desired grasp frame position and z-axis direction with a Nao humanoid. Currently, our planner uses the overall best whole-body configuration as goal. In general, one can think of rooting multiple trees at the generated goal configurations and initializing different RRT searches in parallel from which the best solution is chosen afterwards.

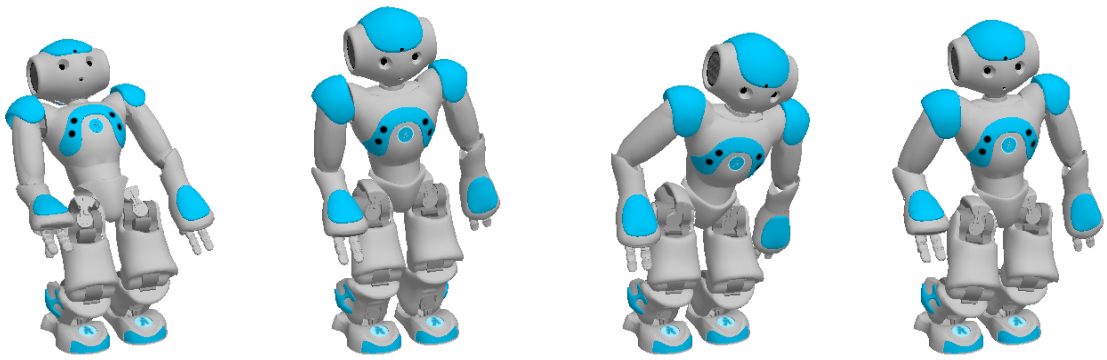


Figure 6.3.: Examples of valid whole-body goal configurations for a given pose of the robot's right hand. The feet of the robot remain fixed and the 5D grasping goal for the hand relative to them is identical in all configurations. The left arm is in a safe configuration adjacent to the robot's body.

6.4. RRT-Connect for Manipulating Articulated Objects

Based on the whole-body RRT-Connect planner introduced in the previous sections, we now describe our extensions for constraints originating from the manipulation of articulated objects. We hereby assume that the robot already grasped the object handle. To ensure contact with the articulated object, we enforce the hand poses to follow the trajectory of the object handle while planning.

Articulated objects are represented by a volumetric model, a joint with the corresponding position, and a handle position. From this, a generative model can predict the trajectory of the articulated parts and the handle while the object is being manipulated. Articulation models can be learned from previous experience while carefully manipulating the environment (Sturm et al., 2011). For the scope of this work, we assume the model to be given. We are particularly interested in prismatic and revolute models, which correspond to drawers and doors respectively, and are illustrated in Figure 6.4. A prismatic model has one degree of freedom as a translation along a fixed axis. The handle trajectory is constrained on a 3D vector denoting the opening direction. A revolute model has one degree of freedom around a specified 3D axis of rotation, its hinge. The handle trajectory is constrained on the circular arc described by the axis of rotation and the radius of the handle position.

Our extended RRT-Connect algorithm that also considers manipulation constraints is shown in Algorithm 6. Compared to the default RRT-Connect in Algorithm 3, this version uses the articulated object parameterization as optional input. We will now describe the differences concerning tree initialization and extension.

Given a start configuration $\mathbf{q}_{\text{start}}$ with a hand attached to the object handle and the articulated object parameters, our approach first computes a trajectory of hand poses $\mathcal{H} = \langle \mathbf{h}_0, \ldots, \mathbf{h}_k \rangle$ in line 4. The hand pose $\mathbf{h}_0$ corresponds to $\mathbf{q}_{\text{start}}$, while $\mathbf{h}_k$ corresponds to $\mathbf{q}_{\text{goal}}$. Figure 6.5 illustrates examples for a linear and circular object trajectory with the generated hand poses.

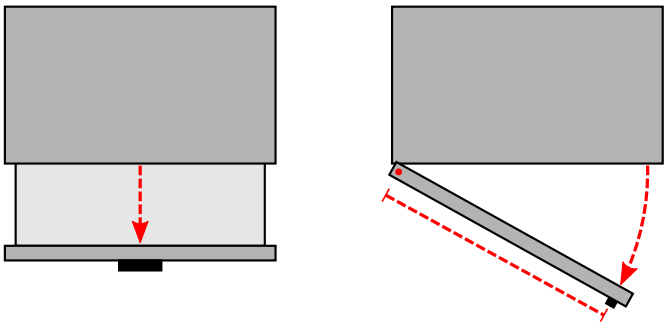


Figure 6.4.: Two examples of articulated objects: a drawer and a door.

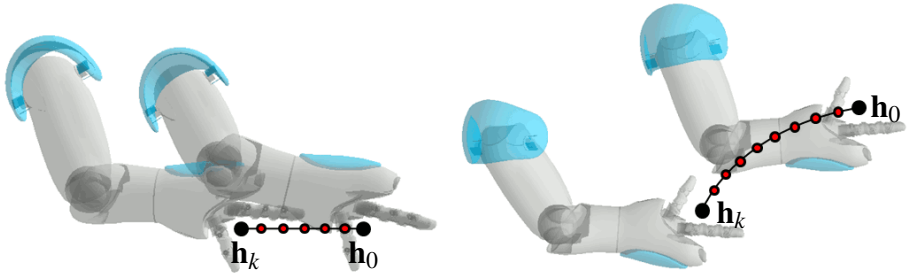


Figure 6.5.: Example end-effector trajectories as desired hand poses for opening a drawer (left, side view) and a door (right, top view).

During manipulation, the grasping hand is required to remain on the object handle trajectory for each new configuration generated during tree expansion. Thus, the trajectory $\mathcal{H}$ is used as a parameter for modified *extend* and *connect* functions. The new *connect* function simply passes $\mathcal{H}$ on to *extend*. The new *extend* function uses $\mathcal{H}$ to adjust $\mathbf{q}_{\text{new}}$ after it is generated such that it remains valid, thus projecting the configuration back onto the new constraint manifold. Our extended RRT-Connect implementation assigns a discrete index in $\mathcal{H}$ to each configuration in the search trees, starting with 0 at $\mathbf{q}_{\text{start}}$ and k at $\mathbf{q}_{\text{goal}}$. When expanding the starting tree, this so-called hand pose index increases, while it decreases when expanding the goal tree. This procedure ensures that the hand poses used during tree expansion properly follow the object trajectory.

To obtain the correct joint angles for a given hand pose and sampled whole-body configuration, our approach again uses analytic IK for the arm as in Section 6.3. In the set of all IK solutions $\mathcal{I}$ found for the hand pose, our system selects $\hat{\mathbf{q}}^{\text{arm}}$ that minimizes the configuration space distance to the arm joint angles of the closest configuration $\mathbf{q}_{\text{near}}$:

$$\hat{\mathbf{q}}^{\text{arm}} = \underset{\mathbf{q}_i^{\text{arm}} \in \mathcal{I}}{\arg\min} \left\| \mathbf{q}_i^{\text{arm}} - \mathbf{q}_{\text{near}}^{\text{arm}} \right\| . \tag{6.2}$$

In $\mathbf{q}_{\text{new}}$, we then adjust the arm joint angles to the values of $\hat{\mathbf{q}}^{\text{arm}}$ and thus obtain a whole-body configuration that respects all constraints.

Algorithm 6: Extended RRT-Connect with constraints from manipulated objects

Input: Start and goal configurations $\mathbf{q}_{\text{start}}$, $\mathbf{q}_{\text{goal}}$, database of stable configurations DS_DATABASE, *object_params* of manipulated object (optional)
Output: A path from $\mathbf{q}_{\text{start}}$ to $\mathbf{q}_{\text{goal}}$ or FAILURE

```
1  T_a.init(q_start)                                  // initialize tree a with start node
2  T_b.init(q_goal)                                   // initialize tree b with goal node
3  if articulated object grasped then
4      H ← computeHandTrajectory(q_start, object_params)
5  else
6      H = ∅
7  for i = 1 to max_iterations do
8      q_rand ← random configuration from DS_DATABASE
9      if extend(T_a, q_rand, H) ≠ TRAPPED then
10         if connect(T_b, last(T_a), H) = REACHED then
11             if articulated object grasped then
12                 return path(T_a, T_b)
13             else
14                 return smoothPath(T_a, T_b)
15     swap(T_a, T_b)
16 return FAILURE
```

To ensure following the hand trajectory $\mathcal{H}$, the two search trees can no longer be connected between their two closest nodes as before. Instead, configurations may only be connected if their configuration space distance is below ε and their hand pose indices are adjacent, as illustrated in Figure 6.6.

The final difference of our extended RRT-Connect to the standard implementation concerns smoothing the found path. When manipulating an articulated object, this is no longer admissible. Since the smoothing is performed in the configuration space, it could cause a violation of the manipulation constraints in Cartesian space. Nevertheless, the enforcement of manipulation constraints already ensures that the plans for the end-effector directly follow the trajectory of the articulated object, and thus are reasonably smooth in the Cartesian space.

6.5. Implementation Details

Our planner is implemented in the *MoveIt!* framework in ROS (Chitta et al., 2012b). We use FCL (Pan et al., 2012) to check for collisions of the robot with itself and the environment, based on a collision mesh model of each robot link. To analytically solve

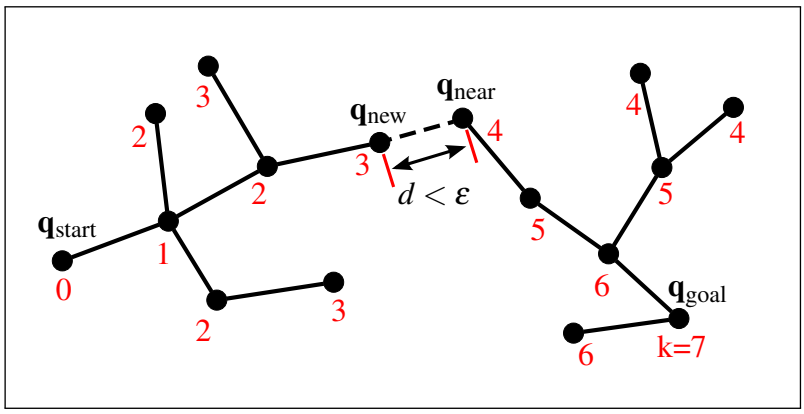


Figure 6.6.: Connection between the two random trees when manipulating an articulated object. The start tree grows from the left and the goal tree from the right. Numbers denote the hand pose indices associated to each node. A connection between the two trees can be established when two nodes with adjacent hand pose index can be connected.

IK, we rely on *IKfast* (Diankov, 2010). The 5D grasping goals along a handle hereby directly correspond to the 5-DOF arm of the Nao humanoid we used in the experiments.

To approximate the constraint manifold for planning whole-body motions, we used a database of 463 statically stable double support configurations, generated while sampling for 10 000 iterations. The success rate of only 4.63% demonstrates the low probability of generating valid configurations when the configuration space is sampled completely at random. For generating goal configurations, we allow a maximum number of 3000 iterations. For efficiency, we stop searching when more than six goal configurations have been found and choose the best one according to Eq. (6.1). We used a maximum number of iterations for RRT expansion of 3000 and an expansion step width of $\varepsilon = 0.1$.

6.6. Evaluation

We evaluated our approach with a V4 Nao humanoid. The robot and its kinematic model is described in detail in Appendix A. Our planner operates on 21 of the total 25 DOF since we do not articulate the head joints and use the hands only when grasping. Upon reaching an object to grasp, such as a handle, the hand closes around it and it opens again for releasing the object.

Inertia, mass, and center of mass for each robot link are known from its CAD model. For efficient collision checks, we created a low-vertex collision mesh model. In the following experiments, we planned off-board on a single core of a standard desktop CPU (Intel Core 2 Duo, 3 GHz).

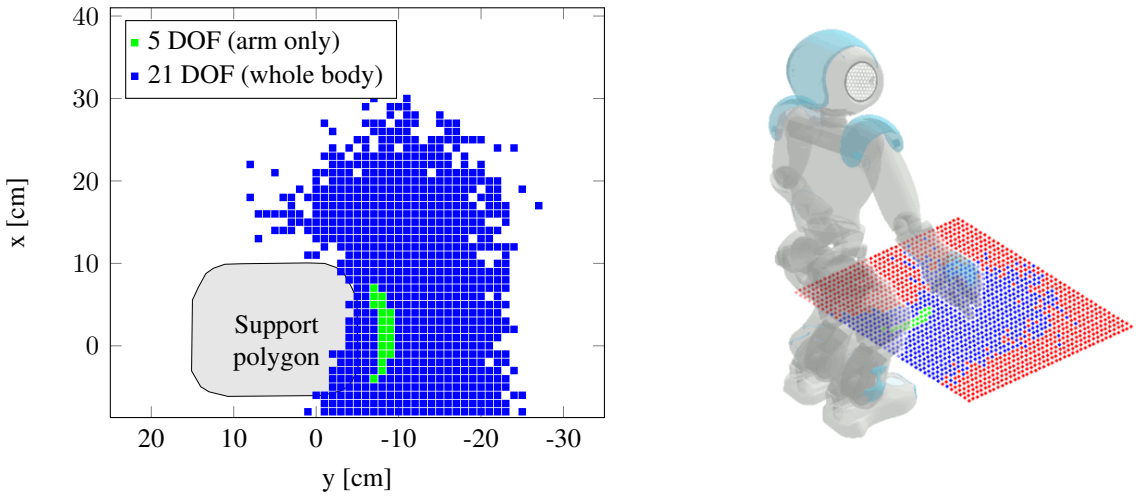


Figure 6.7.: Reachability map of the right hand for 5D end-effector goals at height $z = 20\,\text{cm}$ as 2D projection (left) and relative to the robot model (right). Green squares denote success for arm-only and whole-body planning, while blue squares are only reachable with whole-body planning.

6.6.1. Reachability Analysis

In the first experiment, we evaluated the extended reachability when using our whole-body planning approach in fixed double support mode. The robot had to plan stable whole-body motions to reach a 5D manipulation goal with its right end-effector, similar to the goals depicted in Figure 6.3. The z-axis of the grasp frame was horizontal and perpendicular to the robot's orientation at a height of $z = 20\,\text{cm}$, e.g., to grasp the handle of a drawer. We varied x and y in intervals of 1 cm in an area of $40 \times 40\,\text{cm}^2$. Figure 6.7 shows the resulting reachability map, in which blue and green squares denote successful results for both goal generation and planning. The goal generation failed for the other locations, indicating that they are not reachable. When planning with the 5-DOF arm instead of the whole body, only a small subset of all locations is reachable due to the kinematics. This demonstrates that whole-body planning clearly extends the manipulation range around the robot, particularly compared to planning only for the arm.

6.6.2. Planning Collision-Free Motions

In a set of different planning scenarios, we evaluated several aspects of our whole-body planning approach. These scenarios present different challenges to the planner. For evaluation, we attempted 100 motion plans for each scenario. The results are summarized in Table 6.1. In all scenarios, we supplied the goal location as well as the parameters of articulated objects, if present, to the planner.

Scenario	Goal generation [s]	Planning time [s]	Expanded nodes
Shelf (upper)	16.4±13.74	0.09±0.27	19.84±30.06
Shelf (lower)	19.06±16.75	10.44±0.83	1164.89±98.99
Door (reach)	8.59±7.54	0.09±0.05	32.16±9.43
Door (open)	2.63±2.47	0.15±0.08	35.17±14.51
Drawer (reach)	5.42±5.17	0.08±0.04	32.81±8.41
Drawer (open)	4.14±3.94	0.11±0.07	30.36±12.69
Drawer w. obstacle (reach)	16.24±12.64	7.71±3.00	1067.93±333.54
Drawer w. obstacle (open)	8.53±7.21	0.2±0.19	32.16±21.09

Table 6.1.: Performance of whole-body planning in different scenarios as mean and standard deviation of 100 planning attempts.

Collision-Free Reaching and Grasping

In the first scenario we evaluated the performance of the planner in the presence of obstacles constraining the possible motions. Here, the robot needed to reach into one shelf of a cabinet and then into a second one below without colliding. A resulting plan execution is shown in Figure 6.8. With our chosen parameters for the algorithm, the entire task of reaching the grasping goals located inside the upper and lower shelf could be planned successfully in 77% of all attempts. The comparably long times for goal generation and planning resulted from the highly constrained workspace. In particular the second plan required a high number of expanded nodes, since it had to plan from one cavity into another below. For this, the robot first had to move its manipulator out of the first shelf and then into the second. Such a scenario can be problematic for Jacobian-based optimization techniques due to the local minima induced by the cavities.

Figure 6.9 shows an exemplary application to a pick and place task, in which the robot had to pick up a small object and displace it into a container. After picking up the object, its 3D mesh model becomes attached to the robot's end-effector for checking collisions with the environment. For the first plan to reach the small green box, our algorithm required 3.28 s to generate a goal configuration and 0.37 s for the whole-body motion plan, hereby expanding 62 nodes. For the second plan to place the box over the basket, our algorithm generated a goal configuration within 9.16 s and a motion plan within 0.18 s, expanding 38 nodes.

