Institute of Parallel and Distributed Systems
University of Stuttgart
Universitätsstraße 38
70569 Stuttgart
Germany

Diplom Thesis No. 2909

Real-Time Motion Planning for a Robot Arm in Dynamic Environments

Tobias Kunz

Major:	Computer Science
Examiners:	PD Dr. Michael Schanz Prof. Dr. Alexander Verl
Advisor:	Dipl.-Ing. Ulrich Reiser
Commenced:	March 23, 2009
Completed:	August 6, 2009
CCS Classification:	I.2.8, I.2.9, I.2.10

Abstract

In order for robots to operate safely in human environments, they need to be able to react to a dynamically changing environment in real-time. This involves both fast processing of sensor data and fast motion planning based on the sensor data.

This work describes the implementation of a path planner for a 7-DOF robot arm, which is able to react to a dynamic environment very quickly. A SwissRanger SR3000 3D time-of-flight sensor is used to detect obstacles. This work uses the Probabilistic Roadmaps for Changing Environments algorithm introduced by Leven and Hutchinson. The algorithm generates a roadmap for an obstacle-free environment during a preprocessing stage. During the online planning stage blocked parts of the roadmap are invalidated and the roadmap is used for fast planning. Fast invalidation of blocked parts of the roadmap is accomplished through a mapping from occupied workspace grid cells to blocked roadmap parts. This workspace mapping is also generated during the preprocessing stage. It is shown that the implementation is able to plan within about 100 milliseconds while avoiding sensed obstacles, and, thus, is able to react to a dynamic environment. The path planner is implemented on the Care-O-bot 3, which is a home assistant robot developed at Fraunhofer IPA.

Contents

Chapter 1

Introduction

1.1 Motivation

Robots have been quite successful in accomplishing tasks in well-known environments like a work cell within a factory. The much harder problem of a robot acting in unstructured and dynamic environments, like those humans normally act and live in, is still an open research area. Care-O-bot 3, which is developed at the Fraunhofer Institute for Manufacturing Engineering and Automation (IPA) in Stuttgart, Germany, is supposed to work in such a unstructured and dynamic environment. Care-O-bot 3 is a mobile service robot which embodies the vision of a home assistant robot that can execute typical tasks of a butler. The tasks that are currently used to demonstrate the robot's abilities include serving a drink and opening a door. As the robot operates in the same environment side-by-side with humans it must be able to do so safely, i. e. it must be able to recognize dynamic obstacles like humans and move around them. Recognizing dynamic obstacles and adapting the motion accordingly must be be done fast as the environment changes quickly, too.

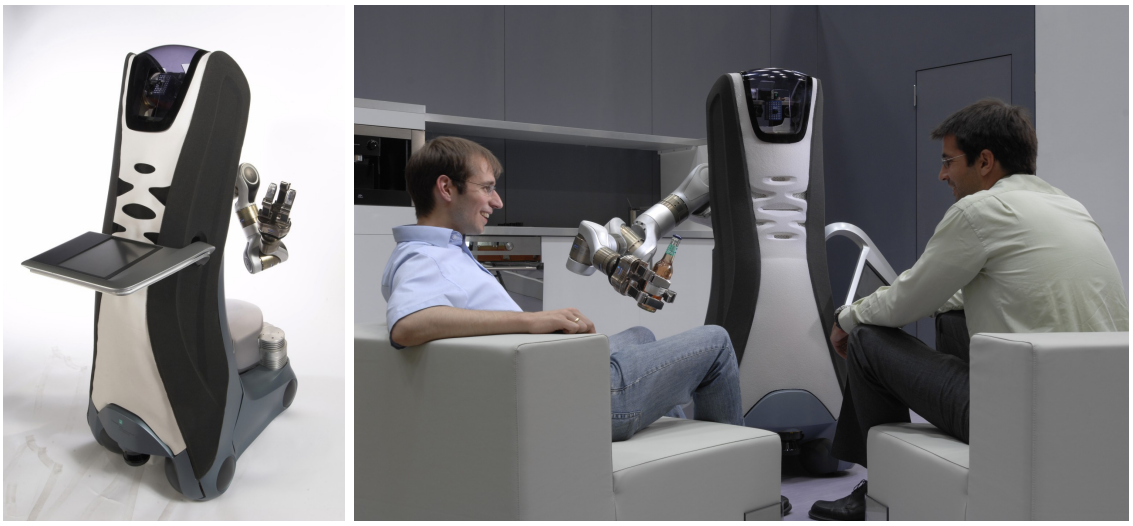


Figure 1.1: Care-O-bot 3

1.2 Current Situation

Planning for the robot base (3 DOF) and the arm (7 DOF) are currently independent of each other. While the arm is moving the base is not, and otherwise round.

Planning for the base is done using elastic bands [1, 2] to dynamically avoid unknown obstacles, e.g. humans walking around, in addition to known obstacles like walls or tables that have been modeled by hand.

In contrast, the planner for the robot arm currently neither uses sensor data to detect unknown obstacles nor is fast enough to be able to react to a dynamic environment. [3, 4] The current path planner uses the SBL [5] implementation of the Stanford Motion Planning Kit. It only checks for collision with a hand-crafted static model shown in figure 1.2. No sensors are used to detect unknown obstacles. The planning process takes about 2 seconds. The collision checker uses bounding boxes of the robot and the environment. It only returns whether the given configuration is in collision but no information on the distance to the closest obstacle. Checking one robot configuration for collision takes about 0.3 milliseconds.

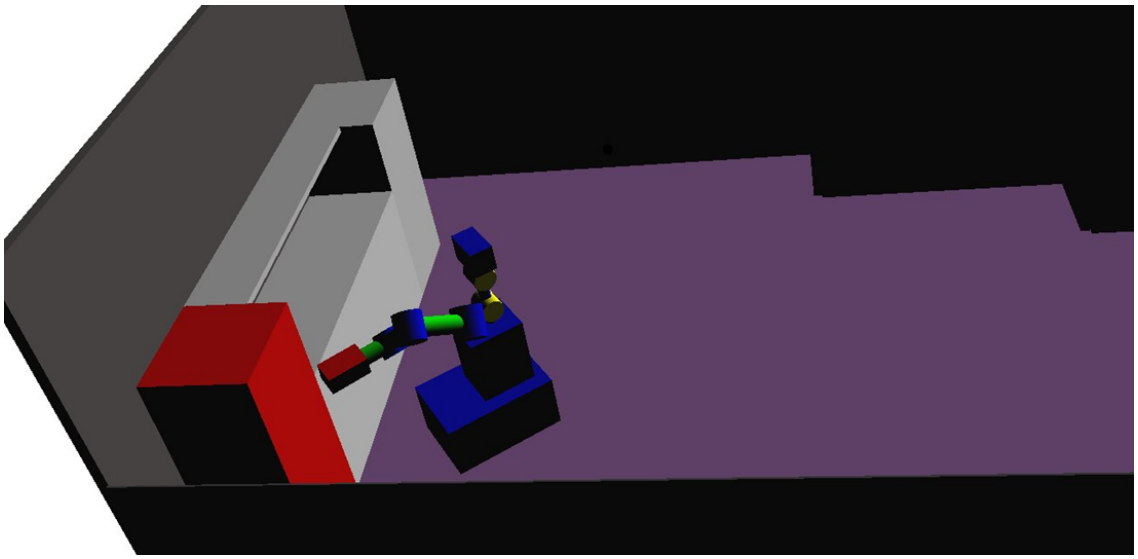


Figure 1.2: The environment model currently used for planning

1.3 Problem Description

The currently used path planner for the robot arm is not able to meet the requirements described in section 1.1. It ignores dynamic obstacles, and even if they would be included in the planning process, the planner is too slow to react to a changing environment. Thus, the goal of this work is to develop a new planner for the arm that is able to plan quickly and to take sensor data into account.

The Care-O-bot 3 has a SwissRanger SR3000 sensor mounted in its head, which is to be used by the new motion planner to detect dynamic obstacles. The SwissRanger

Figure 1.3: SwissRanger SR3000

SR3000 (see figure 1.3) provides a 3D depth image with a resolution of 176 x 144 pixels. Distance information is retrieved by measuring the time-of-flight of emitted infrared light.

The main problem to be solved here is that of developing a motion planner for the robot arm that is fast enough to react to a dynamic environment and, thus, is able to adapt the movement of the arm online while it is moving. The goal of this work is to achieve a planning rate of 5-10 Hz.

However, not only the actual planning must be fast. Also the acquiring and processing of the sensor data might take time and must be fast enough. Processing and representing the sensor data are not the core of this work. Though, different motion planners might have different requirements for the representation of sensor data. A motion planner that can directly use a point cloud retrieved from the sensor needs less preprocessing than one which only works with sensor data processed into a geometric model. A motion planner that only needs to know whether a particular configuration is in collision with obstacles recognized by the sensor might be faster than one that needs to know the distance to the closest obstacle. As the choice of a motion planner affects the representation of the sensor data and the amount of preprocessing necessary, it must be shown that not only the planner itself can run in real-time but also that it is able to integrate sensor data in real-time, which includes possibly necessary preprocessing.

1.4 Overview

Chapter 2 describes several planning algorithms that have been designed for fast motion planning in changing environments and evaluates them with respect to the posed problem. In section 2.4 PRM for Changing Environments (PRMCE) [6], which is the algorithm that is used in this work, is described. Later chapters build on top of this high-level description of the algorithm. In chapter 3 the decision to use PRMCE as basis for the implementation is motivated and the overall design of the implemented motion planner is explained. As metrics for configuration space have proved to be quite important for the quality of the resulting path as well as for the speed of the planning process, several different metrics are described in chapter 4. Chapter 5 presents the different steps of the algorithm and how they were implemented. Chapter 6 analyzes the implemented motion planner. Amongst other things it is shown that the planner is able to execute a planning query within about

100 milliseconds and, thus, is achieving the real-time requirements. It is also demonstrated that the planner is able to plan around a human that was detected by the SwissRanger sensor. In chapter 7 several ideas on how this work can be improved and continued are given. And finally, chapter 8 concludes this work.

Chapter 2

Related Work

Path planning in high-dimensional configuration spaces has been dramatically expedited through sampling based techniques, which give up optimality in order to achieve an acceptable performance. [7]

Sampling-based planners include SBL (Single-query Bidirectional PRM Planner with Lazy Collision Checking) [5] and RRT (Rapidly Exploring Random Trees) [7] planners. These planners plan from scratch every time they are queried for a plan. Hence, for every planning query they have to explore the configuration space and check possible path segments for collision. Since they always plan from scratch, these planners are suited for changing environments. However, they are too slow to fulfill real-time requirements. Especially checking the whole final path and possibly also other considered path segments for collision takes a lot of time.

Multi-query planners like the Probabilistic Roadmap (PRM) Planner assume that several different planning queries are to be executed within the same static environment. Therefore, they can spent a large amount of time for preprocessing to create a data structure that will speed-up the individual planning queries. Multi-query planners speed-up the planning process but since they assume a static environment during several planning queries, they are not suited for our situation.

In contrast to planning approaches, reactive control approaches do not create a global plan from the start to the goal configuration. Instead, they control the robot online based on local information. Such local information can be represented by a potential field [8]. The robot is then moved around by artificial forces based on the potential field. The potential field is designed to attract the robot toward the goal and repulse it from obstacles. As reactive control is only based on local information it sometimes fails to guide the robot to the goal although there exists a collision-free path to the goal. With potential field methods this happens when the robot gets stuck in a local minimum of the potential field. Although there have been different solutions proposed to counter that problem, the missing global information stays an inherent drawback of reactive control approaches.

As none of the basic motion planning methods described above is well suited for our situation, I looked at five approaches, which explicitly deal with fast motion planning for high-dimensional configuration spaces in dynamic and unpredictably changing environ-

ments. These five approaches are:

- Elastic Bands [1, 2]
- Elastic Strips [9, 10]
- Decomposition-Based Motion Planning [11, 12]
- PRM for Changing Environments (PRMCE) [6]
- Flexible Anytime Dynamic PRM (FADPRM) [13]

Each of these approaches tries to achieve the real-time requirements very differently. Elastic bands and elastic strips combine path planning with reactive control to obtain a solution with the speed of reactive control approaches and a global view of planning approaches. Decomposition-based motion planning tries to achieve the real-time requirements by decomposing the planning problem into two simpler ones, one in workspace and one in configuration space. PRM for changing environments adapts the PRM approach in order to benefit from preprocessing also in changing environments. Flexible anytime dynamic PRM tries to achieve the real-time requirements by reusing the previously planned path and by accepting suboptimal solutions.