Manipulating Articulated Objects

The next scenarios demonstrate the manipulation of a door and a drawer. From its initial configuration, the robot first had to plan for reaching the known handle location, grasp it, and then plan an end-effector trajectory given by the parameterization of the articulated object. Figure 6.10 shows the generated start and goal configurations, while Figure 6.11 and 6.12 show the resulting whole-body motion plans. The robot successfully grasped the

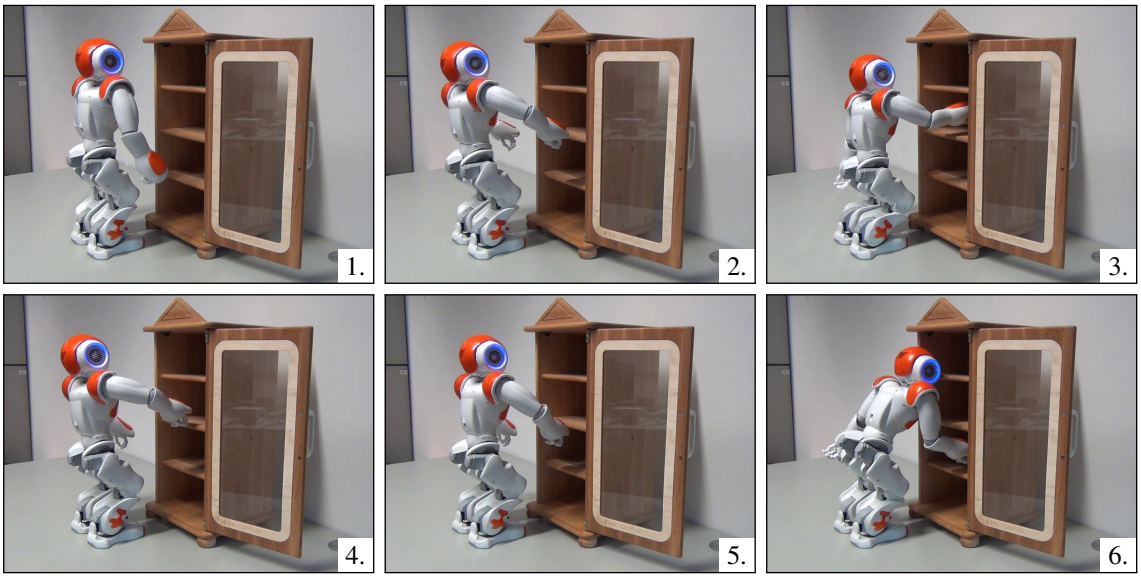


Figure 6.8.: Execution of a whole-body plan for reaching collision-free into different shelves of a cabinet.

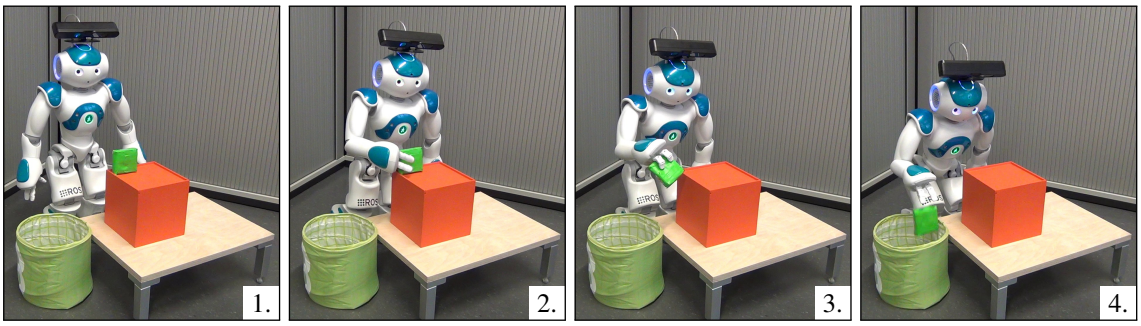


Figure 6.9.: Execution of a whole-body plan for picking up a small object and displacing it into a container.

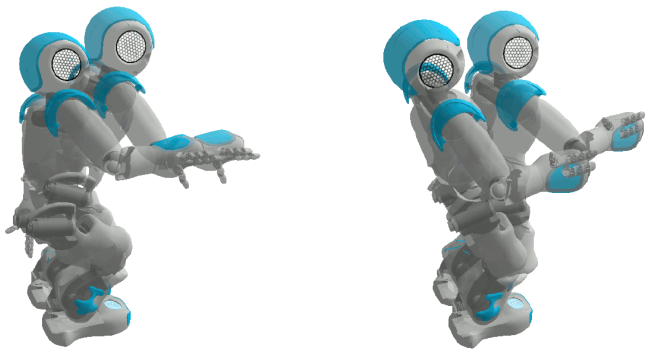


Figure 6.10.: Start and goal configuration for opening a drawer (left) and a door (right).

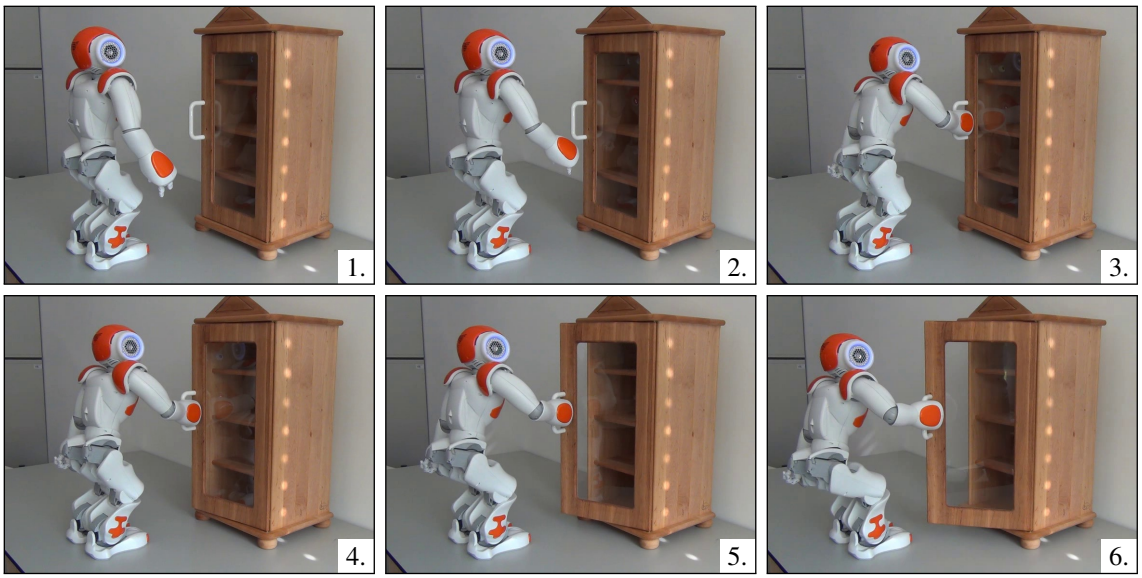


Figure 6.11.: Execution of a whole-body manipulation plan for opening a door. The robot first plans a motion to grasp the handle and then a motion to open the object.

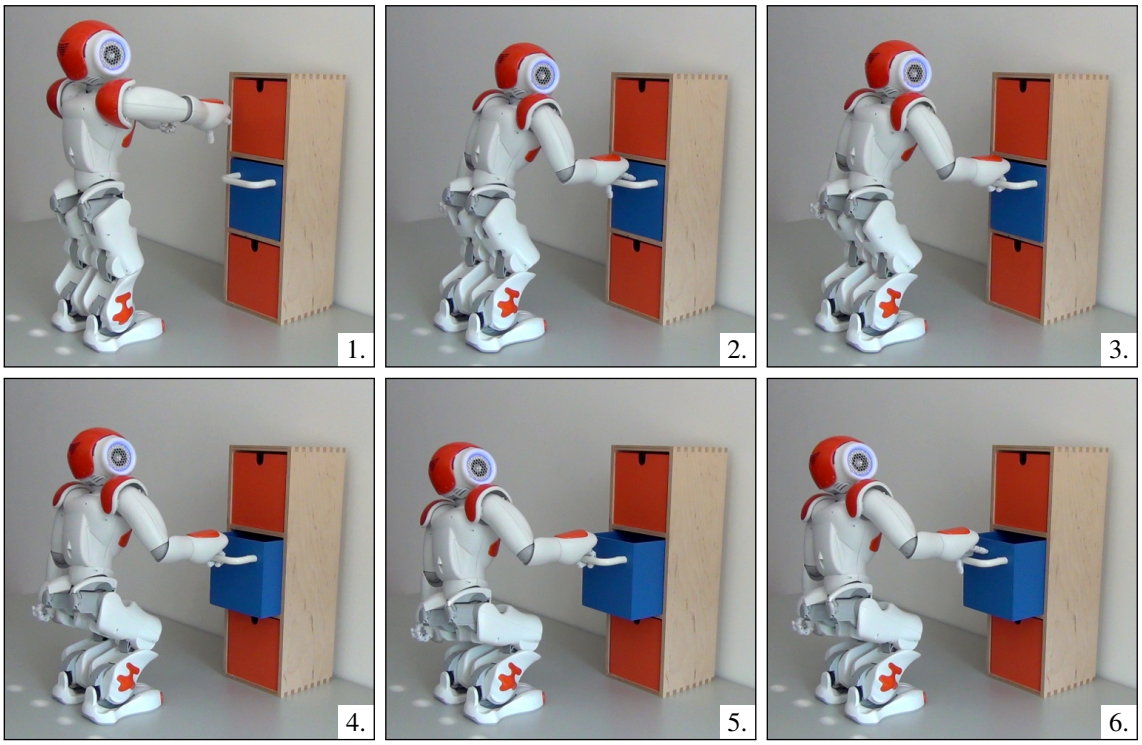
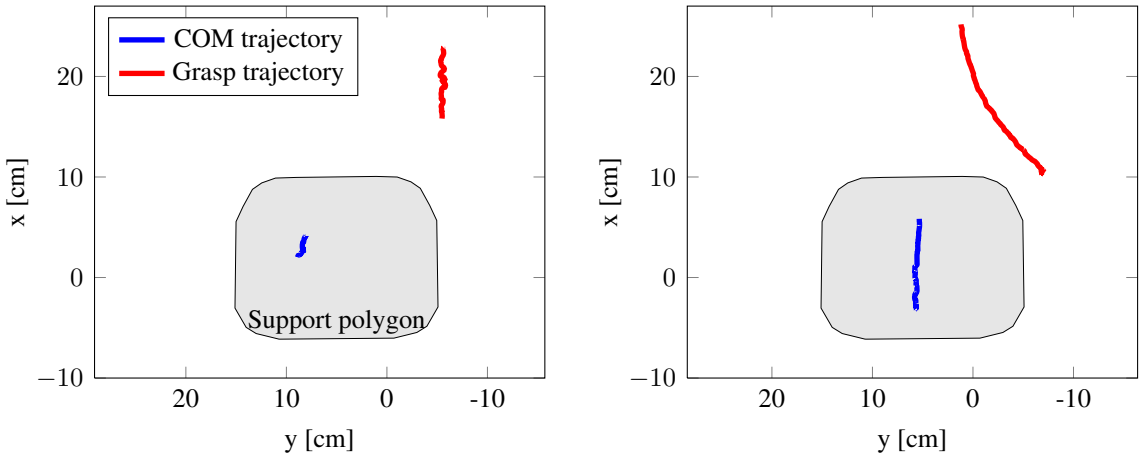


Figure 6.12.: Execution of a whole-body manipulation plan for opening a drawer. The robot first plans a motion to grasp the handle and then a motion to open the object.



(a) Opening a drawer **(b)** Opening a door

Figure 6.13.: Trajectory for the right hand and center of mass (COM) over the support polygon while opening a drawer and a door.

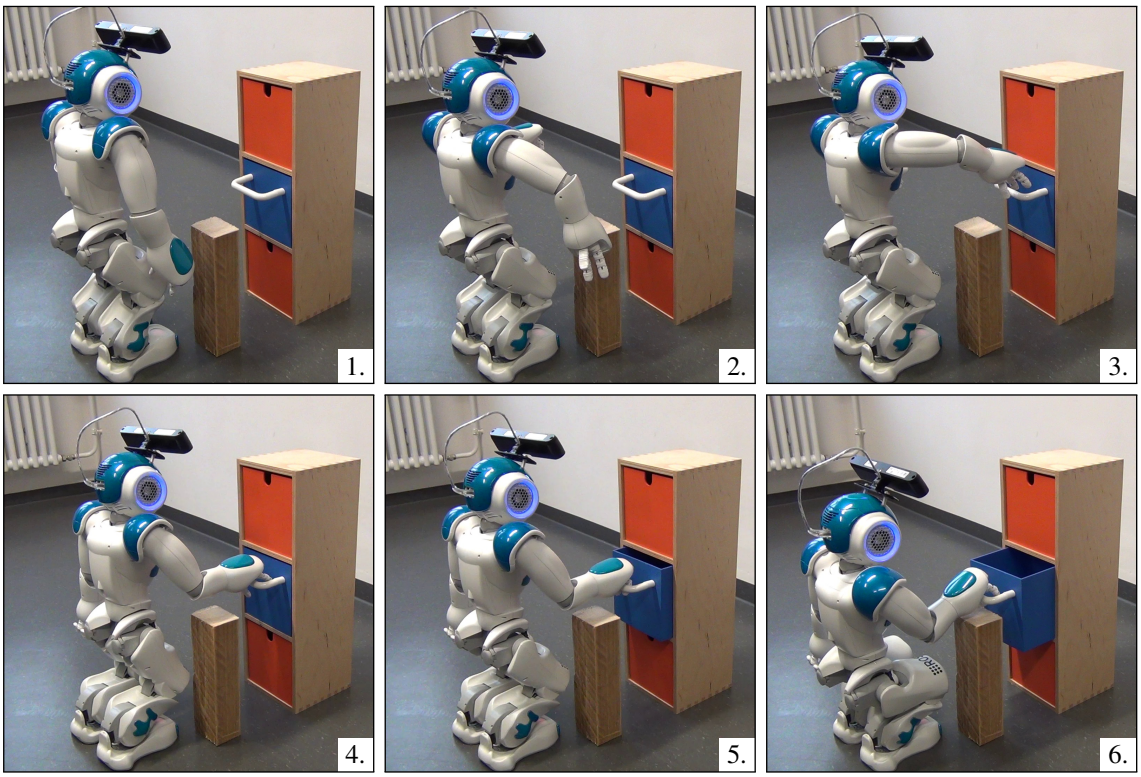


Figure 6.14.: Execution of a whole-body manipulation plan for opening a drawer with an added obstacle.

handle and opened the object while keeping its balance. Figure 6.13 shows the trajectories of the right hand and COM, as measured by the joint angle sensors during execution. As can be seen, our approach leads to hand trajectories that closely follow the given model, while the COM remains safely within the support polygon. With the chosen parameters of our algorithm, planning for the entire manipulation task could be solved successfully in 89% of all attempts for the drawer, and 78% for the door.

In a last scenario, the robot had to avoid an additional obstacle while opening the drawer, as shown in Figure 6.14. The added obstacle increased the computation time for both goal generation and planning, since it is placed close to the robot and the robot had to plan around it. Planning the entire task succeeded in 83% of the runs.

The success rates for planning certainly increase as more planning time is allowed in the form of a higher number of maximum iterations. All planned motions were executed on the real Nao platform multiple times collision-free and without falling. Videos of all motions are available online at http://www.youtube.com/watch?v=VUmnFy58w90.

6.7. Related Work

Most approaches to generate whole-body motions for complex, high-DOF robots can be classified into either Jacobian-based methods or randomized, sampling-based motion planning techniques. Although approaches belonging to the former category rather deal with online motion generation, many of their concepts can be inherited for planning motions offline.

Using generalized IK with task-specific Jacobians, joint velocities that minimize given task functions can be iteratively computed. For example, Kanoun et al. (2009) plan local motions for a humanoid robot based on the prioritized kinematic control scheme. The authors define multiple tasks, represented by linear equality and inequality systems, which can be organized in an arbitrary order of priority. They successfully applied the approach on the humanoid robot platform HRP-2 for the task of reaching a ball underneath an object. The authors have recently extended the approach to locomotion by modeling footsteps as a chain of rotational and prismatic joints in an extended virtual kinematic structure (Kanoun et al., 2011). This approach, however, bears the risk of the virtual chain being trapped in local minima, as generally encountered with numerical optimization methods. Narrow environments with cavities particularly bear the risk of local minima when planning collision-free motions.

Yoshida et al. (2006) presented an approach that couples generalized IK with a dynamic walking pattern generator, also based on prioritized tasks. While generating whole-body motions, several criteria such as manipulability and joint limits are monitored. As soon as the task is detected to be unfeasible, new foot locations are computed to extend the robot's reachable space. The stepping motion itself is provided by the walking pattern generator. Mansard et al. (2007) proposed using visual servoing with whole-body control for a humanoid robot. The authors split the control scheme into several sensor-based

control tasks that are executed simultaneously. Tasks are sorted in increasing order of priority and can be removed or added during execution. With control at the task level, the HRP-2 was able to grasp an object while walking. However, avoiding obstacles and self-collisions has not been taken into account.

As an alternative to Jacobian-based control methods, probabilistic roadmaps (Kavraki et al., 1996) and rapidly-exploring random trees (RRTs) are proven methods to plan collision-free paths in high-dimensional configuration spaces (LaValle, 2006). Kuffner et al. (2002) first employed probabilistic motion planning to generate whole-body motions for humanoids. The authors presented a variant of the RRT-Connect planner and introduced a new sampling strategy. Since random samples in the configuration space are unlikely to be valid with respect to the robot's stability, their planner resorts to a pre-computed set of statically stable configurations in the tree expansion process. We extended this approach for the task of manipulating articulated objects, in which the object parameterization constrains the end-effector trajectory.

Hauser et al. (2005) presented an approach for non-gaited locomotion of humanoids based on PRM, with a focus on the constraint of maintaining balance. In their approach, the authors employ numerical IK during sampling for an iterative constraint enforcement. This enabled planning whole-body motions for climbing ladders or rough terrain in simulation, using hands and knees as contact points.