I did not take approaches into account that require a geometric representation of the environment, like for example Elastic Roadmaps [14] does. Acquiring the necessary geometric representation needs additional time and would make the planning process much slower. Additionally, it does not make sense to create a geometric model of the environment just to avoid obstacles. If the model were created anyway for some other purpose, it could be reasonable to also use it for avoiding obstacles. But that is currently not the case.

2.1 Elastic Bands

Published 1992-1995. PhD work by Sean Quinlan advised by Oussama Khatib at Stanford.

The Elastic Bands framework [1, 2] combines global path planning with reactive control. To avoid the problems inherent in reactive control approaches, the control system implemented here does not act directly on the robot but instead deforms a collision-free path while the robot is moving along this path. The initial collision-free path is created by an arbitrary path planner. This deforming collision-free path is called an elastic band.

To make sure that the path stays collision-free while it is deformed, the free space around the path needs to be represented. Calculating and representing free space in the high-dimensional configuration space is either computationally expensive or very conservative. As the computation needs to be fast for real-time execution, elastic bands use a conservative approach, underestimating the actual free space. This is done by overlapping convex regions of free space, called bubbles, centered on the path. Each bubble represents some free space around a given configuration. The size of the bubbles in configuration space is calculated based on the distance from the robot to the closest obstacle in workspace.

$$\beta_q = \{p : d_r(p, q) < d_{obs}\}$$

Where $d_r(p, q)$ is the metric described in section 4.4, which gives an upper bound on the maximum distance traveled, and d_{obs} is the distance from the robot to the closest obstacle. This ensures that all configurations within the ball are collision-free. The elastic band is represented as a finite set of particles, which are center points of bubbles. A collision-free path is guaranteed to exist if the bubbles around neighboring particles overlap.

The elastic band is deformed by applying artificial forces on the particles. There is an internal and an external force. The internal force corresponds to the contraction force in a real elastic band and smooths the path. The external force repulses the particle from obstacles. The external force is defined by the workspace vector d that is the shortest vector from the obstacles to the robot. $d = x - y$ where x is a point on the robot and y is a point of the obstacle region. The workspace force is mapped into configuration space by the Jacobian transpose at point x.

$$f_{ext} = k_r J_x^T (d_o - \|d\|) \frac{d}{\|d\|}$$

The elastic band is deformed iteratively. In each iteration all particles are moved by a small amount based on the artificial forces acting on them. To ensure that the path stays collision-free after moving the particle, the motion of the particle is restricted to stay within its old bubble. After moving the particles their bubbles are recalculated. This may lead to neighboring bubbles not overlapping anymore. In such a case additional particles with bubbles are inserted until all neighboring bubbles overlap. Also, bubbles that got redundant may be removed.

There is a simple way to calculate a collision-free path out of the particles: We choose a shared configuration between each pair of neighboring bubbles. Such a shared configuration exists because neighboring bubbles overlap. By connecting two neighboring particles with two straight line segments through the shared configuration, we get a collision-free path. Although this is a simple way to generate a collision-free path, the generated path is not smooth and the robot will have problems executing it. Therefore elastic bands uses a better approach based on cubic B-splines. The splines are generated such that they always stay within the bubbles.

Elastic bands are a good way to achieve real-time constraints because they avoid time-intensive path planning by just adopting the existing path using control methods. However, there are is also a drawback. The performance of the algorithm depends on the number of bubbles necessary to represent the collision-free path. The size of the bubbles in configuration space is calculated based on the distance to the closest obstacle in workspace. This underestimates the actual possible size of the bubble. How much the size of the bubble is underestimated depends on the situation, but the likelihood of the size of the bubble being underestimated by a large margin increases with more DOF. This makes the algorithm scale not very well with DOF. [9] In our domain, if we wanted to include the movement of the robot base into the planning process, the robot base being close to an obstacle like a table would slow down the motion planning for the arm.

2.2 Elastic Strips

Published 1997-2002. PhD work by Oliver Brock advised by Oussama Khatib at Stanford.

The elastic strips approach [9, 10] is a direct successor of elastic bands. It improves elastic bands in three ways: It represents free space directly in workspace which makes it scale better with DOF. It integrates the obstacle avoidance with task-level constraints. Finally, it integrates some kind of replanning within the framework. Those three improvements are independent of each other.

The elastic strips approach does not represent free space in configuration space like elastic bands do but directly in workspace. This leads to a less conservative representation of free space and makes the algorithm scale better with the number of DOF. However, unlike with elastic bands, we need more than one bubble to cover one configuration. For each configuration the robot is bounded by a protective hull consisting of a number of spheres. The number of bounding spheres necessary depends on the distance of the robot to obstacles. The closer the robot is the more spheres are necessary. If the robot is far enough from obstacles it can be bounded by just one single sphere. Deciding whether the transition between two adjacent configurations is collision free is a little bit harder than with the approach of elastic bands. We need to check whether the volume swept by the robot while transitioning between two neighboring configurations is contained within the joint volume of the two protective hulls.

Elastic strips use the operational space formulation [15] for controlling the robot. The operational space formulation takes the dynamics of the robot into account and allows the integration of task-level constraints with obstacle avoidance. The forces that are resulting from the obstacle avoidance behavior are mapped such that they do not interfere with the task. In some situations it might not be possible to fulfill the task constraints and avoid obstacles at the same time. Therefore, elastic strips includes a mechanism to detect such a situation and to suspend the task constraint temporarily.

Elastic strips include a method to repair a path that has become very suboptimal, e. g. when an obstacles passes the path of the robot. This is done by letting the elastic strip tear apart, which allows the obstacle to pass through. Afterwards the elastic strip is repaired by the internal forces. Without this method an external path planner would have to be called once in a while to avoid a too suboptimal path. The method described here only tackles this very specific situation. There are still situations that can only be solved by a global task planner. Such situations include a change in the topology of free space, e. g. closing a door.

The elastic strips approach improves elastic bands in different ways, the representation of free space in workspace instead of configuration space being the most important improvement. As the improvements are independent of each other, it would be possible to use just one or two of them and ignore the others. The representation of free space in workspace could be useful for our scenario, especially if incorporating the movement of robot base into the planning process. Whether the additional complexity is worth the speed up in our case, would need to be investigated further. The integration of task-level constraints is not part of my work but elastic strips would make it possible to integrate them in the future. The consideration of the robot dynamics is neither needed nor wanted at the current stage of the Care-O-bot development. This does not hinder the use of elastic strips, as it could also be used without considering dynamics.

2.3 Decomposition-based Motion Planning

Published 2000-2001 by Oliver Brock and Lydia Kavraki.

Decomposition-based motion planning [11, 12] is based on the observation that free space can be easier represented in the lower dimensional cartesian workspace while the robot motion has to be executed in the higher dimensional configuration space. Therefore decomposition-based motion planning decomposes the problem of finding a path from a start to a goal configuration into two subproblems, one in workspace and one in configuration space. The two are connected through the volume in workspace that is swept by the robot following a path in configuration space. The first subproblem is to find a volume in workspace, called tunnel, that contains the volume swept by at least one solution path. The second subproblem is to find a solution path in configuration space using the workspace tunnel.

In [11, 12] one specific implementation of the framework for a simulated mobile manipulator is given. In this implementation, first, a wavefront expansion algorithm is used, which fills the free space with bubbles. It starts with one bubble at the goal configuration of the robot. New bubbles are created towards the direction of the start configuration. A new bubble is always created on the surface of an existing bubble, which becomes its parent. The bubbles form a tree. The algorithm terminates when the goal configuration is enclosed in a bubble. In a second step, the bubbles are used to create a navigation function. The navigation function is defined such that the gradient within one bubble is always directed toward the center of the parent bubble (or toward the goal in the root bubble). This navigation function and the nearby obstacles result in forces acting on the robot.

The framework described above does not provide a general solution, but leaves important problems to be solved by a particular implementation. This includes making sure that a tunnel is actually a superset of the volume swept by at least one solution path. Also, the given implementation of the framework leaves me with some doubts and questions. I think decomposition-based motion planning is an attempt to generalize the idea to represent free space directly in workspace, which was originally part of elastic strips. But more work needs to be done to make the framework actually useful.

2.4 PRM for Changing Environments (PRMCE)

Published 2000-2002 by Peter Leven and Seth Hutchinson (University of Illinois at Urbana-Champaign). This approach introduced by Leven and Hutchinson was not given a name by the authors. The name used here, originates from several presentations by Steven M. Lavalle et al. This is the algorithm I used for my implementation.

RPM for changing environments (PRMCE) [6] tries to achieve the real-time requirements by outsourcing a lot of work into a preprocessing stage. This preprocessing must only be done once for a particular robot and can take a long time, i.e. several hours or days. PRMCE directly builds on top of PRM, which was designed for static environments, and extends it to changing environments. PRM builds a roadmap once for the given environment and is then able to quickly calculate a path for any given start and goal configurations.

PRMCE accommodates for changing environments by generating a roadmap for an

obstacle-free environment and then invalidating those parts of the roadmap online that are blocked by dynamic obstacles. To determine which parts of the roadmap are blocked at any given time, the environment is discretized into a grid. For each configuration of the robot there is a set of grid cells that are occupied by the robot at this configuration. If one of these cells is occupied by an obstacle, this obstacle is potentially in collision with the robot. Therefore, during the preprocessing stage a mapping from grid cells to blocked nodes and edges of the roadmap is calculated. This mapping is stored with each grid cell and allows for fast invalidation of blocked nodes and edges. The roadmap and workspace mapping are generated once during the preprocessing phase and then stored for further use.

The roadmap generation of PRMCE is similar to PRM [16, 17]. A normal PRM algorithm is able to guide its roadmap generation by the location of obstacles. It could create more samples in regions that are harder to navigate in, e. g. close to obstacles and in regions that are very narrow. This is not possible in the case of PRMCE as it does not know where obstacles are. While PRMCE cannot use information on the location of obstacles, it can use information that is specific to the robot. Leven and Hutchinson apply two different sampling methods: uniform sampling and manipulability-based sampling. Manipulability is a measure of the ability of the robot to move and position its end-effector. In areas where the manipulability is low it is harder to find a path. Taking more samples in those areas increases the probability of finding a path.

To generate edges, for each node a set of neighboring candidate nodes is selected, which are tried to connect to. This set consists of the k nearest neighbors of the current node according to some metric. The path to each candidate node is then checked for collision. If it is collision-free, an edge between the current node and the candidate node is created.

After the roadmap has been generated, Leven and Hutchinson take additional steps to optimize it. The objective is to decrease the likelihood of the roadmap to become disconnected in the presence of obstacles. Amongst other things, additional connections are created in poorly connected areas.

The roadmap and workspace mapping need a lot of memory space (several hundred megabytes in my implementation). In order to be able to plan in real-time, the whole roadmap and workspace mapping need to be kept in physical memory and must not be swapped out onto disk. With current computers having main memory sizes of several gigabytes, this is becoming less of a problem. But still, the memory consumption may be the limiting factor to the size of the roadmap and the grid. For this reason, Leven and Hutchinson put a lot of effort in compressing the roadmap and workspace mapping. This actually takes up 40% of their paper.

2.5 Flexible Anytime Dynamic PRM (FADPRM)

Flexible Anytime Dynamic PRM (FADPRM) [13] tries to achieve the real-time requirements by, first, reusing the previously calculated path, only repairing it where necessary, and, second, by calculating a highly suboptimal solution very quickly and then improving it as long as there is time available. The first property makes the algorithm dynamic and the second one makes it anytime.