Similar to our approach, Stilman (2010) presented an approach to cope with task constraints as restrictions on the end-effector motion. The local planner used in the tree expansion process is enhanced by projecting the sampled configurations on a sub-manifold that solves the task. The technique was applied to a manipulator on a mobile base opening doors and drawers, without considering stability constraints. Berenson et al. (2009) follow a similar approach. In their work, they introduce an extension to the Constrained Bidirectional RRT planner that considers constraints as task space regions (TSRs), a concept previously used for goal specification. To describe more complex constraints, these TSRs can be chained together. The authors' experiments demonstrate efficient planning times for motion planning under stability, manipulation and closure constraints for a humanoid robot.

A similar algorithm called Constellation was recently proposed by Kaiser et al. (2012) to plan reaching motions for a bipedal humanoid. It requires constraints formulated as Jacobians for TSRs, for collision avoidance, and for maintaining balance. Starting from multiple random configurations, the algorithm iteratively searches the configuration space for an intersection of all constraint manifolds, which then corresponds to a solution that satisfies all constraints. The authors applied their method to a simulated bipedal robot.

Kanehiro et al. (2012) presented an approach composed of a planning and execution phase. A coarse plan is quickly generated with RRT-Connect and analytical IK. This plan is executed online while compensating approximation errors in realtime and maintaining constraints. Although capable of reaching a target object, the approach does not consider a subsequent manipulation of the object at this stage. Oriolo and Vendittelli (2009) proposed a control-based approach to task-constrained motion planning. Unlike previous

methods, their RRT planner makes use of a Jacobian-based motion generation scheme that allows for continuous constraint satisfaction. Closure or stability constraints were not considered.

Dalibard et al. (2009) combined local Jacobian-based methods with randomized motion planning for humanoids. Their local planner employs a prioritized pseudo-inverse technique (Yoshida et al., 2006) to generate goal configurations prior to planning and to project randomly generated samples onto the constraint manifold. In a later work, the authors planned door opening actions with a humanoid in simulations (Dalibard et al., 2010). The global planner used an RRT to plan for the three DOF of the robot's bounding box. El Khoury et al. (2013) recently proposed another extension that used the result of the randomized planner as a seed for a subsequent optimal control algorithm, thus providing a good initial guess for avoiding local minima in the optimization. The authors successfully planned fast motions for avoiding self-collisions and for reaching into a shelf with the HRP-2 humanoid. The comparably long optimization times currently prevent the applicability of the approach to online planning or reactive motions.

Nozawa et al. (2012) integrated a constrained end-effector trajectory into the walking controller of the humanoid HRP-2. This is suitable for pushing objects such as carts or opening a door while walking. It is, however, unclear how collision avoidance can be integrated into the control scheme.

6.8. Conclusion

6.8.1. Summary

We presented an approach for whole-body motion planning particularly suited for the manipulation of articulated objects. The challenges in this setting are the high number of degrees of freedom that have to be planned and various constraints arising from the task: The humanoid needs to keep its balance, respect its joint limits, and avoid collisions with itself and the environment. Additionally, when manipulating an articulated object, the robot's end-effector must follow the grasped handle trajectory.

We based our approach on the RRT-Connect planner, which grows two random trees and tries to connect them from the start to the goal configuration. Our planner hereby relies on a precomputed approximation of the constraint manifold that contains valid double support configurations. Based on the models of the articulated objects, our approach generates hand poses of sampled configurations that follow the object handle trajectory. The planner then uses IK to reach the given hand pose from a sampled configuration. In a similar way, we generate the goal configuration of the plan by combining a sampled statically stable configuration with IK to satisfy the constraints.

As we demonstrated in the experimental evaluation with a real Nao humanoid, our planner is able to solve a variety of motion planning tasks that involve reaching into shelves, opening doors and drawers, as well as picking up objects. Whole-body motion

planning hereby greatly extended the reachable range of the robot compared to using only its 5-DOF arm. The planning queries could be answered within a few seconds, while scenarios involving a challenging obstacle placement took longer due to the collision checks. A major part of the time was used to generate the goal configuration, which offers a promising area of research for future improvements.

6.8.2. Future Work

Our work offers a basis for numerous further extensions in future work. The aforementioned goal generation could be improved by sampling from a modified double support database and adjusting some of the joint angles with IK. The configurations in the database should be universal enough to provide valid goal configurations. Task-space regions as used by Berenson et al. (2009) could provide an alternative method of specifying constrained goal configurations.

Integrating the perception of the environment enables interactively building an environment representation for collision checking, e.g., with the approach presented in Chapter 2. Detecting the object handle or grasping point will provide the position to the planner and feedback while grasping, which allows to correct for initial localization errors and an inaccurate execution. Combining object detection and grasp selection as Müller et al. (2012) could enable the robot to grasp more different objects and handles. Without prior knowledge of the articulated object parameterization, i.e., its DOF and hinge location, the robot could grasp the handle and carefully learn the trajectory through manipulation as proposed by Sturm et al. (2011).

In this work, we considered the robot to remain in double support mode. An even greater manipulation range can be expected when the humanoid is able to plan for changing its COM, and thus its stance leg. This could be even combined with a footstep planning approach from Chapter 5, resulting in the ability to cope with true mobile manipulation tasks.

Chapter 7

Navigation in Three-Dimensional Cluttered Environments

In this chapter, we present a fast, integrated approach to solve path planning in three-dimensional cluttered environments using a combination of the efficient 3D environment representation introduced in Chapter 2 and an anytime search-based motion planner. For collision checking, our approach relies on the 3D representation with an additional multi-layered 2D environment representation to improve planning speed. This allows the generation of almost realtime plans with bounded sub-optimality. Experiments with the two-armed mobile manipulation robot PR2 carrying large objects through demanding passageways in a cluttered environment demonstrate the efficiency and practicability of the presented approach.

Home environments are a major area of interest for personal robotics. Current research explores how robots can ease daily chores or provide assistive care in these environments for an aging society (Chen et al., 2013; Graf et al., 2004, 2009; Mertens et al., 2011; Pollack et al., 2002; Yamazaki et al., 2012). For example, a robot could fetch and process laundry for the user, deliver medication, or clean up a room. Most of these tasks require combined manipulation and navigation abilities of the robot, which is called mobile manipulation. However, home environments pose a number of challenges for robotic navigation. Compared to most industrial settings, these environments are highly unstructured and dynamic. To enable fast planning times, current state of the art navigation approaches consider the 2D position and orientation of a robot base in a 2D grid map (Marder-Eppstein et al., 2010). In these approaches, sensor data is either two-dimensional or projected down to 2D from 3D and a projected footprint is typically used to plan for the robot on a 2D grid map. The downside of these approaches is that goal locations where the 2D projection of the robot is in collision in the 2D map cannot be reached, even if the 3D configuration of

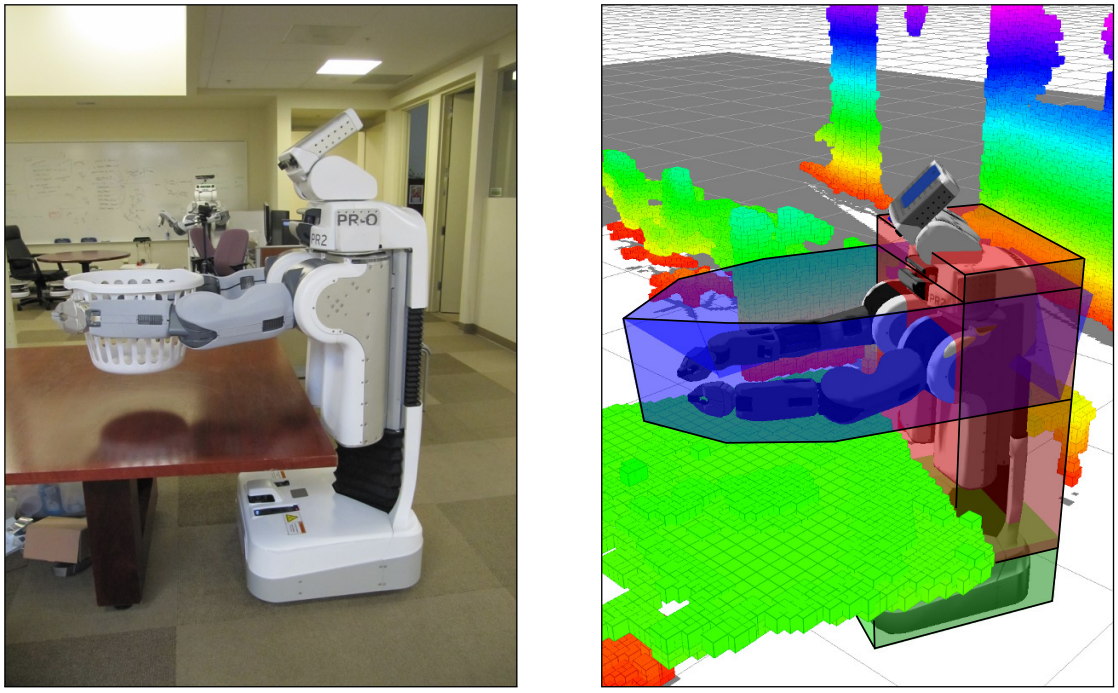


Figure 7.1.: A PR2 picks up a laundry basket from a table in order to carry it away. To allow for efficient collision checks while considering the 3D structure of the environment and the robot, we use a multi-layered representation for the robot and the environment, shown on the right. For the PR2, it consists of projected layers for the base (green), spine (red), and arms (blue) in addition to a full 3D collision map.

the robot is actually collision-free in the 3D world. This presents a typical scenario for a mobile manipulation robot, e.g., when picking up a large object from a table. Figure 7.1 illustrates this use case for the PR2 mobile manipulation robot. Here, the robot can only pick up the basket by moving its base underneath the table with its arms above. Such a configuration is unachievable using a traditional two-dimensional environment representation in navigation planning. Additionally, when carrying a large object such as the laundry basket, no path through a narrow passage might be found using a 2D map. This is due to the enlarged footprint in the 2D map needed to account for the basket and the extended arms.

A possible solution could be full 3D collision checking considering the complete robot body, as we have employed in the previous chapter. With a complex robot shape with multiple degrees-of-freedom (DOF) in typical indoor environments, however, this is expensive and will lead to long planning times when planning paths for navigation. A coarse environment representation could speed up the collision checks, but the lack of high-resolution information can prevent the robot from reaching goals in clutter.

To overcome these limitations, we propose an integrated navigation framework that utilizes a combination of collision checks in the full 3D representation as well as a multi-layered 2D representation of both the robot and the environment, illustrated in Fig-

ure 7.1 (right). For the scope of this chapter, we assume a fixed-base wheeled manipulation robot for simplicity, as opposed to the bipedal humanoid robot shape in the other chapters. This has the advantage that we can focus on the navigation system without the need to consider the walking gait and its stability while carrying large objects.

At the beginning, the robot builds an octree-based 3D occupancy map with the framework described in Chapter 2. This 3D map is then projected down onto a multi-layered 2D representation. Based on the current configuration of the robot and any object it might be holding, corresponding projected footprints for the different layers are automatically computed. The robot then uses the projected 2D maps and footprints for a first set of collision checks, and only uses the 3D map when required. The advantage of this method is that expensive 3D collision checks during planning can be avoided when no collision is found in any layer. We apply the anytime repairing A* search to generate a global plan on a lattice graph based on the layered representation. A local planner then computes the required commands to execute this path and validates the path during execution. The robot hereby constantly updates its environment representations from 3D sensor data.

Our novel contributions in this chapter are as follows. First, we present a system that is capable of efficient 3D planning and navigation. Our approach is particularly suitable for mobile manipulation tasks, such as carrying large objects in cluttered environments. Another important contribution is the integration of our efficient 3D representation for large-scale indoor environments with fast realtime motion planning, implemented and validated on a real robot operating with noisy sensor data in a cluttered environment. Our approach provides the first implementation of efficient navigation planning for mobile manipulation systems of different shapes in arbitrary configurations.

We validated our approach through real-world experiments on the wheeled PR2 mobile manipulation robot. Using our approach, the PR2 is able to carry large objects in a cluttered indoor environment. It is able to navigate collision-free and efficiently, transporting objects to parts of the workspace unreachable with traditional 2D navigation planning approaches.

7.1. The PR2 Robot Platform

We implemented our system on the PR2, which is a wheeled mobile manipulation system equipped with an omnidirectional base, an articulated sensor head and two 7-DOF arms. Throughout this work, we used the Hokuyo UTM-30LX 2D laser range finder in the base for localization and the stereo sensor in the head for 3D perception and mapping. In addition to the laser, the Monte Carlo localization (MCL) uses odometry information from the wheel encoders and an inertial measurement unit. The 2D map for localization contains walls and other static parts of the environment. Opposed to a walking humanoid, for which the 3D localization introduced in Chapter 3 yields a higher accuracy, localizing in 2D is sufficient here since the robot is constrained to the ground plane and the 2D laser range finder operates in the same space.

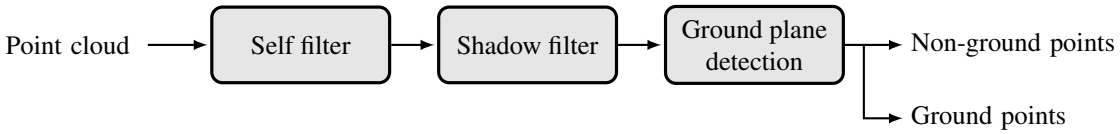


Figure 7.2.: Perception pipeline for navigation in cluttered environments.

The stereo cameras are augmented by a structured light projector to provide dense data (Konolige, 2010). Calibrated joint encoders provide accurate proprioceptive data about the robot's kinematic configuration. The stereo sensor serves as the main sensing input for detecting obstacles in the environment for our approach. Alternatively, an RGB-D camera such as the Microsoft Kinect could be used. The lack of an external shutter control on this consumer-level camera, however, requires an additional time synchronization for a robust self-filter and pose registration of the 3D point clouds while the robot moves.

The dense stereo pair of the PR2 has a narrow field of view of 55°. To make that practical for navigation, particularly to avoid obstacles, we implemented an active sensing behavior which always points the sensor head into the current driving direction. Because the PR2 features a quasi-omnidirectional drive, the driving direction can be to the side or even backwards.

The sensing pipeline for our approach builds on existing components developed for a tabletop manipulation application (Ciocarlie et al., 2010) and is shown in Figure 7.2. The point clouds from the stereo cameras are first run through a self filter. Based on the known robot mesh model and proprioception, the self filter removes points from the point cloud that are on the robot's body and on manipulated or carried objects. Accurate time synchronization between stereo and proprioceptive data on the PR2 ensures an effective self filtering. A so-called shadow filter then removes all errors inherent to stereo sensing: erroneous points that cannot be seen by both cameras of the stereo pair and veiling points on sharp edges and corners of the joints. Finally, a RANSAC-based ground-plane detection labels the remaining points as either ground or not ground for insertion into the octree-based 3D occupancy map.

7.2. Environment Representation

Mobile manipulation requires the ability to represent and process a fairly large environment efficiently and accurately. The representation must be compact and easy to incrementally update to account for dynamic obstacles and changing scenes.

7.2.1. Octree-Based 3D Occupancy Map

For collision checks between the robot and the environment, a full 3D representation is required in order to maintain full flexibility. Since 3D occupancy grids pose challenges on

computational memory requirements, we use the octree-based volumetric representation introduced in Chapter 2 to efficiently build and store a probabilistic 3D occupancy grid at a resolution of 2.5 cm.

For incrementally mapping the environment in 3D, the robot constantly updates the map with the most recent sensor information. Based on the pose estimate from MCL in the static 2D map, this corresponds to mapping with known poses. The point clouds output from the perception pipeline are self-filtered with points belonging to the ground plane marked, as described in the previous section. The sensor update of the 3D map, described in Section 2.1.3, uses ray casting from the sensor origin to the end points to update the map. All end points are usually considered as obstacles, while the area between the sensor and the end point is considered to be free. We extend this sensor model here to mark the ground plane as traversable and not as an obstacle. Opposed to simply ignoring the ground points, this enables us to use rays to them for clearing obstacles that have moved, such as people walking through the sensor field of view.

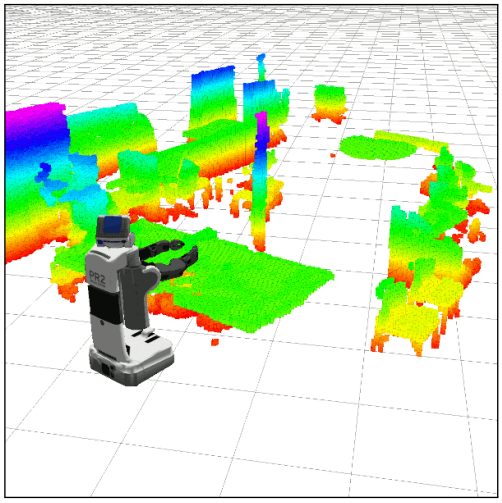
To ensure the proper handling of sensor noise and dynamic updates when obstacles such as manipulated objects or moving people are appearing and disappearing, we apply the probabilistic map update with clamping that was introduced in Chapter 2.

The robot initially builds a 3D map of the environment that serves as a base representation for planning. During task execution, it modifies this map with incremental updates from sensor data, re-planning around appearing obstacles whenever needed. Our framework further processes the 3D map to create a multi-layered 2D representation of the environment for improved efficiency in motion planning. We will now describe this process in detail.

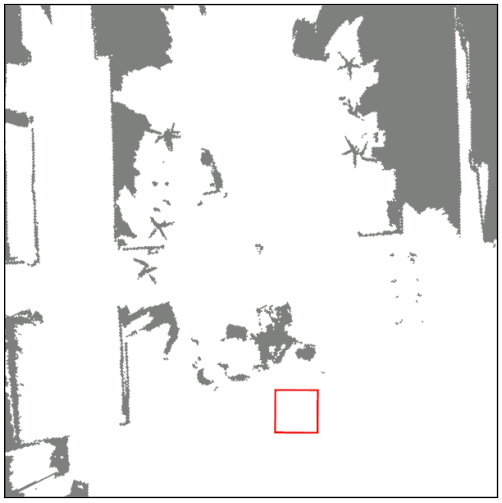
7.2.2. Multi-Layered 2D Obstacle Maps

The geometric structure of a mobile manipulation robot such as the PR2 can be exploited for efficient collision checking by a spatial subdivision into multiple 3D volumes. Such a subdivision is especially applicable when the robot navigates while carrying an object since its geometric structure will stay mostly constant during that task. As illustrated in Figure 7.1, the PR2 can be subdivided into three parts: the box-like base, the spine from base through the head, and the arms with attached objects to be carried or manipulated. While the base and spine layers remain fixed for this robot, the arm layer can change its height and footprint as the arms move, e.g., to adjust to new objects to be carried. Other manipulation robots can be subdivided in a similar manner. Whenever a new plan is requested, we first compute a 2D footprint for each layer from the convex hull of the down-projected robot body part meshes associated to this layer. Figure 7.3 shows an example configuration of the PR2 and the three computed footprints as polygons.

In the same way, we represent the environment in multiple corresponding 2D layers. Each layer contains projected representations of the obstacles that can collide with the body parts associated with that layer. For the PR2 this results in three environment layers, also shown in Figure 7.3. It is clear that in typical household scenarios the layer of the

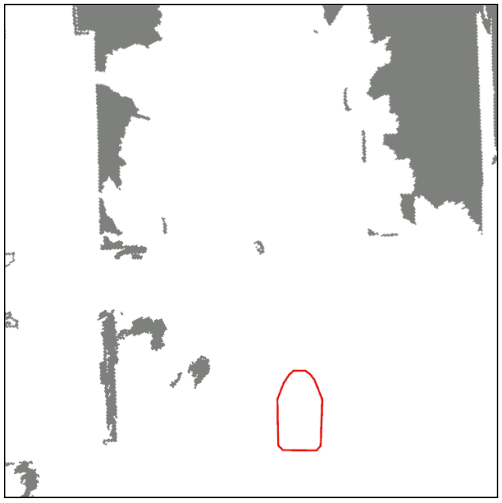


(a) Full 3D occupancy map



(b) Base layer

(c) Spine layer



(d) Arms layer

Figure 7.3.: Full 3D occupancy grid and projected 2D grid maps with footprint (red polygon) for each layer of the PR2 robot.

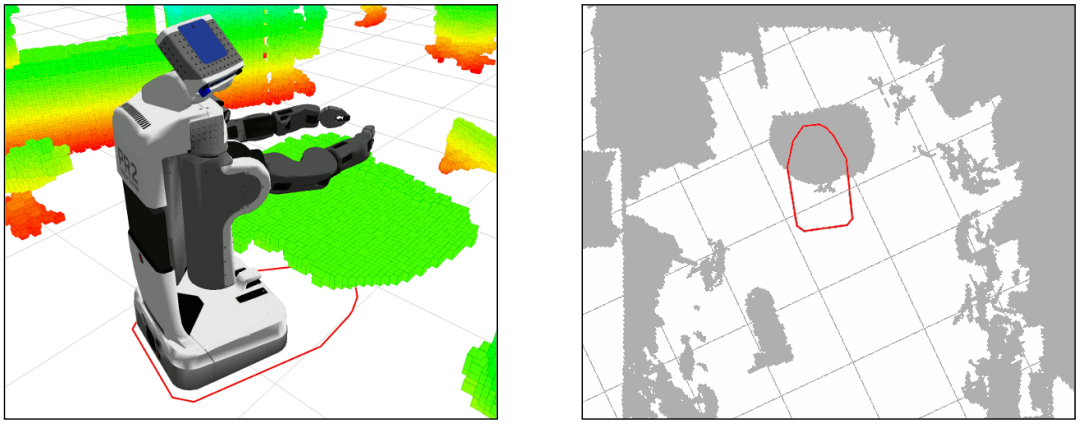


Figure 7.4.: A PR2 approaches a table collision-free with its arms over the table and its base underneath. The robot's projected footprint is in a 2D collision with the projected environment, as indicated on the right

Algorithm 7: Update of multiple projected 2D map layers based on 3D map update

Input: Set of projected map layers $\mathcal{L}$, 3D occupancy map m with updated area marked

Output: Map layers are updated in area observed by sensor

```
1 foreach (x,y) in updated area of 3D map m do
2     foreach l ∈ L do
3         l(x,y) ← FREE
4         foreach z in height range of layer l do
5             if m(x,y,z) is OCCUPIED or UNKNOWN then
6                 l(x,y) ← OCCUPIED
7                 break                    // Exit loop over z for current layer
```

arms only has few obstacles compared to the spine since the arms are at a greater height than most of the obstacles, such as chairs and tables. Using multiple 2D layers in this manner allows motion planners to avoid expensive 3D collision checks. As an example, consider Figure 7.4 in which the robot has its arms over a table and the base under a table, e.g., to pick up an object. When using a single 2D map with a 2D projected robot footprint, this footprint collides with the down-projected environment. A full 3D collision check would be needed to confirm whether a collision is actually occurring. With our approach, however, the table is only projected onto the spine layer, in which it does not collide with the robot. Since none of the footprints are in collision, an expensive full 3D check is not needed in our multi-layered approach.

Starting from an empty map or an initial map, each incremental 3D map update also updates the down-projected 2D maps according to Algorithm 7. The algorithm updates

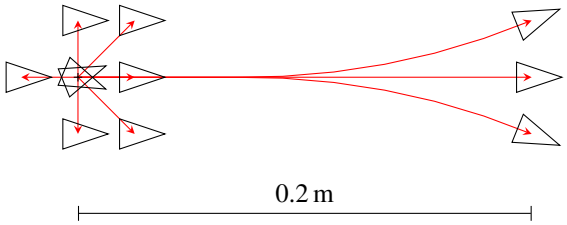


Figure 7.5.: Omni-directional motion primitives for the PR2.

the 2D projections for a given map cell (x,y) in layer l by traversing all discrete levels z in a range corresponding to the minimum and maximum height of l. For instance, the layer used for the robot's arms has a height range determined by the highest and lowest points on the arms and potentially attached objects. If for all z in l, (x,y,z) is free in the 3D map, (x,y) is marked as free in the layer. As soon as one voxel (x,y,z) is occupied in the column of cells, $l(x,y)$ is considered to be occupied. The size of the 2D map cells is hereby determined by the resolution of the 3D occupancy grid. Since our 3D environment representation can distinguish unknown areas from free or occupied ones, the robot can avoid them for navigation by also marking them as occupied obstacles in line 5. This results in a safe and conservative navigation behavior. Occupied and unknown cells in a 2D map layer hence represent possible obstacles in 3D. A free cell, on the other hand, guarantees that there is no obstacle in the 3D environment corresponding to that layer.

Some map cells in a 2D layer have the property that for each (x,y)-cell and each z in the height range of that layer, (x,y,z) is occupied in the 3D map. This is the case for all solid obstacles that span a vertical range, such as walls. In an extension of Algorithm 7, we mark these cells in the layers as *solid obstacle*. The collision check, detailed in the next section, can make use of this classification and save planning time. A collision with a solid obstacle cell in 2D guarantees that there is a collision in 3D, whereas regular obstacle cells in the 2D projection only indicate a potential collision and require an additional 3D collision check.

7.3. Planning and Navigation Framework

The 3D map and the multi-layered 2D obstacle maps serve as input to our planning and navigation framework. We apply a *global planner* using search-based planning on lattice graphs to construct a plan to the goal in the position and planar orientation space (x,y,ψ). This plan is then executed by a *local planner*, which creates the motion commands from the global plan and validates them during execution. To localize the robot during navigation, we rely on 2D Monte Carlo localization in a static 2D map.

7.3.1. Search-Based Planning on Lattice Graphs

The global planner operates on a lattice graph corresponding to the 2D space with orientations (Likhachev and Ferguson, 2009; Pivtoraiko and Kelly, 2005). Similar to our footstep planning framework introduced in Chapter 5, states are connected by a set of motion primitives. Motion primitives are short, kinematically feasible motion sequences. The path found by the planner is then a concatenation of these motions. The advantage of this state space representation is that the resulting plans contain smooth paths that can handle non-holonomic constraints and non-circular robot footprints. Figure 7.5 shows the set of motion primitives we employ for the PR2. In addition to driving forward and turning on the spot, there are also primitives for sideways and backwards motions.

The robot footprint is not restricted to be circular and can assume any shape, depending on the current manipulator configuration and any object the robot might be holding. For efficient 2D collision checking with an arbitrary robot footprint, we compute a sequence of footprint collision cells for each motion primitive and each possible state orientation of the planner by rolling out the projected robot footprint along the path of the motion primitive. In our multi-layer representation, this needs to be done for every projected layer of the robot. This step took approximately three seconds in our trials and is only necessary if the robot configuration changes. For known configurations, it can be precomputed. For example, the base and spine layers of the PR2 are fixed with respect to the robot's base coordinate frame, and thus their footprints never change.

On the lattice graph representation, we then employ the Anytime Repairing A* (ARA*) search. As introduced in Chapter 5 for footstep planning, ARA* runs a series of weighted A* searches that inflate the heuristic component by a factor $w \geq 1$ while efficiently reusing previous information (Likhachev et al., 2004). The search uses a large w at the beginning, causing it to find a non-optimal initial solution quickly. Then, as time allows, the search reruns with incrementally lower values for w while reusing much of the information from previous iterations. Given enough time, ARA* finally searches with $w = 1$, producing an optimal path. If the planner runs out of time before, the cost of the best solution found is guaranteed to be no worse than w times the optimal solution cost. This allows ARA* to find some solutions faster than regular A*, and approach optimality as time allows.

During the ARA* search, the costs for applying a motion primitive correspond to the length of the trajectory and additionally depend on the proximity to obstacles to prefer solutions further away from obstacles. The heuristic for the planner uses a 2D grid search from the goal state. This heuristic only searches over the 2D grid map of the base layer with obstacles inflated by the base footprint incircle. In the base layer, every 2D collision implies a real collision in 3D. Thus, the heuristic stays admissible and consistent.

7.3.2. Efficient Collision Checking

In the planning process, ARA* checks the states generated at every step for collisions. Algorithm 8 lists our approach for efficient collision checking based on the multi-layer

Algorithm 8: Efficient multi-layer 2D and 3D collision check

Input: Set of projected map layers $\mathcal{L} = \{l_1, \ldots, l_n\}$, corresponding set of projected footprints $\{f_1, \ldots, f_n\}$, 3D occupancy map m, robot configuration $\mathbf{x}$

Output: TRUE if $\mathbf{x}$ collides with an obstacle, FALSE if it is collision-free

```
1 foreach l ∈ L do
2     f ← robot footprint corresponding to l
      // 2D collision check of footprint in corresponding map layer
3     if checkCollisions2D(x, f, l)= TRUE then
4         if l is solid or 2D collision is caused by solid obstacle then
5             return TRUE
6         else
              // 3D collision between the full robot mesh model and the occupancy map
7             return checkCollisions3D(x, m)
8 return FALSE                    // No 2D layer indicated a collision, x is collision-free
```

2D representation. It relies on fast collision checks in the 2D map in line 3 to rule out expensive 3D collision checks between the full kinematic configuration of the robot and the 3D map.

As a complement to solid obstacle cells that are marked during the map-building phase, we here use the fact that some layers correspond to solid parts of the robot. These are mostly box-like parts that solidly fill their projection bounding box, such as the base and spine of the PR2. For these layers, any collision in the 2D projection implies a definite collision in 3D, regardless of the z-coordinate. A full 3D collision check is thus no longer needed. We refer to these layers as *solid* layers.

The algorithm performs a 3D collision check in line 7 only when a possible collision is indicated in 2D, and the collision was not confirmed to be definite by a solid 2D layer or obstacle. In practice, our ARA* implementation does not check a single configuration $\mathbf{x}$, but the fully traversed footprint of the motion primitive.

Note that we use our multi-layered 2D obstacle map only for ruling out full 3D collision checks, not to replace them completely. In order to preserve the full flexibility of an arbitrary robot configuration, it is still necessary to test the kinematic configuration for 3D collisions in some cases. We perform full 3D collision checks in the Open Dynamics Engine (ODE, 2011) using a collision mesh model of the robot.

7.3.3. Plan Execution

During execution, the concatenated discrete motion primitives from the global planner have to be converted into motor commands for the robot's base. Additionally, the validity of the plan has to be checked while it is being executed because obstacle positions might

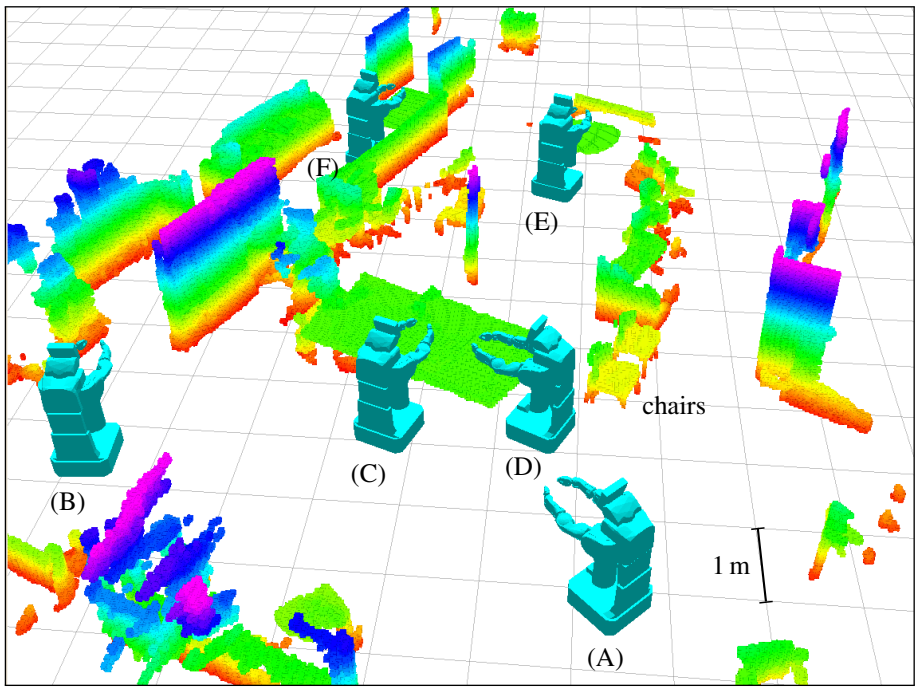


Figure 7.6.: The planning environment with six robot configurations (A)–(F) chosen as start and goal poses. A table and two chairs form a narrow passage. We removed the chairs to create a second, easier scenario.

have changed since planning, or the robot slightly deviated from the original plan due to motion noise. The local planner computes the omnidirectional velocities required to reach the next pose (x, y, ψ) along the path and performs a single trajectory rollout to check collisions in the updated obstacle map. As in the global planner, collision checks are first performed in 2D, and only in 3D when required.

In case collisions are predicted in the trajectory rollout, e.g., because the environment has changed too much or the robot has significantly deviated from its path, the robot stops and the global planner is invoked again to find a new path around the obstacle.

7.4. Evaluation

We carried out extensive experiments to evaluate our approach and demonstrate its usability for realtime mobile manipulation tasks.

7.4.1. Motion Planning Performance

The first set of experiments was designed to show the superior performance of our approach compared to conventional techniques. In our evaluation, we compared the following three approaches for collision checking during navigation planning:

1. *Single-layer 2D:* A traditional navigation approach using only one projected footprint of the robot in a projected two-dimensional environment representation.

2. *Single-layer 3D:* A navigation approach which extends *single-layer 2D* with full 3D collision checks. The full 3D collision checks are always performed when the single 2D footprint is in collision with the projected two-dimensional map, i.e., it indicates a potential collision. Compared to single-layer 2D, this approach can solve more problems but it is less efficient due to the added cost of 3D collision checks.

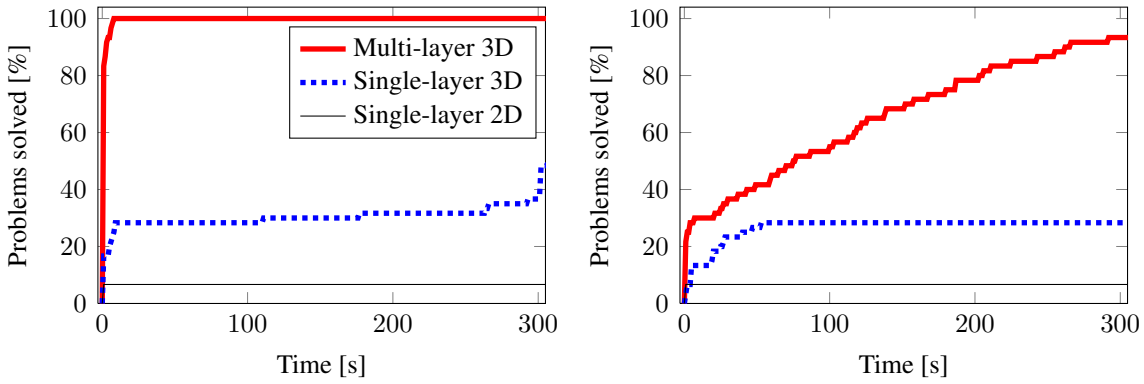
3. *Multi-layer 3D:* Our approach of combining a multi-layered 2D occupancy grid and with full 3D collision checks only when absolutely necessary, i.e., a projected 2D layer indicates a potential collision.