FADPRM builds on top of Anytime Dynamic A* (AD*) [18], which is a graph search

algorithm. But here we do not have a graph that we want to search but a continuous configuration space. Thus, FADPRM combines AD* with probabilistic roadmaps (PRM) [17] to do an anytime and dynamic search within the configuration space. AD* in turn combines two previously independent approaches: dynamic and anytime graph search. It is based on the two algorithms Anytime Repairing A* (ARA*) [19] and D* Lite [20]. I will describe all these algorithms shortly in the following paragraphs.

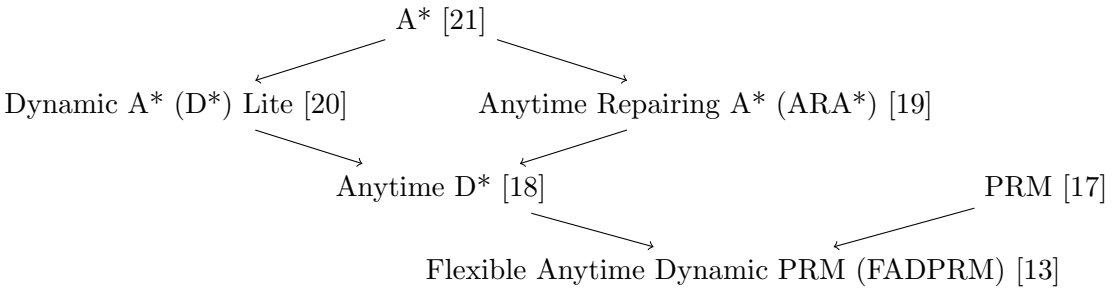


Figure 2.1: How FADPRM is based on previous work

D* Lite is a simpler implementation of D*. D* (Lite) recomputes the shortest path in a graph after some edge costs changed without recomputing the whole path from scratch. This leads to a more efficient search. D* builds a search tree with the goal node as its root. If an edge cost changes only the subtree below that modified edge becomes invalid. The rest of the tree can be reused as is. As D* is mainly used for mobile robots that can only sense the environment in their immediate vicinity, edge costs will only change close to the current location of the robot which is close to leaf nodes of the search tree. Therefore not many nodes are being invalidated.

Anytime Repairing A* (ARA*) generates a suboptimal solution very quickly by inflating the heuristic function, i. e. multiplying all heuristic values by a fixed factor. The larger the factor, the faster the search but the less optimal the path will be. With increasing inflation factor the search becomes more similar to a best-first search. ARA* searches for a path iteratively, starting with a high inflation factor and then decreasing it over time.

Anytime Dynamic A* (AD*) combines the two previously described algorithms. The main problem here is how to choose the inflation factor after some edge costs were updated. With a low inflation factor it may take a along time to repair the changes. A high inflation factor may make the path less optimal than it was before the change in edge costs.

FADPRM does graph search like AD*. But, whenever it expands a node, it samples a new node in the neighborhood of the expanded node. The choice of the next node to expand depends on, first, the density of nodes around a given node (as in PRM) and, second, on the minimum expected cost of a path through the given node (as in A*). The two criteria are weighted based on the inflation factor. For a high inflation factor (yielding a fast, highly suboptimal solution) the second criteria prevails, for a low inflation factor the first one dominates. I. e. with increasing inflation factor we are moving from an A*-like toward a PRM-like expanding scheme. FADPRM also incorporates desired and undesired areas in workspace but this is of no interest to us.

There are several points that make me skeptical about FADPRM. First, D* does well when changes in the search graph are close to the current location of the robot

as is the case for mobile robots. But for a robot arm and a sensor that observes the workspace independent of the current configuration of the arm, changes to the graph could be everywhere. A change close to the goal configuration renders the whole previously computed search tree useless.

Second, FADPRM only deals with optimizing graph search, which is just one part of the whole planning process. FADPRM does not deal with the problem of efficiently updating the edge costs.

Chapter 3

Design

3.1 Algorithm Selection

As Decomposition-based Motion Planning and FADPRM did not seem mature enough to me to be useful, I was left with two different options: Elastic Bands/Strips and PRMCE. They both seem to be good approaches to our problem. Elastic bands could optionally use the free space representation of elastic strips. The consideration of robot dynamics as in elastic strips is not necessary for us right now. I finally chose PRMCE. However, it was a close decision. One reason for choosing PRMCE over elastic bands/strips was my feeling that PRMCE was easier to implement. The second reason for PRMCE was the fact that it fits better within the current implementation of the Care-O-bot and allows to build on work that has been done at Fraunhofer IPA previously.

The currently implemented collision checker, which checks for self-collision and collision with the static environment model, does not calculate distance information but only whether a given configuration is in collision or not. For elastic bands/strips the collision checker would have to be replaced by one that calculates the distance. PRMCE can build on top of the existing collision checker. In addition, a 3D grid representation of the environment has been used previously at Fraunhofer IPA in work both related and unrelated to the Care-O-bot.

3.2 Scope of the Planner

My planner plans a motion for the arm only. The robot base is assumed to be static while the arm is moving. This implies that the workspace grid is fixed on the robot base, covering the workspace of the arm around the robot. The grid covers a cube with an edge length of approximately 2.5 meters, the robot base being at its center.

Another option would have been to also include the robot base in the planning process, which would have added 3 DOF to the 7 DOF of the arm and would have made the workspace grid fixed to the world coordinate frame, covering the whole space the robot is supposed to act within. I did not include the robot base in the planning process for two reasons. First, I do not know how well the algorithm scales to 10 DOF and I think it is a good idea to start with a simpler case that is more likely to work. And, second, the current architecture of the Care-O-bot keeps the control of the arm and the base separated. This would make a architectural change necessary to integrate planning for the arm and the

robot base, which I did not want to get myself into. Extending my planner to include the robot base is left for future work. See section 7.5.

3.3 Determining Blocked Workspace Grid Cells

The PRMCE algorithm includes the workspace mapping and the method on how to plan fast once we know which parts of the discretized workspace are blocked by obstacles. How exactly we determine which workspace grid cells are blocked is left open by PRMCE. This is the task of a specific implementation.

I use the current SwissRanger SR3000 sensor data to determine blocked workspace grid cells. The most important word here is "current". I only consider the most recent sensor reading. The sensor returns a point cloud relative to the coordinate frame of the sensor. Thus, determining blocked workspace grid cells only includes transforming the point cloud into the robot base coordinate frame and discretizing each point into a grid cell.

The problem here is that the robot arm might also be part of the point cloud. Points that represent the arm must be removed and must not block a workspace grid cell. Just one point that is part of the robot arm but is mistaken as an obstacle can invalidate all or most roadmap configurations or path segments around the current configurations. This would make the arm to not move at all. As there are several sensor-specific effects that

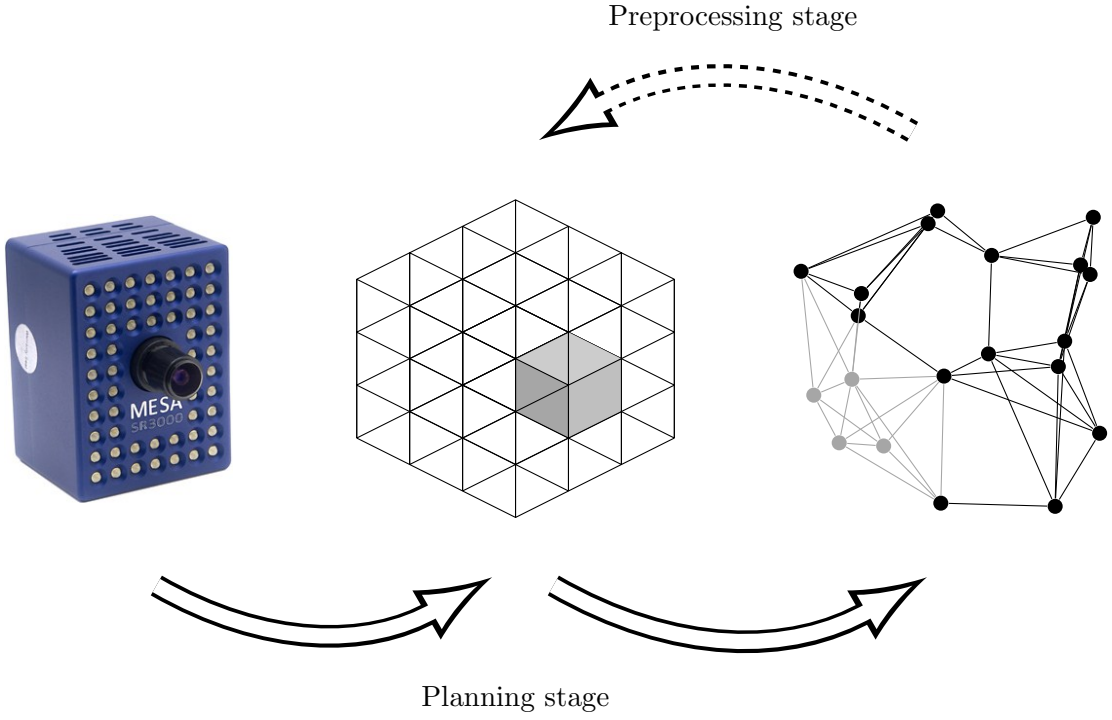


Figure 3.1: Design schematic. The right part reflects the workspace mapping as introduced by PRMCE. The left part shows the integration of the SwissRanger sensor. The model, which is also used to determine blocked grid cells, is not shown.

make the SwissRanger return sensor data that not totally resembles reality, it is hard to completely remove the robot arm out of the sensor data although we know its location. As processing sensor data is not the core of my work, I did not concern myself with that problem. I only tried to remove the robot arm using the most basic approach of just removing points that are located where the robot arm is supposed to be. I only tried this out because of curiosity not because I expected it to work. And it did not work. Thus, my planner is currently only able to plan as long as the arm is not within the field of view of the sensor.

As the field of view of the sensor is very limited, we cannot expect the robot to recognize and avoid all unknown obstacles. However, we want the robot to avoid at least all known obstacles as the old planner already does. Therefore I also included the hand-crafted model into the process of determining blocked workspace cells. To include the model into the planning process we determine which workspace grid cells are blocked by modeled obstacles in addition to those blocked by sensed obstacles. See section 5.4.1 for how this is done.

3.4 Online vs. Offline Planning

My planner currently plans offline, i. e. before the arm starts moving. However, this does not impair the fact that my whole work aims at enabling online planning through a fast enough path planner. There are two reasons why I did not implement my planner to plan online. First, as the robot arm currently cannot be removed from the sensor data, we cannot plan online anyway because it is very likely that the arm moves through the field of view of the sensor. However, we are able to plan offline for certain configurations if the arm is not inside the field of view of the sensor while it is in the start configuration. Second, planning online would require some architectural change as the planner and the execution of the trajectory need to run in parallel and they currently do not.

Chapter 4

Configuration Space Metrics

Having a n-dimensional configuration space $\mathcal{C}$, we are looking for a metric that defines a distance between any two configuration $p, q \in \mathcal{C}$.

In the 3-dimensional euclidean space there is an obvious way to calculate the distance between two points: The euclidean distance. This is not as straightforward in high dimensional configuration spaces and there is not one obvious way to do it. Instead we need to ask ourselves some questions: What do we want to measure? What should the distance express? When do we consider two configurations close? What should our path planner minimize? Although we are looking for a metric for the configuration space, what we want to measure is most often a property of the movement of the robot in workspace. Below are some possible properties of the robot movement that a metric could represent.

- **Swept volume** by the robot while moving from p to q. [6] A small swept volume means a small likelihood that the path between the two configurations is blocked by an obstacle, i.e. a local path planner is likely to be able to connect the two configurations.
- **Maximum displacement** of some point on the robot. [7] Taken the locations in workspace of any point on the robot at two configurations on the path segment between p and q, this is the maximum distance between those two locations. In most cases the maximum displacement occurs for the two end configuration p and q of the path segment. This property is of interest if we know that the robot is a certain distance away from obstacles and want to assure that the robot stays clear of the obstacles while moving along the path segment.
- **Maximum distance traveled** by some point on the robot while moving from p to q. This property is an upper bound on the maximum displacement.
- **Time** required to move from p to q. If we are mainly interested in generating a path that is time-optimal, we need to use a metric that represents the time needed.

Even if we have decided which property or properties we want our metric to represent, we do not have a usable metric, yet. The problem is that some of the properties above might not be easily calculated (easy referring to execution and implementation time). As path planning involves a lot of distance calculation, we the metric needs to be calculated fast. Therefore we might use metrics that do not exactly represent one or more of the

properties above, but instead one that approximates the property or represents an upper or lower bound on the property.

4.1 Euclidean Distance

$$d_2(p,q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

The euclidean metric (L_2) is probably the most basic configuration space metric you could come up with. The advantage of this metric is that no knowledge of the shape of the robot is needed. In return you do not get much information out of it either.

This metric is used by the current SBL path planner for the Care-O-bot's arm.

4.2 Weighted L_1 Distance

$$d_{w1}(p,q) = \sum_{i=1}^{n} w_i |p_i - q_i|$$

If the right weights are chosen, this metric is an upper bound on the maximum distance traveled while moving between two configurations. To achieve this, the weight w_i must equal the radius of a cylinder centered on the axis of joint i such that all subsequent links always stay within this cylinder.

This metric is used in this work as part of the combined metric described in section 4.9.

4.3 Weighted L_∞ Distance

$$d_{w2}(p,q) = \max_{i=1}^{n} w_i |p_i - q_i|$$

If the right weights are chosen and some assumptions hold, this metric equals the time the robot needs to move from one configuration to the other. The assumption here is that all joints move simultaneously, each at its own fixed speed. This is not totally accurate as the motors need some time to accelerate and decelerate.

4.4 Maximum Distance Traveled

$$d_r(p,q) = \min \left(\sum_{i=1}^{n} r_i(p_{i+1}, ..., p_n) \cdot |p_i - q_i| \ , \ \sum_{i=1}^{n} r_i(q_{i+1}, ..., q_n) \cdot |p_i - q_i| \right)$$

This metric is an upper bound on the maximum distance a point on the robot travels while moving between two configurations. The difference to the weighted L_1 metric described above is the fact that the radii depend on the configuration of the subsequent joints. This metric is introduced and used by the Elastic Bands approach [1, 2] described in section 2.1.

For a given configuration p, the radius r_i is the distance from the axis of joint i to the farthest point of any subsequent link. This can be used to calculate the longest distance d_i a point on the robot travels while joint i is moved from position p_i to q_i.

$$d_i = r_i(p_{i+1}, ..., p_n) \cdot |p_i - q_i|$$

In order to get a result for moving the whole robot from configuration p to configuration q, we imagine moving each joint individually while holding the others fixed starting at the innermost joint. The total distance traveled by a point on the robot cannot be greater than the sum of the distances traveled while moving a single joint. The sum of all d_i is therefore a upper bound on the longest distance traveled by a point on the robot while moving from p to q.

$$d_{max} \leq \sum_{i=1}^{n} d_i = \sum_{i=1}^{n} r_i(p_{i+1}, ..., p_n) \cdot |p_i - q_i| = \bar{d}_r(p, q)$$

As $\bar{d}_r(p, q) \neq \bar{d}_r(q, p)$, we cannot use it as metric directly. In order to achieve symmetry, we have to define our metric $d_r(p, q) = \min\left(\bar{d}_r(p, q), \bar{d}_r(q, p)\right)$. This way we also get a lower upper bound in half the cases.

4.5 2-Norm in Workspace

$$d_2^{\mathcal{W}}(p, q) = \sqrt{\sum_{a \in \mathcal{A}} \|a(p) - a(q)\|^2}$$

This metric is guided by the wish of having a metric that represents swept volume. [6] It was introduced in a simpler form by Kavraki and Latombe [16], which was used in their PRM planner [17].

The metric defines a finite set of reference points $\mathcal{A}$ on the robot. $a(p) \in \mathcal{W}$ is the position of $a \in \mathcal{A}$ in the workspace at a configuration p. For each reference point the distance between the two positions of the point in workspace for the two configurations p and q is calculated. The distance between p and q is the 2-norm of the vector of those distances for all reference points.

For a specific configuration the positions of the reference points in workspace are calculated using the kinematics of the robot arm. After mapping the reference points into the workspace all further calculations are done in the workspace. This metric gives a good measure of how much the robot moves as it does not overestimate the movement as much as for example the maximum distance traveled metric described above in section 4.4.

The more accurate representation of the robot movement comes at the expense of computation time. The calculation of the forward kinematics for all reference points, takes a long time. However, if distances are calculated within a fixed set of configurations repeatedly, the locations of the reference points only need to be calculated once for each configuration. This speeds up the metric substantially and is done in my implementation. Only precalculating the reference points makes this metric useful during the online planning phase. See also section 5.4.3.

This metric is not necessarily positive definite. Depending on the choice of the reference points and the range of the joints, it may only be positive semidefinite. The simplest

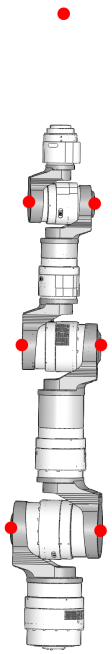


Figure 4.1: Model of the robot arm with reference points (shown in red) for the workspace metrics. The hand is not shown. The reference point floating above the arm is actually at the tip of the hand.

example for two different configurations being a distance of zero apart is a single link attached to a rotational joint at 0 and 360 degrees. Therefore, this metric is not really a metric but a pseudometric.

Although in practice it is relatively unlikely that two distinct random configurations are a distance zero apart, this metric still suffers from the problem that two completely different configurations, may be considered close (but distinct).

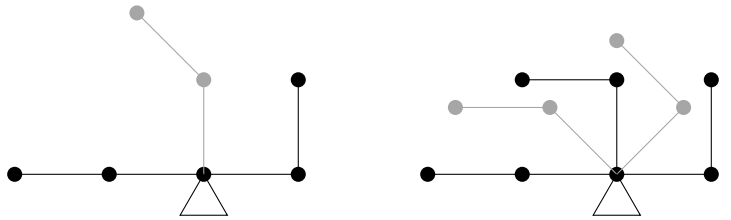
This metric is used in this work as part of the combined metric described in section 4.9.

4.6 2-Norm in Workspace including Midpoint

$$d_{2m}^{\mathcal{W}}(p,q) = \sqrt{\sum_{a\in\mathcal{A}} \|a(p) - a(m)\|^2 + \sum_{a\in\mathcal{A}} \|a(m) - a(q)\|^2} \quad , \quad m = \frac{p+q}{2}$$

This metric counters the problem of mistaking two configurations as close. The reason why the metric described in section 4.5 is not a good representation of the swept volume while transitioning between two configurations in some cases, is the fact hat it only takes the two end configurations into account. The metric described in this section tries to avoid this problem by including the midpoint configuration between the two end configurations in the distance computation.

This metric does not satisfy the triangle inequality. See figure 4.2 for a counter example. Therefore it is only a pseudosemimetric.



(a) Straight-line path between the two end configurations.
Path length: 3.46

(b) Path through an intermediate configuration.
Path length: 1.87 + 1.53 = 3.40

Figure 4.2: An example of a robot with two joints acting in a plane for which the 2-norm in workspace including midpoint metric does not satisfy the triangle inequality. The length of the links is 1. The midpoints used by the metric are shown in gray.

This metric is computationally more expensive than the one without the midpoint because the forward kinematics for the midpoint need to be calculated as well. Unfortunately, the reference points of the midpoint configuration cannot be precalculated efficiently, as the midpoint depends on both configurations p and q.

This is the main metric used in this work. However, its use is mostly avoided during the online planning phase because it takes too much time to compute.

4.7 Admissible 2-Norm in Workspace

$$d_{2a}^{\mathcal{W}}(p,q) = \sqrt{\frac{1}{2}\sum_{a\in\mathcal{A}} \|a(p) - a(q)\|^2}$$

As stated above the workspace metric including the midpoint is more expensive to compute than the one without the midpoint. In addition, the midpoint metric does not satisfy the triangle inequality. For computing the heuristic of an A* search we need a fast metric that is a lower bound on the midpoint metric and satisfies the triangle inequality. The 2-norm in workspace metric described in section 4.5 is not a lower bound on the midpoint metric. Therefore, this metric is introduced. It equals the 2-norm in workspace metric multiplied by $\sqrt{\frac{1}{2}}$.

The following inequality proves the fact that this metric is a lower bound on the midpoint metric:

$$
\begin{aligned}
d_{2m}^{\mathcal{W}}(p,q) &= \sqrt{\sum_{a\in\mathcal{A}} \|a(p)-a(m)\|^2 + \sum_{a\in\mathcal{A}} \|a(m)-a(q)\|^2} \\
&\geq \sqrt{\sum_{a\in\mathcal{A}} \left\|a(p)-\frac{a(p)+a(q)}{2}\right\|^2 + \sum_{a\in\mathcal{A}} \left\|\frac{a(p)+a(q)}{2}-a(q)\right\|^2} \\
&= \sqrt{2\sum_{a\in\mathcal{A}} (\frac{1}{2}\|a(p)-a(q)\|)^2} \\
&= \sqrt{\frac{1}{2}\sum_{a\in\mathcal{A}} \|a(p)-a(q)\|^2} \\
&= d_{2a}^{\mathcal{W}}(p,q)
\end{aligned}
$$

4.8 ∞-Norm in Workspace

$$d_{\infty}^{\mathcal{W}}(p,q) = \max_{a\in\mathcal{A}} \|a(p)-a(q)\|$$

This metric is similar to the 2-Norm in Workspace metric described above in section 4.5. Instead of taking the 2-norm of the distance vector, we take the ∞-norm, which gives us the largest displacement of any of the reference points. This is not necessarily an upper bound on the maximum displacement of any point, however, depending on the choice of the reference points this metric can be a good approximation thereof. For a robot that only consists of polyhedra, this metric can represent the maximum displacement exactly if the right reference points are chosen. [7] The general advantages and drawbacks of the 2-norm workspace metric apply here as well.

This metric is used in my implementation for determining the ϵ distance between two configurations that are checked for collision along a path segment. See section 5.3.1.

4.9 Combined Metric

$$d_{combined}(p,q) = 0.9 \cdot d_2^{\mathcal{W}}(p,q) + 0.1 \cdot d_1(p,q)$$

This metric is a combination of two previously described metrics: the weighted L_1 distance described in section 4.2 and the 2-norm in workspace described in section 4.5. The weights for the L_1 distance are chosen such that it is an upper bound on the maximum distance traveled by any point on the robot. The goal of this combined metric is not to mistake two far-apart configurations as close as the 2-norm in workspace might do, but without the computationally expensive inclusion of the midpoint configuration. This is traded off against a less accurate representation of the robot movement.

For two far-apart configurations the weighted L_1 distance will be large. Thus, taking the weighted L_1 distance into account avoids such configurations from being considered close. However, in all other cases, the influence of the weighted L_1 distance must be kept small as it is less accurate than the 2-norm in workspace metric. Therefore, the weighted L_1 distance only contributes 10% to the combined metric, while the other 90% are based on the 2-norm in workspace.

Another advantage of this combined metric over the midpoint metric is the fact that it satisfies the triangle inequality.

This metric is used in this work during the online planning stage to find the k nearest neighbors of the start and goal configurations.

Chapter 5

Implementation

5.1 Overview

This chapter assumes that the used PRMCE algorithm is known. For a high-level description of PRMCE, see section 2.4.

The following enumeration shows all tasks which need to be solved for implementing the PRMCE planner. The individual steps are then explained in detail in later sections. Like the PRMCE algorithm itself this chapter consists of two major parts: the preprocessing stage, which is executed once, and the planning stage, which is executed repeatedly, each execution being called a planning round.

1. Preprocessing stage
   (a) Generate roadmap
      - Generate nodes (section 5.3.1)
        - Sample configurations
        - Check configurations for collision
      - Generate edges (section 5.3.1)
        - Search for k nearest neighbors
        - Test path segments for collision
   (b) Enhance roadmap (not implemented)
   (c) Generate workspace mapping (section 5.3.2)
      - Voxelize configurations
      - Voxelize path segments
   (d) Compress roadmap and mapping (not implemented)
   (e) Store and load roadmap and mapping (section 5.3.3)

2. Online planning stage
   (a) Identify occupied workspace cells (section 5.4.1)
      - modeled obstacles
      - sensed obstacles
   (b) Remove blocked nodes and edges from roadmap (section 5.4.1)
   (c) Connect start and goal configurations to roadmap (section 5.4.2)
      - Search for k nearest neighbors
      - Test path segments for collision
   (d) Search roadmap (section 5.4.3)

5.2 Data Structures

The roadmap is a linked data structure. Each node and edge is an object and keeps information on whether it is currently blocked. Each node keeps a list of neighbors and stores for each neighbor a pointer to the neighboring node and a pointer to the edge to the neighboring node. The edge itself does not store any pointers or information other than its length and whether it is blocked or not.