While all planning approaches use the same heuristic which estimates the costs along a 2D path, the heuristic is based on different environment representations. The single layer approaches use the projected map with the projected robot footprint incircle as obstacle inflation. Our multi-layer 3D approach uses the projection of the base layer with the base footprint incircle instead, as described in Section 7.3.1.

We generated multiple planning problems in which the robot was posed in a configuration with extended arms for manipulation. This configuration requires a large projected footprint for traditional motion planing. The planning environment is a real cluttered office space and was generated from 3D sensor data of the PR2. The first planning scenario with six different robot configurations as start and goal is shown in Figure 7.6. It includes a narrow passageway between two chairs and a table, which the robot could only negotiate by moving sideways with its base under the table. For a second scenario, we removed the chairs to create an easier planning problem. By planning between each pair of the six robot configurations in both scenarios, we obtained a total of 60 different planning problems.

On each of the 60 problems, we then planned a path using ARA* and the three approaches detailed above. The planner used an initial suboptimality of $w = 10$ and was allowed to continue until either $w = 1$ was reached or a time limit of five minutes had elapsed. The success rate of each planning approach is shown in Figure 7.7 for generating first solutions and for generating the optimal solution. Our multi-layer 3D approach always succeeded in generating a motion plan, even in the most cluttered cases (e.g., planning between (B) and (E)). It outperformed the single-layer 3D approach by finding far more solutions both in a short realtime window and in the long run. With our approach, a first solution was always found within at most 7.3 seconds and the average planning time for the first solution was 0.7 seconds. Note that only the two configurations (A) and (B) have projected 2D footprints that are not in collision, enabling the traditional single-layer 2D approach to succeed in only four cases.

Table 7.1 compares our approach with the single-layer 3D approach in the 32 planning attempts where both succeeded. Data from single-layer 2D is omitted since the approach



(a) First solution ($w = 10$) **(b)** Optimal solution ($w = 1$)

Figure 7.7.: Success rate as percentage of solved problems after a certain time for 60 planning problems in cluttered space.

First solution	Single-layer 3D	Multi-layer 3D (our approach)
Time [s]	130.81 ± 143.67	0.03 ± 0.06
# expansions	13 911 ± 16 572	1 401 ± 1 618
2D collision checks (primitives)	153 375 ± 182 763	15 408 ± 17 799
3D collision checks (primitives)	22 017 ± 24 669	0.00 ± 0.00

Table 7.1.: Mean and standard deviation for first solutions in the 32 planning scenarios where both 3D approaches succeeded within 5 minutes.

First solution	Multi-layer 3D (our approach)
Time [s]	1.52 ± 2.13
# expansions	69 915 ± 107 907
2D collision checks (primitives)	769 069 ± 1 186 974
3D collision checks (primitives)	2.89 ± 9.12

Table 7.2.: Mean and standard deviation for first solutions in the 28 planning attempts where only our multi-layer 3D approach succeeded within 5 minutes.

was only able to plan between two configurations. The results demonstrate that our multi-layered representation greatly improves the performance of the motion planner, allowing it to quickly find an initial solution. In these easy scenarios no 3D collision checks were required at all and only few expansions were necessary for the first solution. Multiple 2D layers allow the heuristic to be more informed while keeping the search away from obstacles and 3D collision checks due to the obstacle inflation, which penalizes states close to obstacles. Additionally, there were only few obstacles in the arm layer of the robot that were not classified as solid obstacles. The refinement of the solution to an optimal solution ($w = 1$) takes more time, but in practice the sub-optimal solutions already lead to an efficient navigation behavior of the robot. In these easier scenarios, our approach reached an average of $w = 2.4$ after five seconds.

Table 7.2 shows data aggregated from the harder 28 planning attempts for which only our multi-layer 3D approach succeeded within five minutes. After five seconds, our approach solved 24 of them with an average of $w = 6.3$.

In Table 7.1 and 7.2, the number of 2D and 3D collision checks refers to the number of complete motion primitives that were checked. A motion primitive may have several intermediate positions to check in addition to the start and end state. Each of our primitives may require up to ten collision checks for individual robot positions, whereas the actual number is usually less since we abort as soon as one intermediate position is in collision.

7.4.2. Docking Maneuvers With a Real Robot

A second set of experiments involved repeated docking and undocking with a table using the real PR2 robot in a similar cluttered office environment. A manipulation robot will have to perform this action when picking up objects from a table. Note that configurations that require the robot to reach over a table are unachievable using traditional 2D navigation approaches. We therefore evaluated this set of planning and execution experiments only for our multi-layered approach. A typical docking maneuver to pick up a large object from a table can be seen in Figure 7.8. In our experiments, all goal configurations were reached collision-free while the robot docked and undocked a table 12 times.

7.4.3. Navigation While Carrying a Large Object

Finally, we evaluated our approach with a set of navigation experiments in the cluttered environment shown in Figure 7.6. In this scenario, the robot was carrying a laundry basket with both arms and had to pass through narrow areas, forcing it to move sideways close to obstacles. The location of the laundry basket was provided a priori to the robot. The robot first planned a docking maneuver to a goal location in front of the basket. It then lifted up the basket with a pre-defined motion of the arms. A bounding cylinder was subsequently used for collision checks with the basket and to filter out the basket from sensor data during navigation.

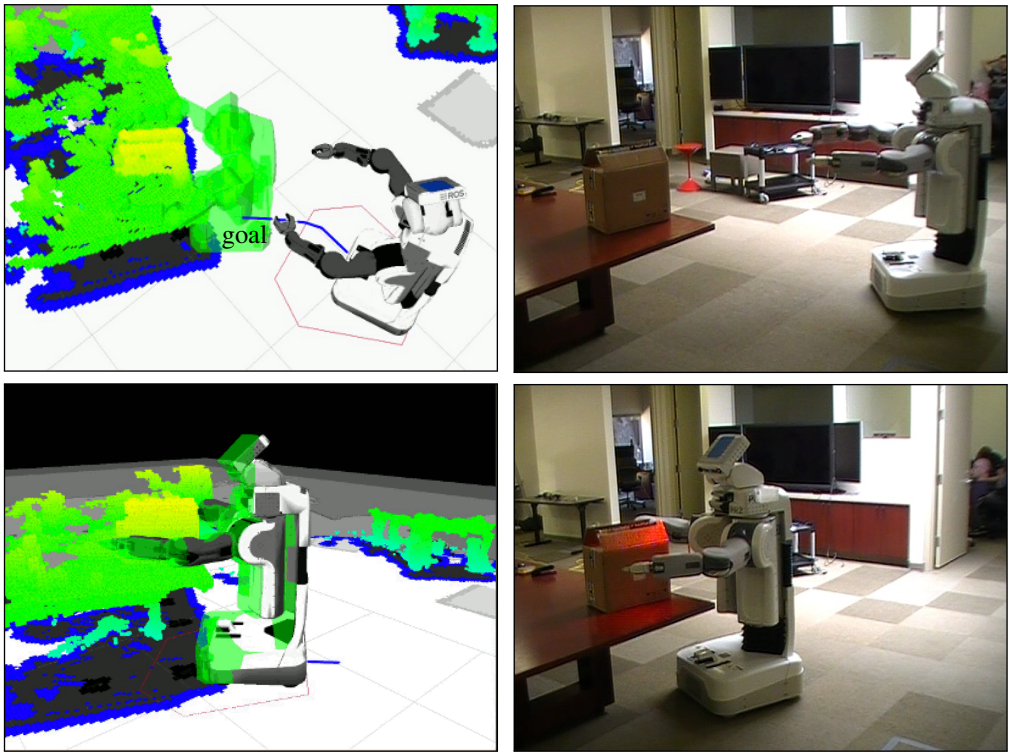


Figure 7.8.: Docking at a table to pick up a large object. *Left*: Goal configuration (shaded green) and resulting plan (blue). *Right*: Execution of the plan by the PR2.

The final navigation goal for the robot was then at another table where it would have to put down the basket. We conducted multiple trials using several combinations from a set of six goal and start states. The robot successfully executed each task.

Figure 7.9 shows a series of snapshots while executing this task, whereas the complete planned path is shown in Figure 7.10. The precomputation time for the footprints was 3.46 s in this scenario and the planning time to the first solution with $w = 10$ was 3.16 s. To pick up the laundry basket, the robot initially had to move its base under the table. Afterwards, the narrow passage with chairs forced it to move sideways under the table with the basket over the table. Finally, the PR2 approached the second table to put the basket down. Note that large parts of the workspace in this scenario are unreachable by the robot using traditional 2D navigation planning approaches. A video of the complete sequence is available at http://www.youtube.com/watch?v=XzkC4Ez8GYE.

7.5. Related Work

Navigation in cluttered environments is a well-studied problem for mobile robots. Most approaches to this problem, however, have focused on navigation for a 2D projected footprint of the robot moving in a projected 2D representation of the world (Burgard et al., 2000; Kümmerle et al., 2009, 2013; Kuwata et al., 2008; Marder-Eppstein et al., 2010; Thrun et al., 1999). Therefore, they are unsuited for our motivating problem of mobile manipulation in a cluttered environment since they will reject any configuration of the robot where the 2D projected footprint of the robot is in collision with the 2D environment representation.

Marder-Eppstein et al. (2010) demonstrated a long-running navigation experiment with the PR2. Their approach relies on 3D sensor data only for the local surroundings of the robot and uses a 2D map with a projected robot footprint for planning. This representation allowed for quick updates, e.g., to avoid dynamic obstacles, but restricted the system to operate only in regions that were collision-free in the 2D projections. A similar approach was proposed by Kümmerle et al. (2013), which enabled an outdoor robot to autonomously navigate amongst pedestrians through a crowded urban environment. Due to the robot's circular footprint, the navigation planner operated in 2D with obstacle detection in the direct vicinity of the robot in 3D.

Scholz et al. (2011) presented cart pushing with a mobile manipulation system. The ARA* planner used for navigation contained motion primitives to drive the robot, but also to articulate the pushed cart which is necessary to drive around corners. Navigation was again limited to goals where the 2D projection of the cart and the robot was not in collision. This disallowed, for example, actions that could have moved and stored the cart under a table.

In the assistive mobile manipulation system presented by Chen et al. (2013), the PR2 assists a user in everyday tasks with varying levels of autonomy. The robot can autonomously navigate to a 2D navigation goal based on the approach by Marder-Eppstein

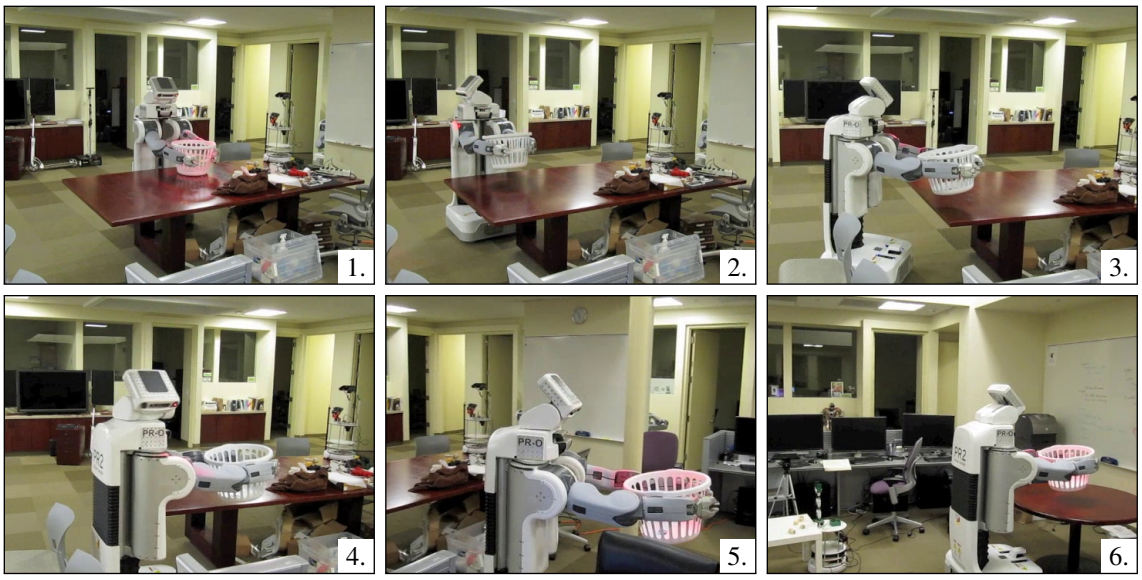


Figure 7.9.: Sequence of snapshots showing the PR2 navigating autonomously with a laundry basket in a cluttered environment. A video of this sequence is available at http://www.youtube.com/watch?v=XzkC4Ez8GYE.

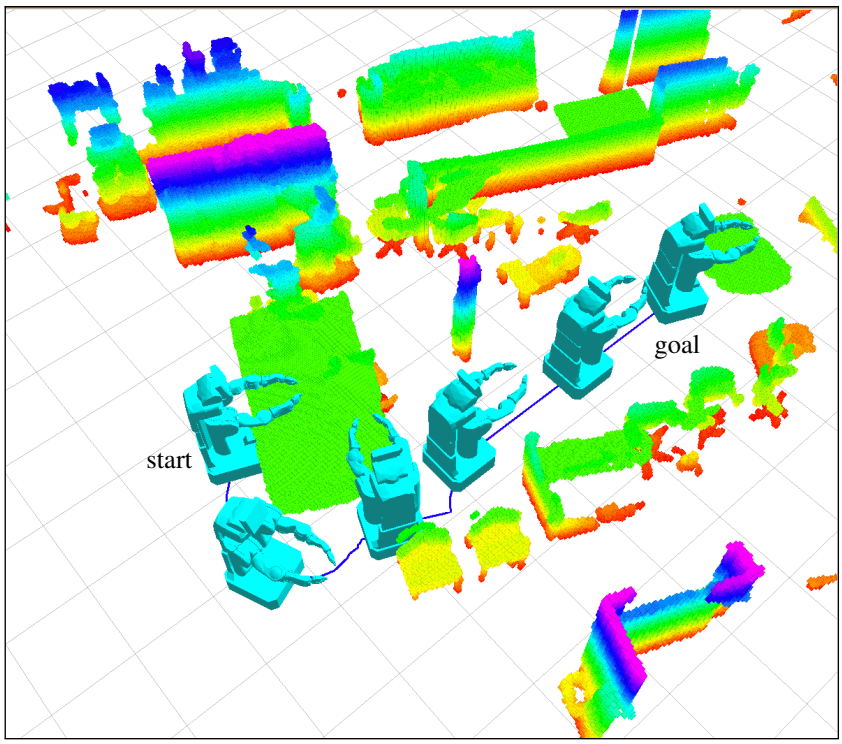


Figure 7.10.: Visualization of the 3D environment and path taken by the PR2 for moving a laundry basket through a cluttered indoor environment in Figure 7.9. Note that the basket itself is not visualized.

et al. (2010). However, goals close to obstacles, e.g., for manipulating objects on a table with the robot's base underneath, have to be manually specified by the user and are approached open-loop without collision avoidance. By integrating our navigation approach, these problems could be solved autonomously, taking more workload off the user.

The home assistant system proposed by Srinivasa et al. (2008) consists of a stationary manipulator that picks up cups from a mobile delivery robot. The robot is treated as a circular 2D shape during navigation. Compared to our approach, this restricts the usage to delivering only small objects that do not protrude from the robot's base footprint.

Lazy collision checks and an enlarged robot model were used for fast RRT-based motion planning in a simulated kitchen environment by Vahrenkamp et al. (2007). The authors concentrated on manipulation tasks in a small part of the environment. They later extended the approach to coarse motion planning for the robot base while further away from the goal with an increased planner granularity closer to the goal (Vahrenkamp et al., 2008). However, their approach does not cover long navigation plans through a cluttered environment. Also, randomized planning approaches have no guarantees on the solution quality and rely heavily on smoothing to shorten the final path.

Perrin et al. (2012b) recently proposed a hybrid bounding box approach for randomized footstep planning. In their approach, the authors approximate the body of a humanoid by two bounding boxes for the feet and a larger one for the upper body. In this way, the authors can plan discrete motions for footsteps and a continuous sequence for the upper body. Similar to our approach, the plan for the upper body layer hereby ignores all small obstacles that only pose obstacles for the feet. The difference to our work is that the authors employ a randomized planner instead of a search-based one, and only check the bounding boxes for collisions, which avoids configurations in which the bounding box collides but the actual robot shape does not.

For efficient collision checking in 3D, several hierarchical approaches have been proposed in the past such as oriented bounding box trees (Gottschalk et al., 1996) or sphere tree hierarchies (Quinlan, 1994; Steinbach et al., 2006). They all rely on a spatial subdivision of the 3D space. Recently, Pan et al. (2012) presented a general framework for collision checking called FCL. Our approach of avoiding 3D collision checks with a set of 2D layers can be seen as orthogonal to these methods, since 3D collision checks are still substantially more expensive. Lau et al. (2011, 2013) introduced techniques for incremental updates of collision maps in a non-circular robot's configuration space. After convolving the map with the discretized shape of the robot in a pre-computation step, collision checks correspond to a single lookup. While their approach can efficiently update the map for dynamic environments, any change in the robot's kinematic configuration, e.g., after picking up an object, requires a costly full re-computation of the configuration space obstacle map.

7.6. Conclusion

7.6.1. Summary

We presented a novel approach for efficient navigation with a mobile manipulation robot in three-dimensional, cluttered environments such as offices or homes. Our integrated approach combines an efficient octree-based 3D representation and an anytime search-based motion planner, allowing the generation of almost real-time bounded sub-optimal plans. For efficient collision checking during planning, our approach utilizes a multi-layered 2D environment representation. This novel representation contains multiple 2D projections of the sensed 3D environment and projected footprints of the robot's body parts. By checking these projected footprints in their corresponding 2D map layer for collisions, our approach can avoid expensive 3D collision checks until they are necessary.

We demonstrated the performance of our framework on the mobile manipulation robot PR2 in exhaustive real-world experiments. The robot was able to navigate efficiently and collision-free, carrying a laundry basket with both arms from one table through a heavily cluttered room to another table. The robot also successfully performed docking and undocking maneuvers with extended arms at a table, thus maximizing its workspace for manipulation by moving the base under the table. Our combination of a multi-layered 2D representation with a full volumetric 3D one outperforms collision checking in full 3D while maintaining flexibility with respect to the robot configuration, navigation task, or environment at hand. Most of the experimental scenarios were not solvable with traditional 2D navigation planning approaches. Our approach found an initial solution on average within 0.7 seconds, while the highest planning time for an initial solution was 7.3 seconds. This underlines the efficiency and practicability of our approach for navigation in mobile manipulation scenarios.

7.6.2. Future Work

A natural extension for future work will be to move larger objects such as chairs or carts with articulated primitives similar to Scholz et al. (2011) in a cluttered environment, e.g., to store them away under a table. It also remains to extend this work towards biped robots for mobile manipulation tasks. The lower body of these robots, however, needs a special treatment since it constantly changes its configuration due to the walking motion. This could be done along the lines of the hybrid bounding box approach of Perrin et al. (2012b) with an enlarged bounding box for the legs that encompasses the complete range of the walking motion.

7.6.3. Impact

A first version of the framework presented in this chapter was released open-source as a ROS stack at http://wiki.ros.org/3d_navigation in 2011. It was later extended and became

part of the basic functionality provided to the participating teams in the ICRA 2012 Mobile Manipulation Challenge[1], enabling them to perform challenging tasks such as setting or clearing a table with the PR2.

[1] http://mobilemanipulationchallenge.org/?page_id=131 (retrieved November 5, 2013)

Chapter 8

Conclusion

8.1. Summary

Within this thesis, we presented several novel contributions to the field of humanoid robotics. We were particularly interested to solve the question on how humanoid robots can autonomously navigate in complex indoor environments. These environments can contain all kinds of clutter or even multiple levels connected by staircases. In our contributions, we coped with challenges in the areas of 3D environment representations, localization, perception, motion planning, and mobile manipulation.

Since an environment model is an integral part of any robot navigation system and two-dimensional representations are not expressive and general enough for humanoid robots, we first introduced an octree-based 3D environment representation. The presented framework allows to represent volumetric 3D information in a probabilistic manner, can distinguish between free and previously unseen areas, is memory efficient, and allows multi-resolution queries. The evaluation with various real-world datasets demonstrates that our approach is able to model the environment in an accurate way and, at the same time, reduces memory usage. This allows a robot to compactly store information about its environment and later use it for localization, path planning, or collision avoidance.

Within this 3D representation, we then presented a probabilistic localization approach to estimate the full 6D pose of a walking humanoid based on noisy range and proprioceptive data. It is critical that a robot has an accurate estimate of its pose in the environment in order to perform any high-level task. We compared different sensor models using either a small 2D laser or a consumer-level depth camera as onboard range sensors. To reduce the number of required particles for Monte Carlo localization, we presented a method for the calibration of the motion model. In extensive experiments with different Nao humanoids, our approach resulted in accurate pose estimates both globally and for tracking while walking.

Next, we considered the task of climbing stairs, which present a common feature in man-made environments. We extended and compared two approaches for plane segmentation and described how they can be used to detect stairs in 3D range data. We employed kinesthetic teaching to learn the whole-body motion for climbing stairs from

human demonstrations. To improve the localization accuracy for the critical task of climbing stairs, we extended our localization approach to additionally use vision data from the robot's onboard camera in an improved proposals particle filter. The presented methods significantly improved the localization accuracy, thus enabling a Nao humanoid to reliably climb up a complicated staircase consisting of ten steps and a winding part.

Afterwards, we discussed search-based footstep planning for biped navigation. We adapted several anytime planning approaches for footstep planning and evaluated them in different scenarios. The ARA* and R* planners return fast initial solutions and improve them as time allows. We hereby found ARA* to be more dependent on a well-designed heuristic, whereas R* is better able to avoid local minima. For incremental replanning, we presented the AD* algorithm, which allows an efficient plan reuse. To efficiently plan longer paths, we introduced adaptive level-of-detail planning, which combines fast 2D planning in open areas with detailed footstep planning close to obstacles.

We furthermore presented an approach for whole-body motion planning that is particularly suited for the manipulation of articulated objects such as doors and drawers. Based on the RRT-Connect planner and inverse kinematics, our approach plans for all degrees of freedom of a humanoid and considers the constraints induced by joint limits, maintaining balance, avoiding collisions, and following the trajectory of an articulated object. In experiments with a Nao humanoid, our planner was able to solve a variety of motion planning tasks that involve reaching into cabinets, opening doors and drawers, as well as picking up small objects.

Finally, we considered navigation in three-dimensional cluttered environments with a mobile manipulation setting. In our approach, we combine our octree-based 3D representation with an anytime search-based motion planner. For efficient collision checking, we introduced a multi-layered projected 2D representation of both the environment and the robot that is incrementally updated from the 3D map. We demonstrated the performance of our framework in real-world experiments with the mobile manipulation robot PR2. The robot was able to navigate efficiently and collision-free while carrying a large object with both arms through a heavily cluttered room.

In summary, we developed solutions to the following questions:

- How can a robot efficiently build and access a three-dimensional environment representation?
- How can a humanoid robot localize itself accurately within such an environment representation despite noisy sensor data?
- How can we enable a biped robot to reliably overcome challenging terrain such as stairs without falling?
- How can a biped robot efficiently reach a navigation goal, thereby stepping around or over obstacles?

- How can we plan challenging manipulation tasks that involve the whole body of a humanoid robot?

- How can a mobile robot reach a navigation goal efficiently and collision-free while carrying objects through cluttered environments?

We thoroughly evaluated all of the presented techniques. For most of our evaluations, we relied on real data of the affordable, small-scale Nao humanoid and thus demonstrated the applicability to a real humanoid platform. We believe that our work provides relevant contributions for advancing the navigation capabilities of humanoid robots in general. Our open-source implementations hereby provide building blocks that can be readily integrated into robotic systems or can be used as state of the art to compare future research against. Indeed, the rapid adoption of some of our presented methods within the research community illustrates the general usefulness and practicability of our work.

8.2. Outlook

While we presented important contributions and encouraging results for autonomous navigation with humanoid robots, there are several extensions possible for future research.

Our localization system currently relies on the availability of a 3D map to localize in. This map must be built and supplied by the user beforehand. A future extension could replace this step with an approach for Simultaneous Localization and Mapping (SLAM), which builds a 3D map as described in Chapter 3 while localizing the robot. The SLAM approach could use our sensor and motion models for humanoid robots in this context. Since SLAM in 3D is a hard problem that requires much computing power, a humanoid could initially explore the environment with the purpose of building the 3D map. It would then localize in the map and only update it when needed. Note that, for localization, the robot does not need a perfect 3D map with all obstacles in it. For obstacle avoidance and path planning, the robot could maintain a local obstacle map that it always keeps up-to-date based on the pose estimate from localization, as described in Chapter 7 for the PR2 and by Maier et al. (2012) for a humanoid.

Our footstep planning framework currently considers a planar surface that is traversable by the robot. Obstacles are also considered to be planar, such that the robot can step over them. By adding new stepping capabilities and by accounting for the actual robot shape in the collision check, the planning framework can be extended to three-dimensional environments. This enables a humanoid to step over clutter and to climb onto obstacles, as recently demonstrated by Maier et al. (2013). Combining this approach with our planning and navigation framework in Chapter 7 would enable humanoid robots to perform mobile manipulation tasks such as picking up and carrying a tray or other large objects. The collision checks for the humanoid's lower body could be sped up with a hybrid bounding box approach similar to Perrin et al. (2012b).

Whole-body manipulation can benefit from integrating perception, both for building an environment representation for collision checking and for detecting the handle of an object to grasp. Visual information can be used to correct inaccurate motions before and during the execution of the motion plan. Without prior knowledge of the articulated object parameterization, i.e., its DOF and hinge location, the robot could grasp the handle and carefully learn the trajectory through manipulation as proposed by Sturm et al. (2011). The robot could extend its manipulation range even more by considering stance leg changes and new footstep placements in the whole-body planner as opposed to remaining in single or double support mode.

Additional capabilities that build upon our contributions for navigation could make humanoid service robots more useful in a general context. Perception and a general framework for object detection would help detecting and localizing objects. In addition, semantic information about different objects is required for a robot to understand queries such as *"bring me the laundry"* or *"clear the table"*. A humanoid could obtain general information about the objects in the world and their semantics from a Web-based knowledge database such as *RoboEarth* (Waibel et al., 2011). Finally, for a truly autonomous behavior, a robot may be required to open a door or to move away obstacles blocking the path to the goal. The robot thus needs to reason about its actions and their effect on the world. To this end, our contributed navigation components can be integrated in a framework for high-level planning and reasoning. This could be a symbolic task planner (Dornhege and Hertle, 2013; Dornhege et al., 2009) or a knowledge processing and reasoning system such as *KnowRob* (Tenorth and Beetz, 2013).

Appendix A

The NAO Humanoid Robot Platform

In this part, we describe the humanoid robot platform used for most of our experiments in different configurations. The Nao robot by Aldebaran Robotics is 58 cm tall, weighs 4.8 kg and has 25 degrees of freedom: two in the neck, five in each arm, one in each hand, and five in each leg. In addition, the legs share a coupled hip joint (Gouaillier et al., 2009). The kinematic model is illustrated in Figure A.1. We used two similar hardware revisions throughout this work, V3+ ("Academic Edition") and V4. Both are shown in Figure A.2 with additional range sensors. The major difference between the hardware revisions concern their computing facilities and the kinematics of the arms. While the V3+ Nao is equipped with a 500 MHz AMD Geode CPU with 256 MB RAM, V4 features a 1.6 GHz Intel Atom processor with 1 GB RAM. In this work, we performed most computations offboard on a standard desktop PC, connected to the robot over wireless or wired network.

In order to measure the angle of each joint, the Nao is equipped with magnetic rotary encoders based on the Hall effect. These sensor have a precision of 0.1° and can be used to estimate the joint configuration of the robot even when the motors are turned off, which is useful for kinesthetic teaching or for a kinematic odometry estimate as detailed in Section 3.2.

An inertial measurement unit (IMU) yields an estimate about the robot's orientation. Measurements from a two-axis gyroscope and a three-axis accelerometer are integrated to estimate the orientation of the robot's torso around the world x and y-axis (roll and pitch, respectively). The measurements of this small and lightweight IMU are quite noisy compared to the IMUs often used in robotics.

Two monocular cameras can be used as onboard vision sensor on the Nao, one pointing in horizontal direction, and one pointing 40° downwards. The downwards pointing camera can be used to observe the area in front of the robot's feet. For the hardware revision V3+ the camera fields of view do not overlap, while the overlap in V4 is too small to

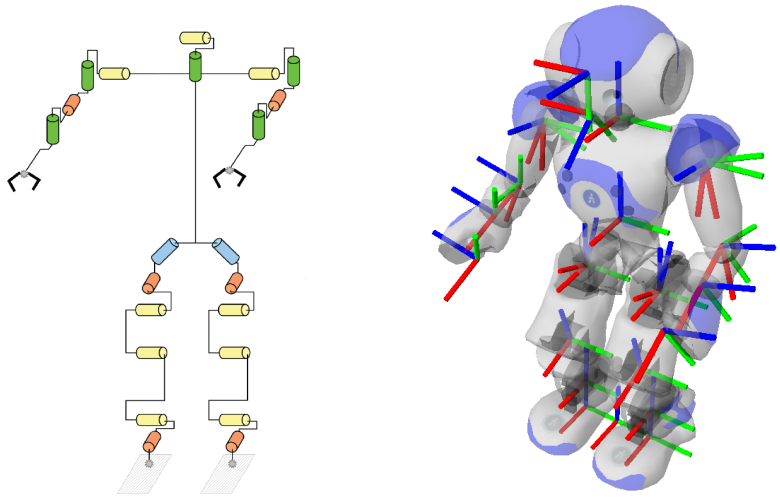


Figure A.1.: *Left:* Kinematic model of the Nao humanoid without wrist joint and hands (Source: Gouaillier et al. (2009)). *Right:* Coordinate frames of the Nao.

be used for stereo vision. Thus, additional sensors are required for a three-dimensional perception of the environment.

Throughout this work, we used the *NaoQI* API version 1.12. It provides access to the Nao's sensors and actuators, as well as a some high-level capabilities such as omni-directional walking (Gouaillier et al., 2010).

A.1. Range Sensor Configurations

In addition to the standard sensors, our Nao humanoids are equipped with two different range sensors, as illustrated in Figure A.2. Since they are mounted on the head, the robot can articulate them by means of the neck joints. For example, 3D points clouds can be obtained from the 2D laser by tilting the head. The first robot uses a Hokuyo URG-04LX laser range finder. This sensor provides 2D range data in a field of view of 240° at 0.33° resolution with an update rate of 10 Hz. The maximum range is 5.6 m. Compared to larger laser range finders, usually employed on a wheeled platform, this sensor is relatively noisy (Kneip et al., 2009).

The second Nao humanoid is equipped with a consumer-grade Asus Xtion Pro Live RGB-D camera, mounted on the robot's head such that its optical axis faces the floor in a 30° angle while walking. The camera is nearly identical to the popular Microsoft Kinect device, since both are based on the PrimeSense depth sensor. The sensor uses a projected infrared light pattern to obtain dense depth data. The camera has a field of view of 58° vertically and 45° horizontally with an operation range of 0.5 m to 5 m, in which the error grows quadratically with the distance (Khoshelham and Oude Elberink, 2012). At a resolution of 640×480, the sensor outputs depth data at 30 Hz.

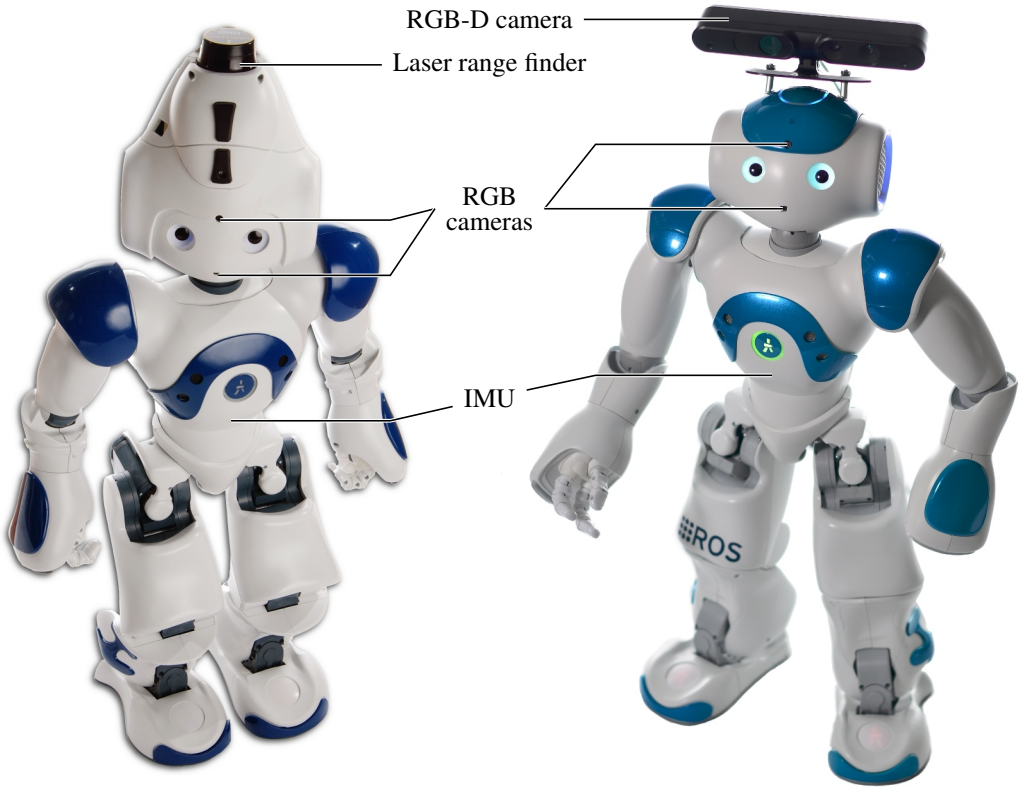


Figure A.2.: Nao humanoids and sensors used throughout this work. The Nao V3+ on the left was extended with a Hokuyo URG-04LX laser range finder, while the Nao V4 on the right was extended with an Asus Xtion RGB-D camera. Additionally, each joint of the Nao humanoid is equipped with Hall effect joint encoders.

List of Figures

List of Tables

List of Algorithms

Bibliography

S. Ahn, S. Yoon, S. Hyung, N. Kwak, and K. Roh. On-board odometry estimation for 3D vision-based SLAM of humanoid robot. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2012.