The workspace grid is an array of grid cells. Each grid cell stores a list of pointers to roadmap nodes and edges that are blocked by this grid cell.

5.3 Preprocessing Stage

5.3.1 Roadmap Generation

Generating Nodes

While Leven and Hutchinson [6] implement and evaluate two different sampling strategies, I only implemented the simplest one: uniform sampling. Configurations are sampled in the 7-dimensional joint space. On the Care-O-bot arm 4 out of the 7 joints have a range of 720 degrees. For sampling I reduced those joint ranges to 360 degrees. This leads to a smaller or denser roadmap and to a better performing workspace metric. With a 720 degrees joint range there are more possibilities for two configurations to be close in workspace but far apart in configuration space.

Generating Edges

The strategy used in this work to generate edges is the same as Leven and Hutchinson used. We try to connect each node to its k nearest neighbors. Two nodes n_1 and n_2 can become connected by either n_1 being one of n_2's k nearest neighbors, or by n_2 being one of n_1's k nearest neighbors. Each path segment to one of the k nearest neighbors is checked for collision. If the path segment is collision-free an edge is added to the roadmap. The metric used to determine the k nearest neighbors is the 2-norm in workspace metric including the midpoint described in section 4.6. The k nearest neighbors for all nodes are retrieved through a brute force comparison of every node with all other nodes. This leads to a quadratic time complexity. The reason for using the brute force method instead

of faster methods like a cover tree is the fact that the metric used does not satisfy the triangle inequality. A cover tree assumes a metric that satisfies the triangle inequality and does not return the correct result if the metric does not. It only returns an almost correct result. For example, for $k = 20$, it returns most of the time 20 out of the 23 nearest neighbors. During the preprocessing stage we can afford a longer running time, therefore the slower but correct brute force method is preferred here.

Path segments are checked for collision by checking discrete configurations on them. As the Care-O-bot's collision checker does not return a distance to the closest obstacle but only returns whether a particular configuration is in collision or not, we cannot decide exactly whether a path segment is in collision. However, we can increase the probability of detecting an existing collision arbitrarily by choosing a small enough distance ϵ between two neighboring checked configurations. The value of ϵ greatly affects the running time of the planner. A large ϵ will increase the probability of wrongly marking a path segment collision-free. A small ϵ will increase the number of collision checks necessary and therefore the running time of the planner. The actual meaning of ϵ depends on the metric used for measuring the distance between two neighboring checked configurations. In order to be able to trade off safety versus running time, it is important to use a metric that is meaningful. The path planner that was used on the Care-O-bot previously used the L_2 metric. This only allows for determining ϵ empirical. In my implementation I use the ∞-norm in workspace as metric for checking path segments (see section 4.8). This gives ϵ the meaning of the maximum displacement of any point on the robot between two collision checks. As I chose fewer reference points for my implementation of the metric as necessary for exactly measuring the maximum displacement, ϵ only represents a approximation of the maximum displacement. Unlike other metrics, which are an upper bound on the maximum displacement, this metric measures the maximum displacement exactly, which reduces the number of collision checks necessary to guarantee a displacement between to checked configurations below a certain threshold. However, this metric also needs more time to compute. But as there is a lot more time needed for a collision check, it is most important to reduce the number of collision checks as far as possible.

5.3.2 Workspace Mapping

In order to be able to quickly invalidate parts of the roadmap that are currently blocked by obstacles, we create a mapping between the workspace and the roadmap. The workspace is discretized into a grid. This grid is centered on and moving with the robot and covering the whole reachable space of the arm. The grid consists of non-overlapping cubical cells. The workspace mapping maps each grid cell onto a set of nodes and edges of the roadmap that collide with this grid cell. This mapping is stored with each grid cell in the form of two lists of pointers to roadmap nodes and edges. Given that we know which grid cells are occupied, this allows for fast invalidation of the blocked parts of the roadmap.

This discretization leads to a loss of accuracy. Parts of the roadmap may get invalidated although they are not blocked. We accept this in order to achieve a fast planning process. The resolution of the grid can be specified such that a trade of between accuracy and speed can be made as required.

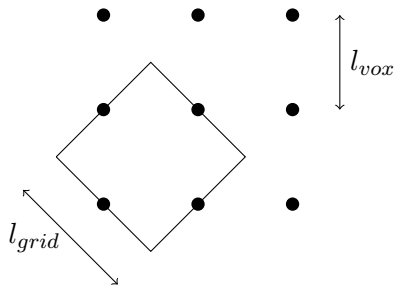


Figure 5.1: Sampling resolution necessary in the 2D case to guarantee that all workspace grid cells within the volume of the robot are hit by at least one sample point.

Voxelizing a Configuration

Leven and Hutchins use a voxelization algorithm similar to one by Kaufman and Shimony [22]. It directly computes a set of grid cells that represents a polyhedron. I decided not to implement this algorithm but a simpler and potentially slower and less accurate one. My algorithm samples points within the volume of the robot arm and on its surface and then maps those points into the grid. Now we only need to map points to grid cells, which is rather simple. This is done by a coordinate transformation from the local coordinate system of the link the point is part of to the robot base coordinate system. This transformation depends on the current configuration of the robot. The transformed point is then discretized into a grid cell.

The most difficult but still rather simple part of my voxelization method is the generation of the sample points. These points are generated only once during initialization. The sample points are arranged within the volume of the robot arm in a grid pattern, the sample points being located at the vertices of a cubic grid. The distance between two sample points l_{vox} depends on the edge length of the workspace grid l_{grid}. If the distance between two neighboring sample points is below a certain threshold we can guarantee that all workspace grid cells within the volume of the robot arm are hit by at least one sample point. For the 2D case shown in figure 5.1, it is easy to see that this can be guaranteed if

$$l_{vox} \leq \frac{l_{grid}}{\sqrt{2}}$$

For the 3D case this is less obvious. I have not proven any threshold for the 3D case. But after trying to figure out different ways of putting a cube within a 3D grid of points, I could only come up with two ways, which both lead to the same threshold as for the 2D case. Therefore, my implementation uses $l_{vox} = \frac{l_{grid}}{\sqrt{2}}$.

Although we can be sure of no workspace grid cells being missed inside the robot volume, this is not ensured on its boundaries. A workspace grid cell could extent partially into the robot volume but not be hit by a sample point. To counter this, additional points are sampled on the surface of the robot. This does still not guarantee that all grid cells that overlap with the robot are hit, but the probability can be increased arbitrarily by decreasing the distance between samples on the surface. Therefore, the distance between neighboring samples on the surface can be configured.

It is possible to guarantee that all (partially) occupied grid cells are hit by samples through the combination of a dense enough sampling on the robot surface and an artificial

increase of the size of the robot. This is currently not implemented.

Voxelizing a Path Segment

For voxelizing path segments I use the same approach as Leven and Hutchinson. It is somehow similar to testing path segments for collision. The midpoint of the path segment is voxelized. The path segment is then recursively subdivided in two parts and the process repeated until no new occupied cells are added anymore. The cells that are occupied by the two endpoint configurations of the path segment are not considered for the set of cells occupied by the path segment. This means for a path segment we are only interested in the cells that are swept by the robot while moving between two configurations but not while being in one of the two configurations. We are not interested in cells occupied by the endpoints because if such a cell is occupied by an obstacle, the roadmap node corresponding to the endpoint configuration will be blocked and the roadmap edge corresponding to the path segment will not be of any use anyway.

5.3.3 Storing and Loading Preprocessed Data

As the preprocessing stage takes several hours, we do want to execute it every time the program is started. Thus, we need to be able to store and load the created roadmap and mapping. This is done using the Boost Serialization library. This library provides methods to serialize a linked data structure and to store it into a file, from where it can later be loaded and deserialized. The Boost Serialization library serializes an object by serializing all its fields, recursively following pointers to other objects. It detects an object that has already been serialized and, thus, can handle circular pointer structures like our roadmap.

5.4 Planning Stage

5.4.1 Invalidating Blocked Parts of the Roadmap

Whenever we want to execute a new planning round, we first need to determine the currently blocked parts of the roadmap. Obstacles are detected using the SwissRanger SR3000 distance sensor. This sensor returns a point cloud in the coordinate frame of the sensor. Each point is transformed into the robot base coordinate frame and then discretized. If it hits a grid cell, all roadmap nodes and edges that are in collision with the grid cell are invalidated for the current planning round. This invalidation is pretty fast as only two lists of pointers to nodes and edges are iterated through and the containing nodes and edges are marked by setting a variable. After a grid cell has been hit once, it is marked as blocked such that future hits do not waste time by trying to invalidate the already invalidated parts of the roadmap again.

The main problem here is that the point cloud we are getting from the SwissRanger sensor does not only consist of obstacles but also of the robot itself. Thus, we need to remove those parts of the point cloud that represent the robot. This is not an easy task and I have not solved it. The problem is that just one point that does not belong to an obstacle and is not removed from the sensor data will probably make the arm not move at all. A point that is not removed from the point cloud and belongs to the arm will lead

to a blocked grid cell close to the arm. This means configurations close to the current one will be invalidated and the possible movement of the arm constrained, possibly resulting in the path planner failing.

Removing the arm exactly from the sensor data is hard because the sensor data is not very accurate. I used the existing workspace grid to remove points belonging to the arm from the senor data. All grid cells that are currently occupied by the robot are marked and later not blocked even if they are hit by a sensor point. Unfortunately, this method only removes parts of the robot arm from the sensor data (see figure 6.14). This is probably because of inaccuracies of the sensor. To counter this problem I removed more points from the sensor data possibly also removing obstacle points. This comes with the risk that obstacles are not being avoided although they have been detected by the sensor. This seems to be acceptable at the current stage of development, especially as the area in which the sensor can detect obstacles is very limit anyway. To remove more points from the sensor data, I relax the condition for a point being removed in three different ways. First, I grow the model, which is used for determining the occupied grid cells, by a fixed amount in all directions. Second, I remove neighboring pixels of removed pixels, also. And third, the distance of a point to the camera is varied and if any of the variations hits a cell occupied by the robot, the point is removed. Although, after implementing those relaxations more parts of the arm are removed, there are still a few points that are not removed.

To make sure that at least all static obstacles are detected in spite of the limited range of the sensor, I also incorporated a model-based obstacle detection. To integrate this into the rest of the planning process, the surface of known static obstacles is sampled during initialization and the points stored in a list. During the planning stage the points are transformed into the robot base coordinate frame and discretized into a workspace grid cell, which is then blocked in the same way as for points retrieved from the sensor. The coordinate transformation of the points depends on the current location and orientation of the robot base.

5.4.2 Connecting Start and Goal Configurations to the Roadmap

In order to be able to plan a path from the start to the goal configuration, we need to connect those two configurations to the roadmap. The problem is that we now need to do collision checks during the online planning stage. As collision checks are computationally expensive, we need to avoid them as much as possible during the planning stage.

Leven and Hutchinson [6] connect the start and goal configurations in the same way as the original PRM method does. They consider roadmap nodes for connection to the start/goal configuration in order of increasing distance from the start/goal configuration. When they found one collision-free connection, they stop testing. I. e. they only create one connection, the shortest one that is collision-free. Creating only one connection saves time but leads to a longer and less smooth path.