P. F. Alcantarilla, O. Stasse, S. Druon, L. M. Bergasa, and F. Dellaert. How to localize humanoids with a single camera? *Autonomous Robots*, 34(1-2):47–71, 2013. doi: 10.1007/s10514-012-9312-1.

J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. In *Proceedings of Eurographics*, Amsterdam, The Netherlands, August 1987.

H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf. Parametric correspondence and chamfer matching: two new techniques for image matching. In *Proc. of the Int. Conf. on Artificial Intelligence (IJCAI)*, 1977.

L. Baudouin, N. Perrin, T. Moulard, O. Stasse, F. Lamiraux, and E. Yoshida. Real-time replanning using 3d environment for humanoid robot. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2011.

P. Ben-Tzvi, S. Charifa, and M. Shick. Extraction of 3D images using pitch-actuated 2D laser range finder for robotic vision. In *Proc. of the IEEE Int. Workshop on Robotic and Sensors Environments (ROSE)*, 2010.

M. Bennewitz, C. Stachniss, W. Burgard, and S. Behnke. Metric localization with scale-invariant visual features using a single perspective camera. In *European Robotics Symposium 2006*, volume 22 of *STAR Springer tracts in advanced robotics*, 2006.

M. Bennewitz, D. Maier, A. Hornung, and C. Stachniss. Integrated perception and navigation in complex indoor environments. In *Proc. of the HUMANOIDS 2011 workshop on Humanoid service robot navigation in crowded and dynamic environments*, 2011.

D. Berenson, J. Chestnutt, S. Srinivasa, J. Kuffner, and S. Kagami. Pose-constrained whole-body planning using task space region chains. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2009.

M. Botsch, A. Wiratanaya, and L. Kobbelt. Efficient high quality rendering of point sampled geometry. In *Proc. of the 13th Eurographics Workshop on Rendering*, 2002.

J.-M. Bourgeot, N. Cislo, and B. Espiau. Path-planning and tracking in a 3D complex environment for an anthropomorphic biped robot. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2002.

A. Bry, A. Bachrach, and N. Roy. State estimation for aggressive flight in GPS-denied environments using onboard sensing. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012.

W. Burgard, A. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 114(1-2), 2000.

F. Burget, A. Hornung, and M. Bennewitz. Whole-body motion planning for manipulation of articulated objects. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

S. Candido, Y.-T. Kim, and S. Hutchinson. An improved hierarchical motion planner for humanoid robots. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2008.

J. F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:679–698, 1986.

J. F. Canny. *The complexity of robot motion planning*. MIT Press, Cambridge, MA, USA, 1988. ISBN 0-262-03136-1.

T. Chen, M. Ciocarlie, S. Cousins, P. M. Grice, K. Hawkins, K. Hsiao, C. Kemp, C.-H. King, D. Lazewatsky, A. E. Leeper, H. Nguyen, A. Paepcke, C. Pantofaru, W. Smart, and L. Takayama. Robots for humanity: A case study in assistive mobile manipulation. *IEEE Robotics & Automation Magazine, Special issue on Assistive Robotics*, 20, 2013.

J. Chestnutt and J. Kuffner. A tiered planning strategy for biped navigation. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2004.

J. Chestnutt, J. Kuffner, K. Nishiwaki, and S. Kagami. Planning biped navigation strategies in complex environments. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2003.

J. Chestnutt, M. Lau, K. M. Cheung, J. Kuffner, J. K. Hodgins, and T. Kanade. Footstep planning for the Honda ASIMO humanoid. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2005.

J. Chestnutt, K. Nishiwaki, J. Kuffner, and S. Kagami. An adaptive action model for legged navigation planning. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2007.

J. Chestnutt, Y. Takaoka, K. Suga, K. Nishiwaki, J. Kuffner, and S. Kagami. Biped navigation in rough environments using on-board sensing. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

S. Chitta, B. Cohen, and M. Likhachev. Planning for autonomous door opening with a mobile manipulator. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

S. Chitta, E. G. Jones, M. Ciocarlie, and K. Hsiao. Perception, planning, and execution for mobile manipulation in unstructured environments. *IEEE Robotics & Automation Magazine*, 19(2):58–71, 2012a.

S. Chitta, I. A. Şucan, and S. Cousins. MoveIt! [ROS topics]. *IEEE Robotics & Automation Magazine*, 19(1):18 –19, 2012b.

M. Ciocarlie, K. Hsiao, E. G. Jones, S. Chitta, R. B. Rusu, and I. A. Şucan. Towards reliable grasping and manipulation in household environments. In *Intl. Symposium on Experimental Robotics (ISER)*, 2010.

D. Cole and P. Newman. Using laser range data for 3D SLAM in outdoor environments. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2006.

R. Cupec, G. Schmidt, and O. Lorch. Experiments in vision-guided robot walking in a structured scenario. In *Proc. of the IEEE Int. Symp. on Industrial Electronics*, 2005.

S. Dalibard, A. Nakhaei, F. Lamiraux, and J.-P. Laumond. Whole-body task planning for a humanoid robot: a way to integrate collision avoidance. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2009.

S. Dalibard, A. Nakhaei, F. Lamiraux, and J.-P. Laumond. Manipulation of documented objects by a walking humanoid robot. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2010.

S. Dalibard, A. El Khouryz, F. Lamiraux, M. Taix, and J.-P. Laumond. Small-space controllability of a walking humanoid robot. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2011.

F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte Carlo localization for mobile robots. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1998.

R. Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.

R. Diankov, S. Srinivasa, D. Ferguson, and J. Kuffner. Manipulation planning with caging grasps. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2008.

J. Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. Technical report, Stanford University, 2006.

C. Dornhege and A. Hertle. Integrated symbolic planning in the tidyup-robot project. In *AAAI Spring Symposium - Designing Intelligent Robots: Reintegrating AI II*, 2013.

C. Dornhege and A. Kleiner. A frontier-void-based approach for autonomous exploration in 3D. *Advanced Robotics*, 27(6):459–468, Feb 2011.

C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel. Semantic attachments for domain-independent planning systems. In *Proc. of the Int. Conf. on Automated Planning and Scheduling (ICAPS)*. AAAI Press, 2009.

A. Doucet. On sequential simulation-based methods for bayesian filtering. Technical report, Signal Processing Group, Dept. of Engeneering, University of Cambridge, 1998.

A. Doucet, S. Godsill, and C. Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10:197–208, 2000.

B. Douillard, J. Underwood, N. Melkumyan, S. Singh, S. Vasudevan, C. Brunner, and A. Quadros. Hybrid elevation maps: 3D surface models for segmentation. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

I. Dryanovski, W. Morris, and J. Xiao. Multi-volume occupancy grids: An efficient probabilistic 3D mapping model for micro aerial vehicles. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

E. Einhorn, C. Schröter, and H.-M. Groß. Attention-driven monocular scene reconstruction for obstacle detection, robot navigation and map building. *Robotics & Autonomous Systems*, 59(5):296–309, 2011.

A. El Khoury, F. Lamiraux, and M. Taix. Optimal motion planning for humanoid robots. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2013.

A. I. Eliazar and R. Parr. Learning probabilistic motion models for mobile robots. In *Proc. of the International Conference on Machine Learning (ICML)*, 2004.

J. Elseberg, D. Borrmann, and A. Nüchter. Efficient processing of large 3d point clouds. In *Proc. of the XXIII Int. Symp. on Information, Communication and Automation Technologies (ICAT '11)*, 2011.

F. Faber, M. Bennewitz, C. Eppner, A. Goeroeg, A. Gonsior, D. Joho, M. Schreiber, and S. Behnke. The humanoid museum tour guide Robotinho. In *Proc. of the 18th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2009.

N. Fairfield, G. Kantor, and D. Wettergreen. Real-time SLAM with octree evidence grids for exploration in underwater tunnels. *Journal of Field Robotics*, 2007.

J. Fournier, B. Ricard, and D. Laurendeau. Mapping and exploration of complex environments using persistent 3D model. In *Computer and Robot Vision, 2007. Fourth Canadian Conf. on*, pages 403–410, 2007.

J. Garimort, A. Hornung, and M. Bennewitz. Humanoid navigation with dynamic footstep plans. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

D. Girardeau-Montaut, M. Roux, R. Marc, and G. Thibault. Change detection on points cloud data acquired with a ground laser scanner. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36:30–35, 2005.

K. Gochev, B. Cohen, J. Butzke, A. Safonova, and M. Likhachev. Path planning with adaptive dimensionality. In *Fourth Annual Symposium on Combinatorial Search*, 2011.

K. Gochev, A. Safonova, and M. Likhachev. Incremental planning with adaptive dimensionality. In *Proc. of the Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2013.

S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. In *Proc. of ACM SIGGRAPH*, 1996.

D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, and B. Maisonnier. Mechatronic design of NAO humanoid. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009.

D. Gouaillier, C. Collette, and C. Kilner. Omni-directional closed-loop walk for NAO. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2010.

B. Graf, M. Hans, and R. Schraft. Mobile robot assistants. *IEEE Robotics & Automation Magazine*, 11(2):67–77, 2004. doi: 10.1109/MRA.2004.1310943.

B. Graf, U. Reiser, M. Hägele, K. Mauz, and P. Klein. Robotic home assistant Care-O-bot 3 - product vision and innovation platform. In *Advanced Robotics and its Social Impacts (ARSO)*, 2009.

C. Gramkow. On averaging rotations. *Journal of Mathematical Imaging and Vision*, 15 (1-2):7–16, 2001.

G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with Rao-Blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007.

J.-S. Gutmann, M. Fukuchi, and M. Fujita. Stair climbing for humanoid robots using stereo vision. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2004.

J.-S. Gutmann, M. Fukuchi, and M. Fujita. Real-time path planning for humanoid robot navigation. In *Int. Joint Conference on Artificial Intelligence (IJCAI)*, Edinburgh, Scotland, 2005.

J.-S. Gutmann, M. Fukuchi, and M. Fujita. 3D perception and environment map generation for humanoid robot navigation. *Int. Journal of Robotics Research (IJRR)*, 27(10): 1117–1134, 2008.

M. Habbecke and L. Kobbelt. A surface-growing approach to multi-view stereo reconstruction. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2007.

R. Hadsell, J. A. Bagnell, and M. Hebert. Accurate rough terrain estimation with space-carving kernels. In *Proc. of Robotics: Science and Systems (RSS)*, 2009.

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

P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2): 100–107, Jul. 1968.

K. Hauser, T. Bretl, and J.-C. Latombe. Non-gaited humanoid locomotion planning. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2005.

K. Hauser, T. Bretl, J.-C. Latombe, K. Harada, and B. Wilcox. Motion planning for legged robots on varied terrain. *Int. Journal of Robotics Research (IJRR)*, 2007.

M. Hebert, C. Caillas, E. Krotkov, I. S. Kweon, and T. Kanade. Terrain mapping for a roving planetary explorer. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, May 1989.

L. Heng, L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys. Autonomous obstacle avoidance and maneuvering on a vision-guided MAV using on-board processing. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

C. Hertzberg, R. Wagner, O. Birbach, T. Hammer, and U. Frese. Experiences in building a visual slam system from open source components. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka. The development of Honda humanoid robot. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1998.

A. Hoover, G. Jean-Baptiste, X. Jiang, P. J. Flynn, H. Bunke, D. B. Goldgof, K. K. Bowyer, D. W. Eggert, A. W. Fitzgibbon, and R. B. Fisher. An experimental comparison of range image segmentation algorithms. *Transactions on Pattern Analysis and Machine Intelligence*, 18(7):673 – 689, 1996.

A. Hornung and M. Bennewitz. Robust and adaptive navigation with humanoid robots. In *Proceedings of the Workshop on Motion Planning: From Theory to Practice at Robotics: Science and Systems (RSS) Conference*, 2012a.

A. Hornung and M. Bennewitz. Adaptive level-of-detail planning for efficient humanoid navigation. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012b.

A. Hornung, M. Bennewitz, and W. Burgard. Learning efficient vision-based navigation. In *Proc. of the Int. Conference on Indoor Positioning and Indoor Navigation (IPIN)*, 2010a.

A. Hornung, M. Bennewitz, C. Stachniss, H. Strasdat, S. Oßwald, and W. Burgard. Learning adaptive navigation strategies for resource-constrained systems. In *Proc. of the 3rd Int. Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS)*, 2010b.

A. Hornung, M. Bennewitz, and H. Strasdat. Efficient vision-based navigation – Learning about the influence of motion blur. *Autonomous Robots*, 29:137–149, 2010c. doi: 10.1007/s10514-010-9190-3.

A. Hornung, K. M. Wurm, and M. Bennewitz. Humanoid robot localization in complex indoor environments. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010d.

A. Hornung, E. G. Jones, S. Chitta, M. Bennewitz, M. Phillips, and M. Likhachev. Towards navigation in three-dimensional cluttered environments. In *Proc. of the IROS 2011 PR2 Workshop: Results, Challenges and Lessons Learned in Advancing Robots with a Common Platform*, 2011.

A. Hornung, A. Dornbush, M. Likhachev, and M. Bennewitz. Anytime search-based footstep planning with suboptimality bounds. In *Proc. of the IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS)*, 2012a.

A. Hornung, M. Phillips, E. G. Jones, M. Bennewitz, M. Likhachev, and S. Chitta. Navigation in three-dimensional cluttered environments for mobile manipulation. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012b.

A. Hornung, M. Phillips, E. G. Jones, M. Bennewitz, M. Likhachev, and S. Chitta. Navigation in three-dimensional cluttered environments for mobile manipulation. In *Proceedings of the Workshop on Robots in Clutter: Manipulation, Perception and Navi-*

gation in Human Environments at Robotics: Science and Systems (RSS) Conference, 2012c.

A. Hornung, D. Maier, and M. Bennewitz. Search-based footstep planning. In *Proc. of the ICRA Workshop on Progress and Open Problems in Motion Planning and Navigation for Humanoids*, 2013a.

A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34:189–206, 2013b. doi: 10.1007/s10514-012-9321-0.

A. Hornung, S. Oßwald, D. Maier, and M. Bennewitz. Monte Carlo localization for humanoid robot navigation in complex indoor environments. *International Journal of Humanoid Robotics*, 2014. To appear.

W. Huang, J. Kim, and C. G. Atkeson. Energy-based optimal step planning for humanoids. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2013.

J. Ido, Y. Shimizu, Y. Matsumoto, and T. Ogasawara. Indoor navigation for a humanoid robot using a view sequence. *Int. Journal of Robotics Research (IJRR)*, 28(2):315–325, 2009.

P. G. Jayasekara, G. Ishigami, and T. Kubota. Testing and validation of autonomous navigation for a planetary exploration rover using opensource simulation tools. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2012.

X.-Y. Jiang and H. Bunke. Fast segmentation of range images into planar regions by scan line grouping. *Machine Vision and Applications*, 7(2):115 – 122, 1994.

P. Kaiser, D. Berenson, N. Vahrenkamp, T. Asfour, R. Dillmann, and S. Srinivasa. Constellation - an algorithm for finding robot configurations that satisfy multiple constraints. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012.

S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2003.

J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. G. Steinbach. Real-time compression of point cloud streams. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012.

F. Kanehiro, E. Yoshida, and K. Yokoi. Efficient reaching motion planning and execution for exploration by humanoid robots. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2012.

O. Kanoun, F. Lamiraux, P.-B. Wieber, F. Kanehiro, E. Yoshida, and J.-P. Laumond. Prioritizing linear equality and inequality systems: application to local motion planning for redundant robots. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009.

O. Kanoun, J.-P. Laumond, and E. Yoshida. Planning foot placements for a humanoid robot: A problem of inverse kinematics. *Int. Journal of Robotics Research (IJRR)*, 30 (4):476–485, 2011.

S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. Journal of Robotics Research (IJRR)*, 30(7):846–894, 2011. doi: 10.1177/0278364911406761.

S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller. Anytime motion planning using the RRT*. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

L. E. Kavraki, P. Svestka, J. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. on Robotics and Automation*, 12(4):566–580, 1996.

A. Kelly, A. Stentz, O. Amidi, M. Bode, D. M. Bradley, A. Diaz-Calderon, M. Happold, H. Herman, R. Mandelbaum, T. Pilarski, P. Rander, S. Thayer, N. Vallidis, and R. Warner. Toward reliable off road autonomous vehicles operating in challenging environments. *Int. Journal of Robotics Research*, 25(5-6):449–483, 2006.

K. Khoshelham and S. Oude Elberink. Accuracy and resolution of Kinect depth data for indoor mapping applications. *Sensors: Journal on the Science and Technology of Sensors and Biosensors*, 12:1437–1454, 2012.

Y. Kida, S. Kagami, T. Nakata, M. Kouchi, and H. Mizoguchi. Human finding and body property estimation by using floor segmentation and 3D labelling. In *Proc. of the IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC)*, 2004.

K. Klasing, D. Wollherr, and M. Buss. A clustering method for efficient segmentation of 3d laser data. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2008.

K. Klasing, D. Wollherr, and M. Buss. Realtime segmentation of range data using continuous nearest neighbors. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009.

L. Kneip, F. Tâche, G. Caprari, and R. Siegwart. Characterization of the compact Hokuyo URG-04LX 2D laser range scanner. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009.

S. Koenig and M. Likhachev. D* Lite. In *Proc. of the National Conf. on Artificial Intelligence (AAAI)*, 2002.

K. Konolige. Projected texture stereo. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

G. Kraetzschmar, G. Gassull, and K. Uhl. Probabilistic quadtrees for variable-resolution mapping of large environments. In *Proc. of the 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles*, 2004.

M. Krainin, P. Henry, X. Ren, and D. Fox. Manipulator and object tracking for in-hand 3d object modeling. *Int. Journal of Robotics Research*, 30(11):1311–1327, Sept. 2011. ISSN 0278-3649.