My implementation differs from the one by Leven and Hutchinson [6]. It connects the start and goal configuration in a similar way as it creates edges during the preprocessing stage. All k nearest neighbors of the start or goal configuration are candidates for connection. Currently, the k is the same k as the one used for creating the roadmap, which I set to 20 for my tests. In the future it may make sense to choose the k for connecting the start and goal configurations lower than the k for creating the roadmap because time

during the planning stage is more valuable than during the preprocessing stage. So, for both, start and goal configuration, there are k path segments to test for collision. But instead of testing all possible connections for collision before the graph search and adding them to the graph only if they are collision-free, my implementation delays the collision checking. Edges are created for all k nearest neighbors. These initially untested edges are tested for collision during the graph search when they become part of the closed set (i.e. when the second node of the edge is expanded). This delayed collision checking was inspired by SBL [5]. Unlike with SBL the collision checking is not delayed until a complete path has been found but only until the edge is expanded. This avoids the need to recompute the whole path when a collision is detected.

To find the k nearest neighbors, I use a cover tree data structure [23]. A cover tree is a data structure that allows fast nearest neighbor queries. Unlike a kd-tree [24] it can be used with any metric. Unlike with a kd-tree, time complexity does not depend on the number of dimensions used to represent the data but only on the doubling constant of the data, which is the inherent dimensionality of the data. I did not implement the cover tree data structure and search myself but used an available implementation. Although the cover tree data structure limits the number of distance calculations necessary to find the k nearest neighbors, using the 2-norm in workspace including midpoint metric is too time consuming. Thus, I use the combined metric described in section 4.9.

This combined metric is only used to find the k nearest neighbors. Which of the k nearest neighbors of the start or goal configuration is actually connected to is decided during the graph search using the midpoint workspace metric. Using this combined metric instead of the midpoint workspace metric also avoids problems due to the midpoint workspace metric not satisfying the triangle inequality.

5.4.3 Graph Search

After currently blocked parts of the roadmap have been invalidated and the start and goal configurations have been connected to the roadmap, we can do the actual search for a path. A* [21] is used to search the graph. The heuristic used is the distance from the current node to the goal using the admissible 2-norm in workspace metric described in section 4.7.

Calculating the location of the reference points takes a lot of time. It actually takes so much time that using A* would not make any sense with this metric. With my initial implementation, A* took much longer than a breadth-first search, which expanded almost all nodes. The time A* saved by expanding fewer nodes was more than compensated by the time necessary to compute the heuristics. To speed up the metric and thus A*, I calculate all the locations of the reference points in advance during the preprocessing phase and store them with the nodes. Also, when a new goal configuration is set, its reference points are calculated. This allows for a fast computation of the heuristics.

5.4.4 Smoothing

The path found by the graph search can optionally be smoothed afterwards. Smoothing makes only sense for offline planning as smoothing the path requires a lot of collision checking. Avoiding collision checking is what this whole planning approach is about. Thus, I normally do not use smoothing in my tests but my implementation gives the user

the option to do so. Smoothing is done by connecting configuration on the path directly and leaving out all configurations in between. This requires as many path segments to be tested for collision as configurations are part of the path excluding the start and goal configurations.

5.4.5 Resetting Status Variables

There are several status variables that have to be kept for each node and edge during a planning round and have to be reset to a default value at the beginning of each planning round. These status variables are:

- whether a node or edge has been blocked by a dynamic obstacle (default: no)
- whether a node has been expanded (default: no)
- path cost to a node (default: infinity)
- whether a node is connected to the goal configuration (default: no)

Most of the status variables are boolean, defaulting to false. Iterating through all nodes and edges at the beginning of each planning round to reset the variables would require too much time. Thus, I use a different method. I introduce a round counter. At the beginning of each round this counter is increased by one. Boolean status variables are represented as integers. They are true if the variable equals the round counter and false otherwise. They are set to true by setting them to the value of the round counter. They can be easily reset all at once by increasing the round counter.

This technique can be extended to other data types, e.g. floating point numbers, by using an additional boolean variable. The value of the original variable is only considered if the boolean variable is true, otherwise a default value is used. When the original variable is set, the boolean variable is set to true. The original variable can be reset by resetting the boolean variable. The boolean variable is represented and reset as described above.

5.5 Configurable Parameters

Name	Symbol	Default value	Description
Number of roadmap nodes	n	16384	
Connection candidates per roadmap node	k	20	The number of nearest neighbors that are considered for creating an edge to.
Roadmap epsilon	$\epsilon_{roadmap}$	0.01 m	Maximum distance between two tested configurations for checking a path segment for collision while creating the roadmap during the preprocessing stage.
Realtime epsilon	$\epsilon_{realtime}$	0.05 m	Maximum distance between two tested configurations for checking a path segment for collision while connecting the start and goal configurations during the online planning stage. This is normally set larger as the roadmap epsilon as time is more valuable during the online planning stage.
Workspace grid resolution	l_{grid}	0.05 m	Edge length of a cubic workspace grid cell.
Voxelization surface resolution		0.015 m	Edge length of the square grid that is used to sample the robot surface for generating the workspace mapping during the preprocessing stage.
Realtime voxelization surface resolution		0.03 m	Edge length of the square grid that is used to sample the surface of the environment model. This is normally set larger than the voxelization surface resolution because the sampled points mapped during the online planning stage.
Block modeled obstacles		true	Whether to block parts of the roadmap that are in collision with the environment model.
Block sensed obstacles		false	Whether to block parts of the roadmap that are in collision with the point cloud retrieved from the sensor.

Chapter 6

Results

The main result of my work is that the path planner is able to plan a path in about 100 milliseconds while taking sensor data into account. This meets the 5-10 Hz planning rate requirement set up in the problem description.

Timing results for the preprocessing stage were retrieved on a 3 GHz Intel Pentium 4. Timing results for the planning stage and for loading the roadmap were retrieved on a 2 GHz Intel Pentium M, which is the computer on the Care-O-bot responsible for controlling the arm.

6.1 Preprocessing Stage

As figure 6.1 shows, the preprocessing takes several hours and depends on the size of the roadmap. Generating and voxelizing nodes are too fast to be seen in the plot. Searching for the k nearest neighbors of each node takes by far the most time. This is because the k nearest neighbors are retrieved through a brute force search. Figure 6.2 shows that a cover tree search is a lot faster and scales better with the number of nodes. However, as described in section 5.3.1, a cover tree does not return the correct result for the metric used here.

In average, each node has 23 neighbors, independent of the roadmap size. I. e. there are about 11.5 times as many edges as there are nodes in the roadmap. For 16,384 nodes, this is about 188,000 edges.

The voxelization algorithm for mapping a configuration into a set of workspace cells occupied by the robot samples 4,558 points within the robot volume and on its surface. (Voxelization surface resolution: 1.5 cm)

Figures 6.3 and 6.4 show how the size of the roadmap and the workspace mapping depend on the number of roadmap nodes and the resolution of the workspace grid. The largest part of the file size is actually the workspace mapping. The size of the mapping is almost linear in both, the number of roadmap nodes and the number of workspace grid cells. The time needed to load the roadmap and the workspace mapping depends on their size and is shown in figure 6.5.

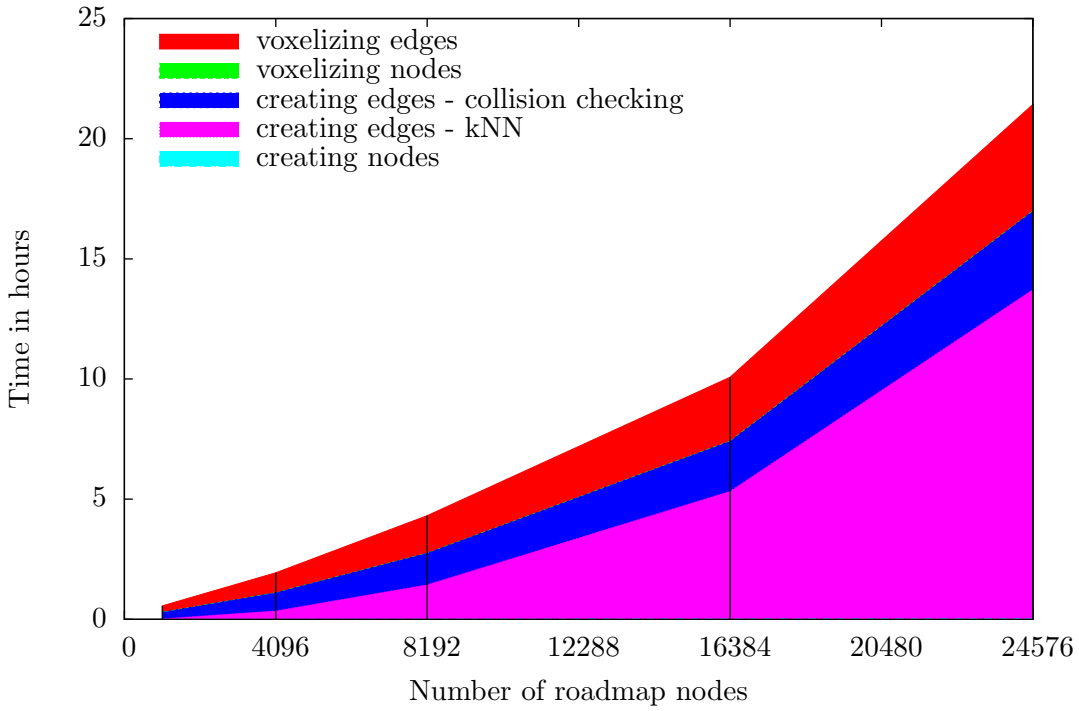


Figure 6.1: Preprocessing time

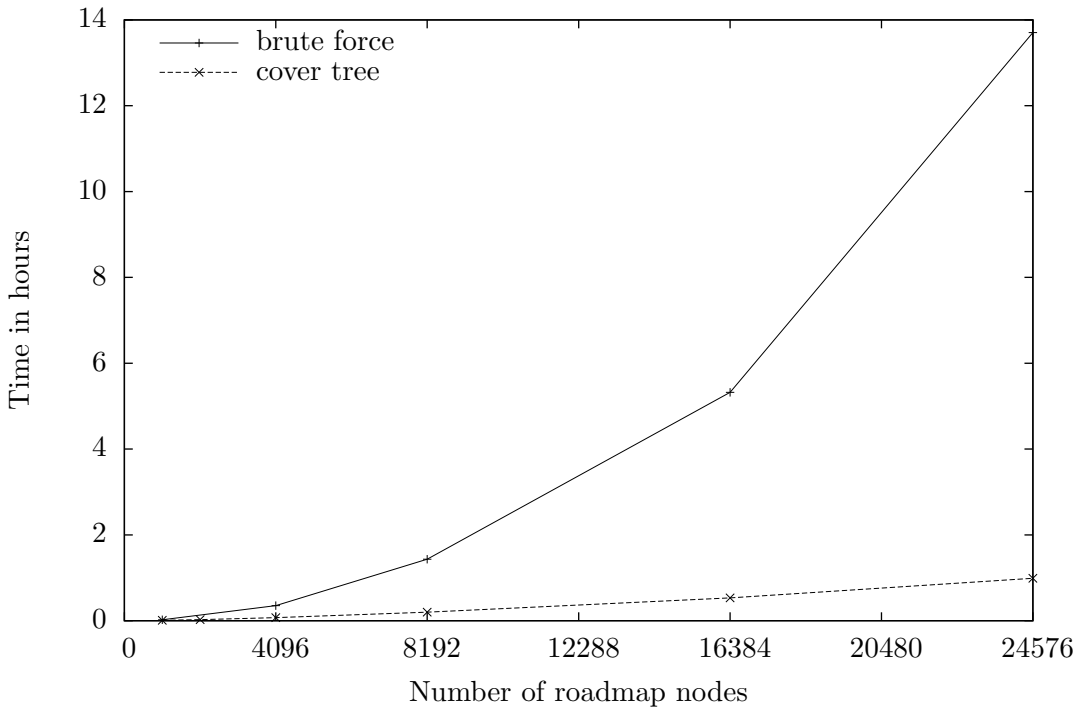


Figure 6.2: Time needed for kNN searches: Comparison of brute force and cover tree search