J. J. Kuffner and S. M. Lavalle. RRT-Connect: An efficient approach to single-query path planning. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2000.

J. J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Footstep planning among obstacles for biped robots. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2001.

J. J. Kuffner, S. Kagami, K. Nishiwaki, M. Inaba, and H. Inoue. Dynamically-stable motion planning for humanoid robots. *Autonomous Robots*, 12:105–118, 2002.

R. Kümmerle, R. Triebel, P. Pfaff, and W. Burgard. Monte Carlo localization in outdoor terrains using multilevel surface maps. *Journal of Field Robotics (JFR)*, 25:346–359, 2008.

R. Kümmerle, D. Hähnel, D. Dolgov, S. Thrun, and W. Burgard. Autonomous driving in a multi-level parking structure. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009.

R. Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard. A navigation system for robots operating in crowded urban environments. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2013.

Y. Kuwata, G. Fiore, J. Teo, E. Frazzoli, and J. How. Motion planning for urban driving using RRT. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2008.

N. Kwak, O. Stasse, T. Foissotte, and K. Yokoi. 3D grid and particle based slam for a humanoid robot. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2009.

S. Laine and T. Karras. Efficient sparse voxel octrees. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 2010.

B. Lau, C. Sprunk, and W. Burgard. Incremental updates of configuration space representations for non-circular mobile robots with 2d, 2.5d, or 3d obstacle models. In *Proc. of the European Conf. on Mobile Robots (ECMR)*, 2011.

B. Lau, C. Sprunk, and W. Burgard. Efficient grid-based spatial representations for robot navigation in dynamic environments. *Robotics and Autonomous Systems*, 61(10):1116 – 1130, 2013. doi: 10.1016/j.robot.2012.08.010.

S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

A. Leeper, K. Hsiao, M. Ciocarlie, I. A. Şucan, and K. Salisbury. Arm teleoperation in clutter using virtual constraints from real sensor data. In *RSS Workshop on Robots in Clutter: Preparing Robots for the Real World*, 2013.

T.-Y. Li, P.-F. Chen, and P.-Z. Huang. Motion planning for humanoid walking in a layered environment. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2003.

M. Likhachev. Search-based planning library. http://www.ros.org/wiki/sbpl, 2010.

M. Likhachev and D. Ferguson. Planning long dynamically-feasible maneuvers for autonomous vehicles. In *Int. Journal of Robotics Research (IJRR)*, 2009.

M. Likhachev and A. Stentz. R* search. In *Proc. of the National Conf. on Artificial Intelligence (AAAI)*, 2008.

M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Proc. of the Conf. on Neural Information Processing Systems (NIPS)*, 2004.

M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime dynamic A*: An anytime, replanning algorithm. In *Proc. of the Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2005.

H. Liu, Q. Sun, and T. Zhang. Hierarchical RRT for humanoid robot footstep planning with multiple constraints in complex environments. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2012.

C. Lutz, F. Atmanspacher, A. Hornung, and M. Bennewitz. Nao walking down a ramp autonomously. In *Video Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

M. Magnusson, T. Duckett, and A. J. Lilienthal. Scan registration for autonomous mining vehicles using 3D-NDT. *Journal of Field Robotics*, 24(10):803–827, Oct 2007.

D. Maier, A. Hornung, and M. Bennewitz. Real-time navigation in 3D environments based on depth camera data. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2012.

D. Maier, C. Lutz, and M. Bennewitz. Integrated perception, mapping, and footstep planning for humanoid navigation among 3D obstacles. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2013.

N. Mansard, O. Stasse, F. Chaumette, and K. Yokoi. Visually-guided grasping while walking on a humanoid robot. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2007.

E. Marder-Eppstein, E. Berger, T. Foote, B. P. Gerkey, and K. Konolige. The office marathon: Robust navigation in an indoor office environment. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

J. Mason, S. Ricco, and R. Parr. Textured occupancy grids for monocular localization without features. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

J. Matas, C. Galambos, and J. Kittler. Robust detection of lines using the progressive probabilistic Hough transform. *Computer Vision and Image Understanding*, 78:119–137, Apr. 2000. doi: 10.1006/cviu.1999.0831.

D. Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129–147, 1982.

A. Mertens, U. Reiser, B. Brenken, M. Lüdtke, M. Hägele, A. Verl, C. Brandl, and C. Schlick. Assistive robots in eldercare and daily living: Automation of individual services for senior citizens. In *Intelligent Robotics and Applications*, volume 7101 of *Lecture Notes in Computer Science*, pages 542–552. Springer Berlin Heidelberg, 2011. doi: 10.1007/978-3-642-25486-4_54.

P. Michel, J. Chestnutt, S. Kagami, K. Nishiwaki, J. Kuffner, and T. Kanade. Online environment reconstruction for biped navigation. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2006.

P. Michel, J. Chestnutt, S. Kagami, K. Nishiwaki, J. Kuffner, and T. Kanade. GPU-accelerated real-time 3D tracking for humanoid locomotion and stair climbing. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007.

M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proc. of the Int. Conf. on Artificial Intelligence (IJCAI)*, 2003.

M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato. A machine learning approach for feature-sensitive motion planning. In *Int. Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2004.

H. Moravec. Robot spatial perception by stereoscopic vision and 3D evidence grids. Technical Report CMU-RI-TR-96-34, Robotics Institute, Pittsburgh, PA, September 1996.

H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1985.

H. P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, pages 61–74, 1988.

J. Müller, N. Kohler, and W. Burgard. Autonomous miniature blimp navigation with online motion planning and re-planning. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2011.

J. Müller, U. Frese, and T. Röfer. Grab a mug - Object detection and grasp motion planning with the Nao robot. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2012.

R. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*, 2011.

K. Nishiwaki, S. Kagami, Y. Kuniyoshi, M. Inaba, and H. Inoue. Toe joints that enhance bipedal and fullbody motion of humanoid robots. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2002.

K. Nishiwaki, J. Chestnutt, and S. Kagami. Autonomous navigation of a humanoid robot over unknown rough terrain using a laser range sensor. *Int. Journal of Robotics Research (IJRR)*, 31(11):1251–1262, 2012.

S. Nozawa, Y. Kakiuchi, K. Okada, and M. Inaba. Controlling the planar motion of a heavy object by pushing with a humanoid robot using dual-arm force control. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012.

A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann. 6D SLAM – 3D mapping outdoor environments. *Journal of Field Robotics*, 24(8-9):699–722, 2007. doi: 10.1002/rob.20209.

ODE, 2011. Open dynamics engine. http://www.ode.org, retrieved Feb. 2011.

K. Okada, S. Kagami, M. Inaba, and H. Inoue. Plane segment finder: Algorithm, implementation and applications. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2001.

K. Okada, T. Ogura, A. Haneda, and M. Inaba. Autonomous 3D walking system for a humanoid robot based on visual step recognition and 3D foot step planner. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2005.

G. Oriolo and M. Vendittelli. A control-based approach to task-constrained motion planning. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

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.

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 and Systems (IROS)*, 2011a.

S. Oßwald, J.-S. 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)*, 2011b.

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 and Systems (IROS)*, 2012.

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.

G. Papadopoulos, H. Kurniawati, A. Shariff, L. Wong, and N. Patrikalakis. Experiments on surface reconstruction for partially submerged marine structures. *Journal of Field Robotics*, 31(2):225–244, 2014. doi: 10.1002/rob.21478.

K. Pathak, A. Birk, J. Poppinga, and S. Schwertfeger. 3D forward sensor modeling and application to occupancy grid based sensor fusion. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007.

P. Payeur, P. Hebert, D. Laurendeau, and C. Gosselin. Probabilistic octree modeling of a 3-d dynamic environment. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1997.

J. Pearl. *Heuristics: Intelligent search strategies for computer problem solving*. Addison-Wesley, Reading, MA, 1984.

N. Perrin, O. Stasse, F. Lamiraux, and E. Yoshida. Weakly collision-free paths for continuous humanoid footstep planning. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2011.

N. Perrin, O. Stasse, L. Baudouin, F. Lamiraux, and E. Yoshida. Fast humanoid robot collision-free footstep planning using swept volume approximations. *IEEE Transactions on Robotics*, 28:427–439, 2012a.

N. Perrin, O. Stasse, F. Lamiraux, Y. J. Kim, and D. Manocha. Real-time footstep planning for humanoid robots among 3D obstacles using a hybrid bounding box. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012b.

J. Pettré, J.-P. Laumond, and T. Siméon. A 2-stages locomotion planner for digital actors. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2003.

P. Pfaff, R. Triebel, C. Stachniss, P. Lamon, W. Burgard, and R. Siegwart. Towards mapping of cities. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2007.

M. Phillips, A. Dornbush, S. Chitta, and M. Likhachev. Anytime incremental planning with E-Graphs. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2013.

M. Pivtoraiko and A. Kelly. Generating near minimal spanning control sets for constrained motion planning in discrete state spaces. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2005.

M. Pollack, S. Engberg, J. Matthews, S. Thrun, L. Brown, D. Colbry, C. Orosz, B. Peintner, S. Ramakrishnan, J. Dunbar-Jacob, C. McCarthy, M. Montemerlo, J. Pineau, and N. Roy. Pearl: A mobile robotic assistant for the elderly. In *Workshop on Automation as Caregiver: the Role of Intelligent Technology in Elder Care (AAAI)*, 2002.

E. Pot, J. Monceaux, R. Gelin, and B. Maisonnier. Choregraphe: a graphical tool for humanoid robot programming. In *Proc. of the 18th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2009.

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.

S. Quinlan. Efficient distance computation between non-convex objects. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1994.

J. H. Reif. Complexity of the mover's problem and generalizations. In *Proceedings IEEE Symposium on Foundations of Computer Science*, 1979.

M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proc of the IEEE Int. Conf. on Neural Networks (ICNN)*, 1993.

Y. Roth-Tabak and R. Jain. Building an environment model using depth information. *Computer*, 22(6):85–90, Jun 1989.

T. Rühr, J. Sturm, D. Pangercic, M. Beetz, and D. Cremers. A generalized framework for opening doors and drawers in kitchen environments. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012.

R. B. Rusu, A. Sundaresan, B. Morisset, K. Hauser, M. Agrawal, J.-C. Latombe, and M. Beetz. Leaving flatland: Efficient real-time three-dimensional perception and motion planning. *J. Field Robotics*, 26(10):841–862, 2009.

J. Ryde and H. Hu. 3D mapping with multi-resolution occupied voxel lists. *Autonomous Robots*, 28(2):169–185, 2010.

R. Schnabel and R. Klein. Octree-based point-cloud compression. In *Symposium on Point-Based Graphics*. Eurographics, 2006.

J. Scholz, S. Chitta, B. Marthi, and M. Likhachev. Cart pushing with a mobile manipulation system: Towards navigation with moveable objects. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

J. Seara and G. Schmidt. Intelligent gaze control for vision-guided humanoid walking: methodological aspects. *Robotics & Autonomous Systems*, 48(4):231–248, 2004.

R. Shade and P. Newman. Choosing where to go: Complete 3D exploration with stereo. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

G. Sibley, C. Mei, I. Reid, and P. Newman. Adaptive relative bundle adjustment. In *Proc. of Robotics: Science and Systems (RSS)*, 2009.

M. Smith, I. Baldwin, W. Churchill, R. Paul, and P. Newman. The New College vision and laser data set. *Int. Journal of Robotics Research*, 28(5):595–599, May 2009. doi: 10.1177/0278364909103911.

C. Sprunk, B. Lau, P. Pfaff, and W. Burgard. Online generation of kinodynamic trajectories for non-circular omnidirectional robots. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

S. Srinivasa, D. Ferguson, J. M. Vandeweghe, R. Diankov, D. Berenson, C. Helfrich, and H. Strasdat. The robotic busboy: Steps towards developing a mobile robotic home assistant. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2008.

C. Stachniss, M. Bennewitz, G. Grisetti, S. Behnke, and W. Burgard. How to learn accurate grid maps with a humanoid. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2008.

O. Stasse, B. Verrelst, A. Davison, N. Mansard, F. Saidi, B. Vanderborght, C. Esteves, and K. Yokoi. Integrating walking and vision to increase humanoid autonomy. *Int. Journal of Humanoid Robotics (IJHR), special issue on Cognitive Humanoid Robots*, 5(2):287–310, 2008.

R. Steffens, M. Nieuwenhuisen, and S. Behnke. Multiresolution path planning in dynamic environments for the standard platform league. In *Workshop on Humanoid Soccer Robots at the IEEE-RAS Int. Conf. on Humanoid Robots*, 2010.

K. Steinbach, J. Kuffner, T. Asfour, and R. Dillmann. Collision and self-collision detection for humanoids based on sphere tree hierarchies. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2006.

A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1994.

M. Stilman. Global manipulation planning in robot joint space with task constraints. *IEEE Trans. on Robotics and Automation*, 26(3):576–584, 2010.

M. Stilman, K. Nishiwaki, S. Kagami, and J. Kuffner. Planning and executing navigation among movable obstacles. *Springer Journal of Advanced Robotics*, 21(14):1617–1634, 2007.

T. Stoyanov, M. Magnusson, H. Andreasson, and A. J. Lilienthal. Path planning in 3d environments using the normal distributions transform. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

J. Sturm, C. Stachniss, and W. Burgard. A probabilistic framework for learning kinematic models of articulated objects. *Journal on Artificial Intelligence Research (JAIR)*, 41: 477–626, August 2011.

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 and Systems (IROS)*, 2012. http://cvpr.in.tum.de/data/datasets/rgbd-dataset/download.

I. A. Şucan and S. Chitta. Motion planning with constraints using configuration space approximations. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2012.

H. Surmann, A. Nüchter, and J. Hertzberg. An autonomous mobile robot with a 3d laser range finder for 3d exploration and digitalization of indoor environments. *Robotics and Autonomous Systems*, 45(3):181–198, 2003.

R. Tellez, F. Ferro, D. Mora, D. Pinyol, and D. Faconti. Autonomous humanoid navigation using laser and odometry data. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2008.

M. Tenorth and M. Beetz. KnowRob: A knowledge processing infrastructure for cognition-enabled robots. *Int. Journal of Robotics Research (IJRR)*, 32(5):566–590, 2013. doi: 10.1177/0278364913481635.

S. Thompson and S. Kagami. Humanoid robot localisation using stereo vision. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2005.

S. Thompson, S. Kagami, and K. Nishiwaki. Localisation for autonomous humanoid navigation. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2006.

S. Thrun. A probabilistic online mapping algorithm for teams of mobile robots. *Int. Journal of Robotics Research*, 20(5):335–363, 2001.

S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Minerva: A second-generation museum tour-guide robot. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1999.

S. Thrun, C. Martin, Y. Liu, D. Hähnel, R. Emery-Montemerlo, D. Chakrabarti, and W. Burgard. A real-time expectation maximization algorithm for acquiring multi-planar maps of indoor environments with mobile robots. *IEEE Trans. on Robotics and Automation*, 2003.

S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT-Press, 2005.

R. Triebel, W. Burgard, and F. Dellaert. Using hierarchical EM to extract planes from 3D range scans. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2005.

R. Triebel, P. Pfaff, and W. Burgard. Multi-level surface maps for outdoor terrain mapping and loop closing. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2006.

N. Vahrenkamp, T. Asfour, and R. Dillmann. Efficient motion planning for humanoid robots using lazy collision checking and enlarged robot models. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007.

N. Vahrenkamp, C. Scheurer, T. Asfour, J. J. Kuffner, and R. Dillmann. Adaptive motion planning for humanoid robots. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2008.

M. Vukobratovic and B. Borovac. Zero-moment point – thirty five years of its life. *Int. Journal of Humanoid Robots*, 1, 2004.

M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J. M. M. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle, and R. van de Molengraft. RoboEarth – A World Wide Web for robots. *IEEE Robotics & Automation Magazine*, 18(2):69–82, 2011. doi: 10.1109/MRA.2011.941632.

T. Weise, T. Wismer, B. Leibe, and L. Van Gool. In-hand scanning with online loop closure. In *ICCV Workshops*, 2009.

T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. McDonald. Robust real-time visual odometry for dense rgb-d mapping. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2013.

J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Trans. Graph.*, 11(3):201–227, 1992. ISSN 0730-0301.

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

K. Yamazaki, R. Ueda, S. Nozawa, M. Kojima, K. Okada, K. Matsumoto, M. Ishikawa, I. Shimoyama, and M. Inaba. Home-assistant robot for an aging society. *Proceedings of the IEEE*, 100(8):2429–2441, 2012. doi: 10.1109/JPROC.2012.2200563.

M. Yguel, O. Aycard, and C. Laugier. Update policy of dense maps: Efficient algorithms and sparse representation. In *Field and Service Robotics, Results of the Int. Conf., FSR 2007*, volume 42, pages 23–33, 2007a.

M. Yguel, C. T. M. Keat, C. Braillon, C. Laugier, and O. Aycard. Dense mapping for range sensors: Efficient algorithms and sparse representations. In *Proc. of Robotics: Science and Systems (RSS)*, 2007b.

E. Yoshida, O. Kanoun, C. Esteves, and J.-P. Laumond. Task-driven support polygon reshaping for humanoids. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2006.

T. Yoshikawa. Manipulability of robotic mechanisms. *Int. Journal of Robotics Research (IJRR)*, 4(2):3–9, 1985.

S. Zickler and M. Veloso. Variable level-of-detail motion planning in environments with poorly predictable bodies. In *Proc. of the European Conf. on Artificial Intelligence (ECAI)*, pages 189–194, 2010.