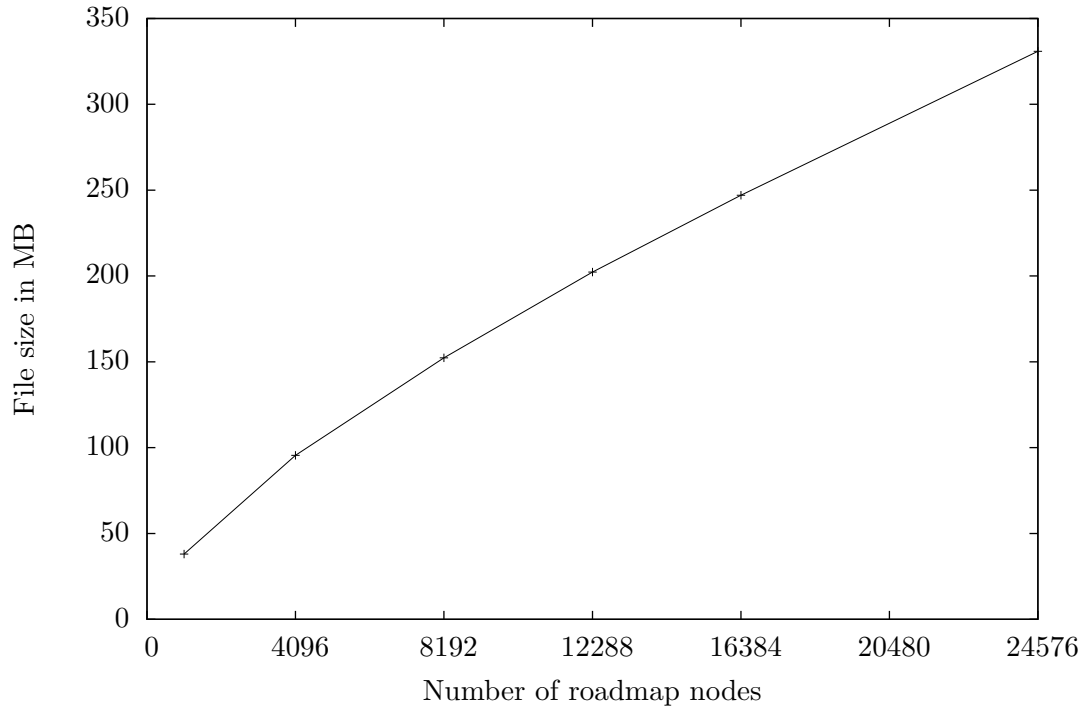


Figure 6.3: File size of saved roadmap and workspace mapping against number of roadmap nodes

Figure 6.4: File size of saved roadmap and workspace mapping against the resolution of the workspace grid

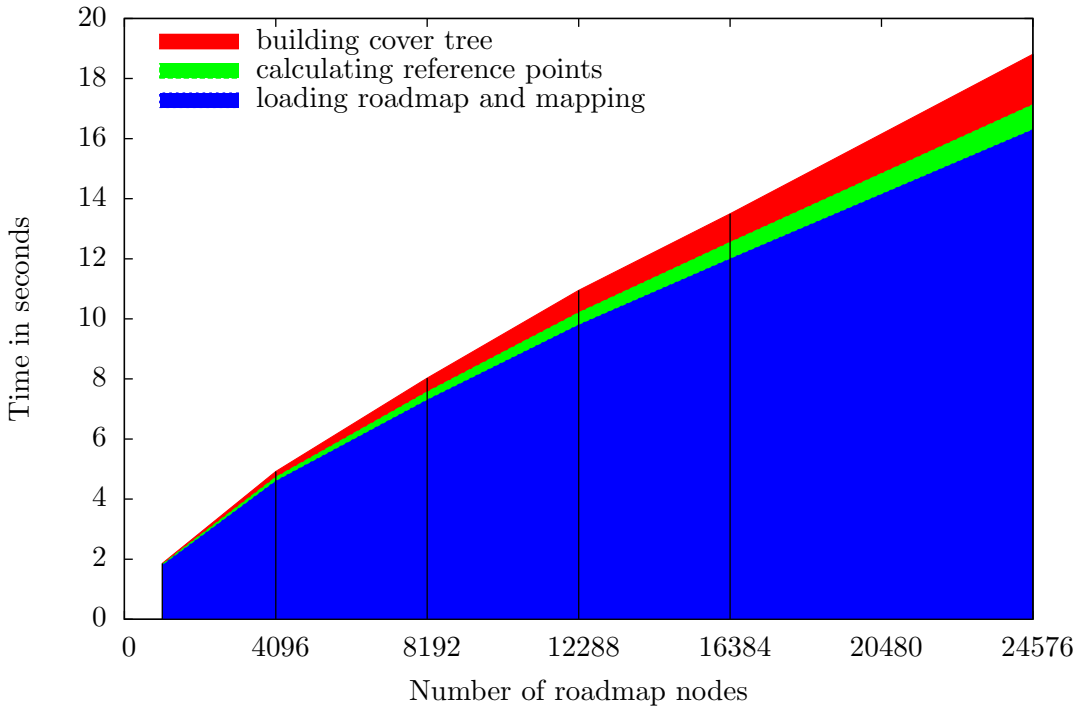


Figure 6.5: Time needed to load roadmap and workspace mapping

6.2 Planning Stage

All experiments in this section are based on a single planning run with a single roadmap for each data point. A specific roadmap, which is generated randomly, might be better or worse for a particular planning query. This fact adds some randomness to the results and could have been countered by averaging each data point over several planning runs with different roadmaps.

Figure 6.6 is the most important figure in this chapter. It shows the time required for one online planning round. The time needed is around 100 milliseconds, almost independent of the number of roadmap nodes. The graph is split up to show how much time is spent on the individual subtasks. Some subtasks do depend on the size of the roadmap. The kNN and graph searches need more time with growing roadmap size. Testing path segments, in contrast, takes less time with growing roadmap size. This is because a larger roadmap means the configuration space is more densely covered by the roadmap. Thus, the k nearest neighbors of the start and goal configurations are closer to the start and goal configurations and there are fewer collision checks necessary to test those path segments. Acquiring the sensor data over the network currently takes about 20 milliseconds, which is quite long and could probably be optimized.

I use A* for searching the graph. Figure 6.7 shows that A* is much faster than Dijkstra's algorithm, which does not make use of a heuristic function. This is not self-evident. With my initial implementation A* was actually slower than Dijkstra's algorithm because the time gained by expanding fewer nodes was more than made up for by the time necessary to compute the metric used as heuristic function. Only the precomputation of the reference points for the 2-norm in workspace metric made A* perform fast.

Figures 6.8 and 6.9 show how the resulting path depends on the size of the roadmap. For a larger roadmap the resulting path consists of more intermediate configurations but the path cost in lower.

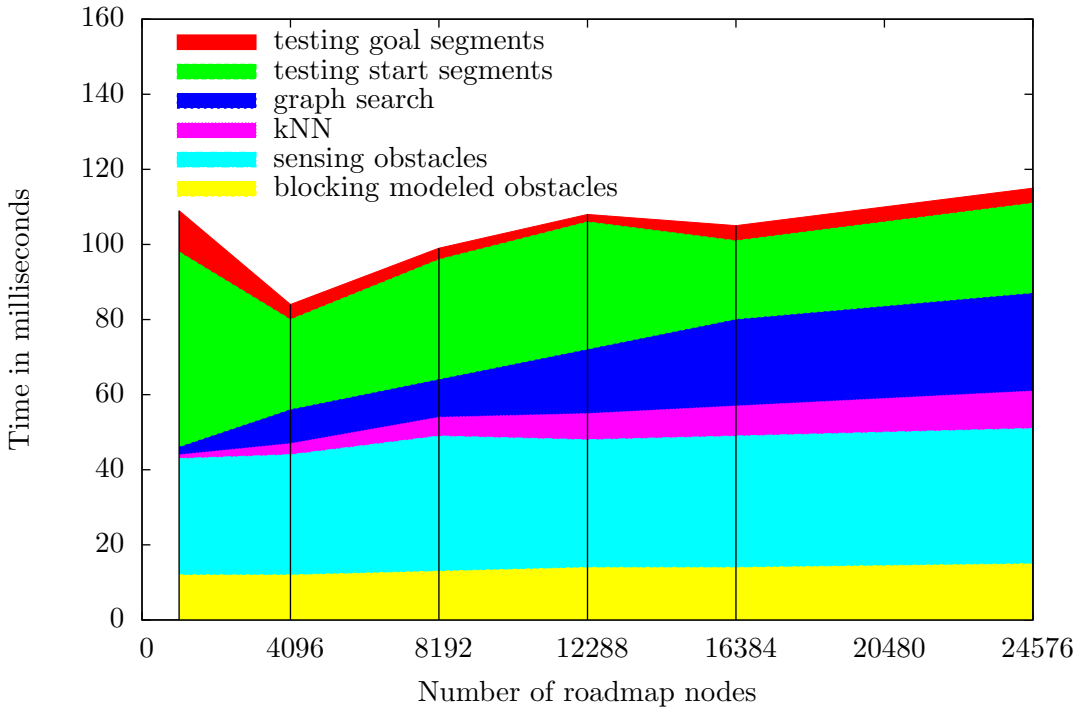


Figure 6.6: Planning time

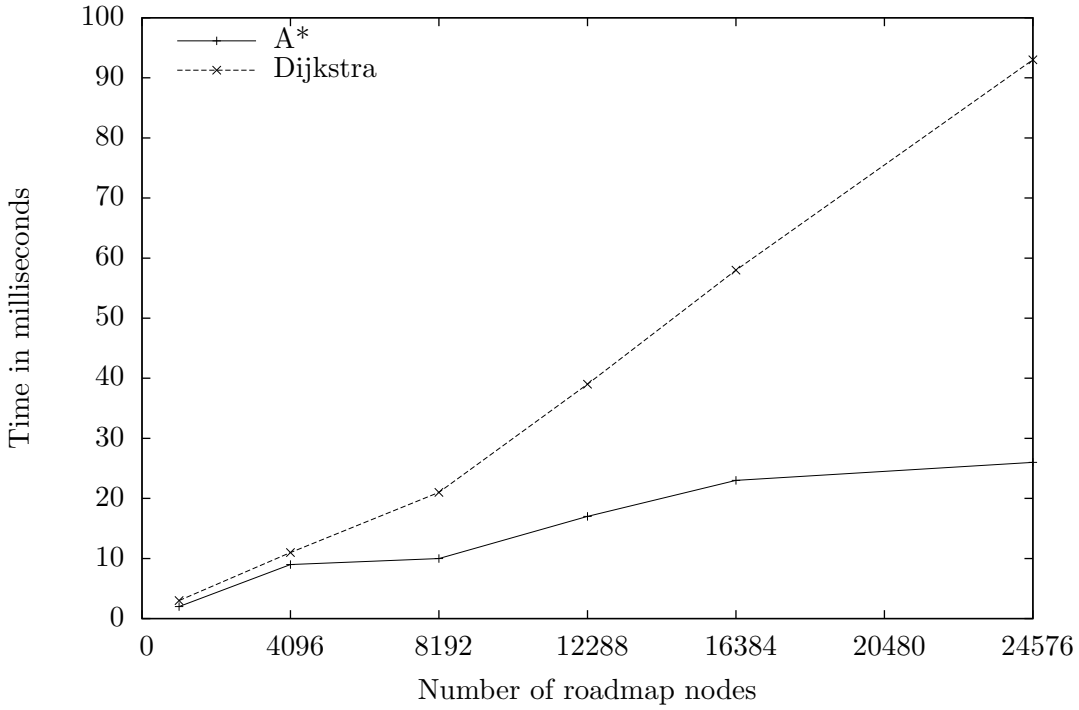


Figure 6.7: Comparison of search times using A* and Dijkstra's algorithm

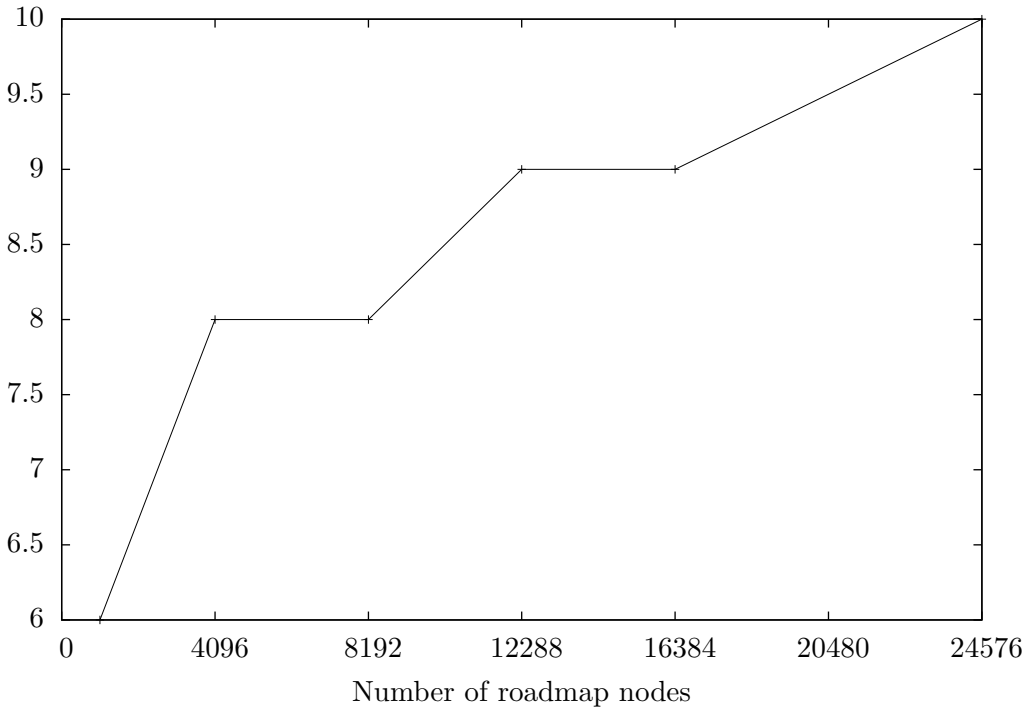


Figure 6.8: Number of intermediate configurations on the path including start and goal

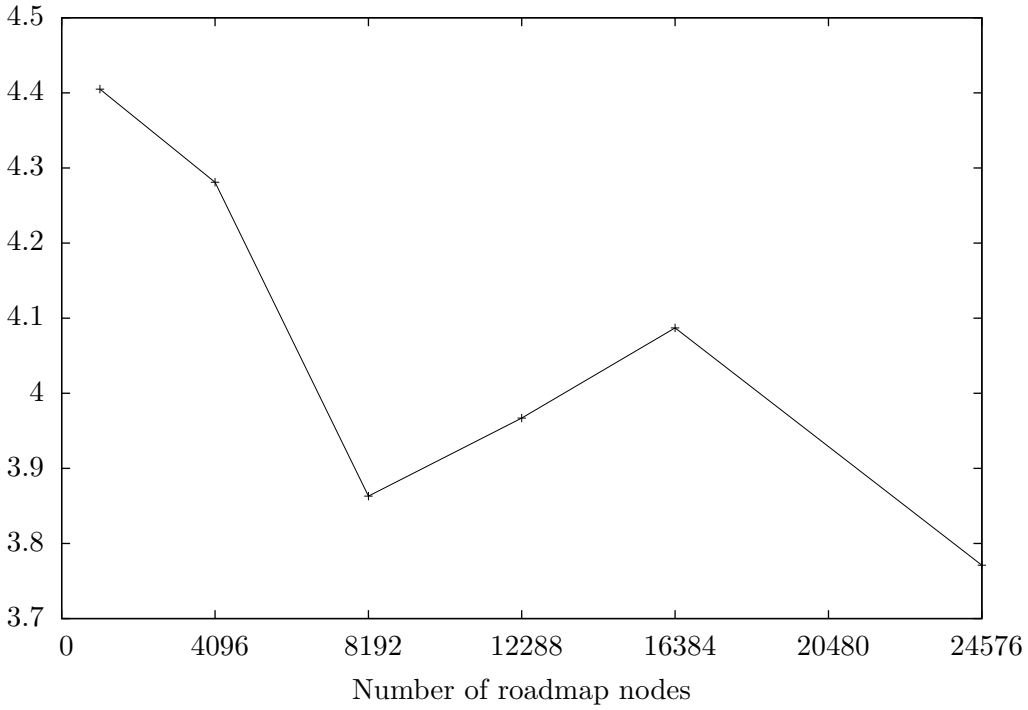


Figure 6.9: Path cost

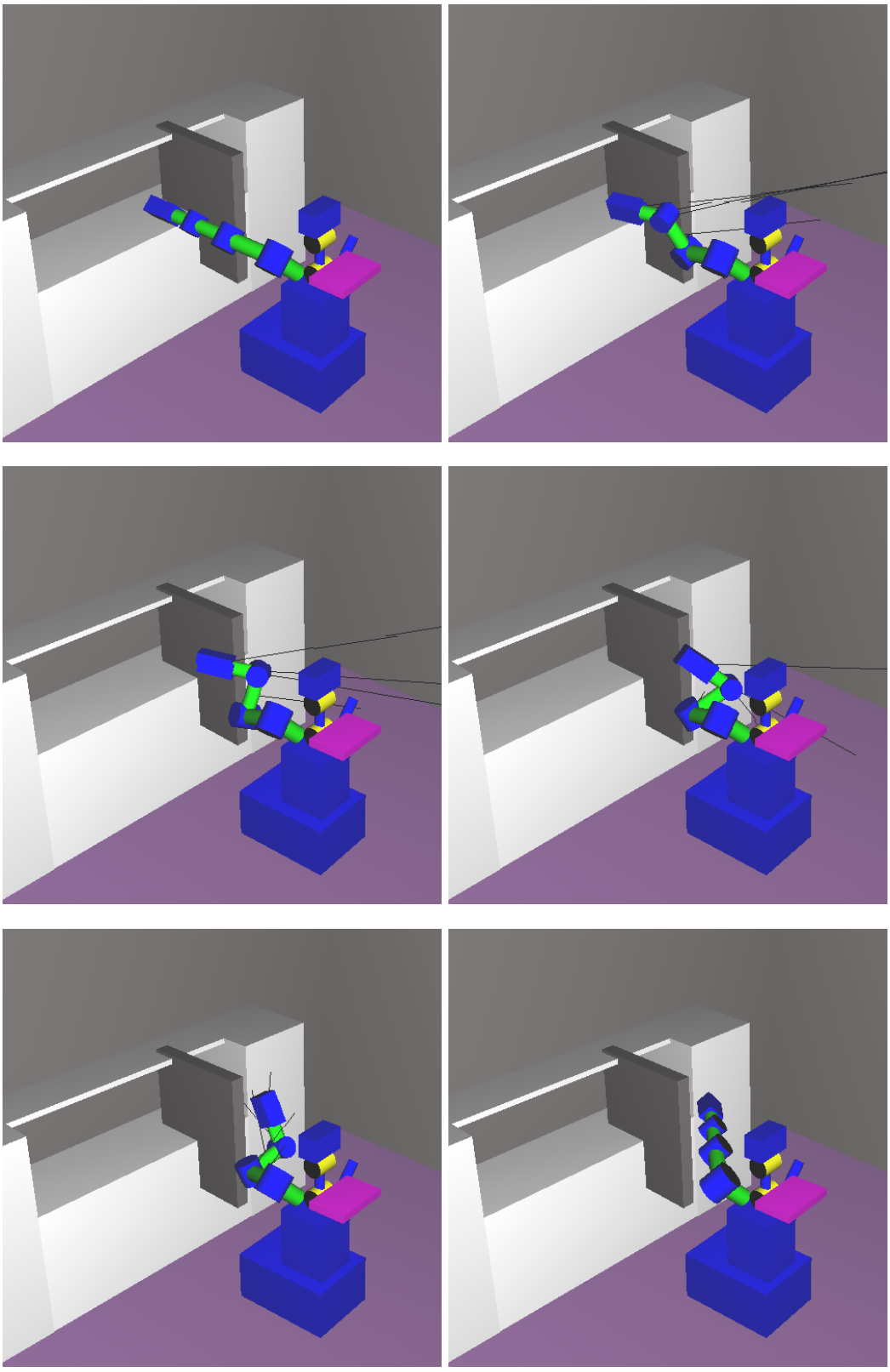


Figure 6.10: The robot planning around a modeled obstacle in simulation.

6.3 Resulting Motion

This section shows the paths produced by the planner in both simulation and on the real robot. Figure 6.10 shows the robot planning around a modeled obstacle in simulation. Figures 6.11, 6.12 and 6.13 show the real robot planning in between two fixed configurations. In figure 6.11 there is no obstacle interfering with the movement of the robot. The robot moves in between the two configurations relatively straight. The small contraction of the arm is probably caused by the discrete roadmap configurations the arm has to move through and the fact that contracting the arm a little bit actually leads to a shorter path. Figure 6.12 and 6.13 show the robot moving around a human obstacle, which was detected using the SwissRanger sensor. In figure 6.13 the human is closer and the robot is having a harder time avoiding the obstacle.

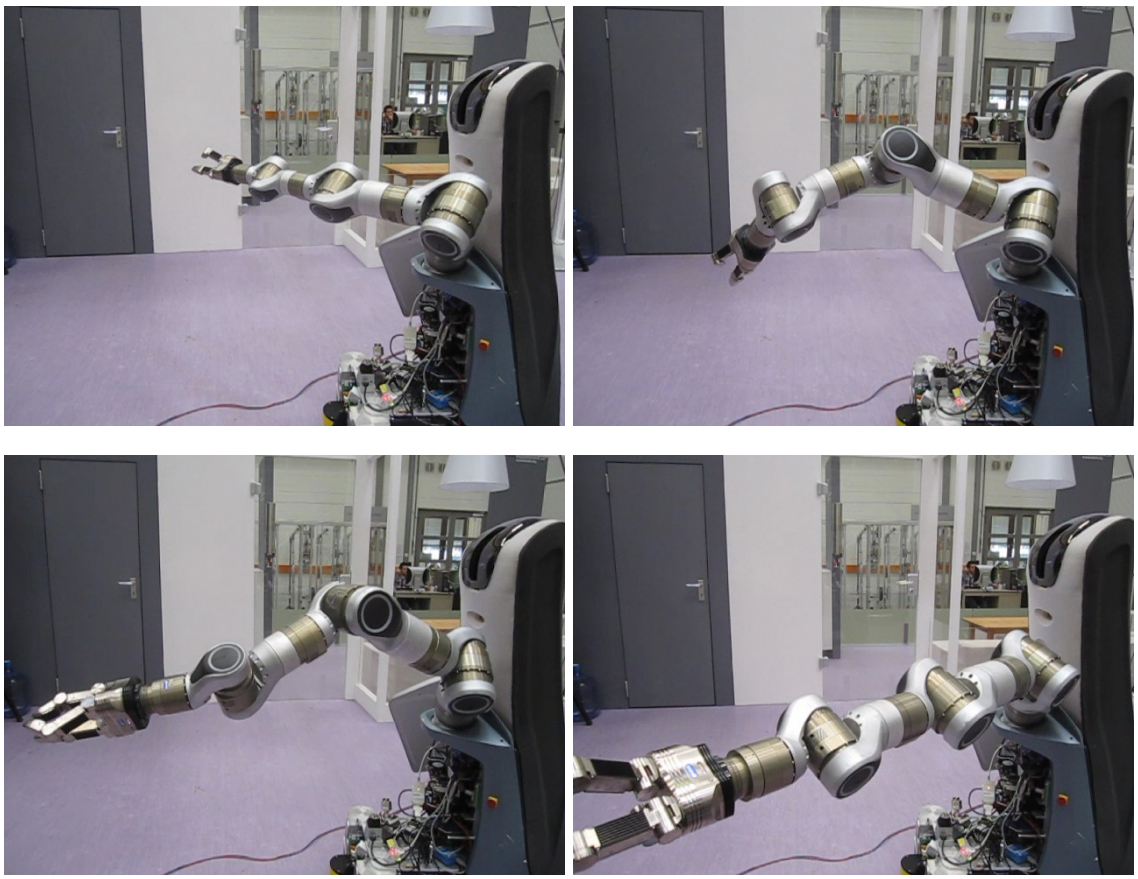


Figure 6.11: The robot planning in between two configurations without any obstacles interfering with the robot. This is for comparison with figures 6.12 and 6.13.

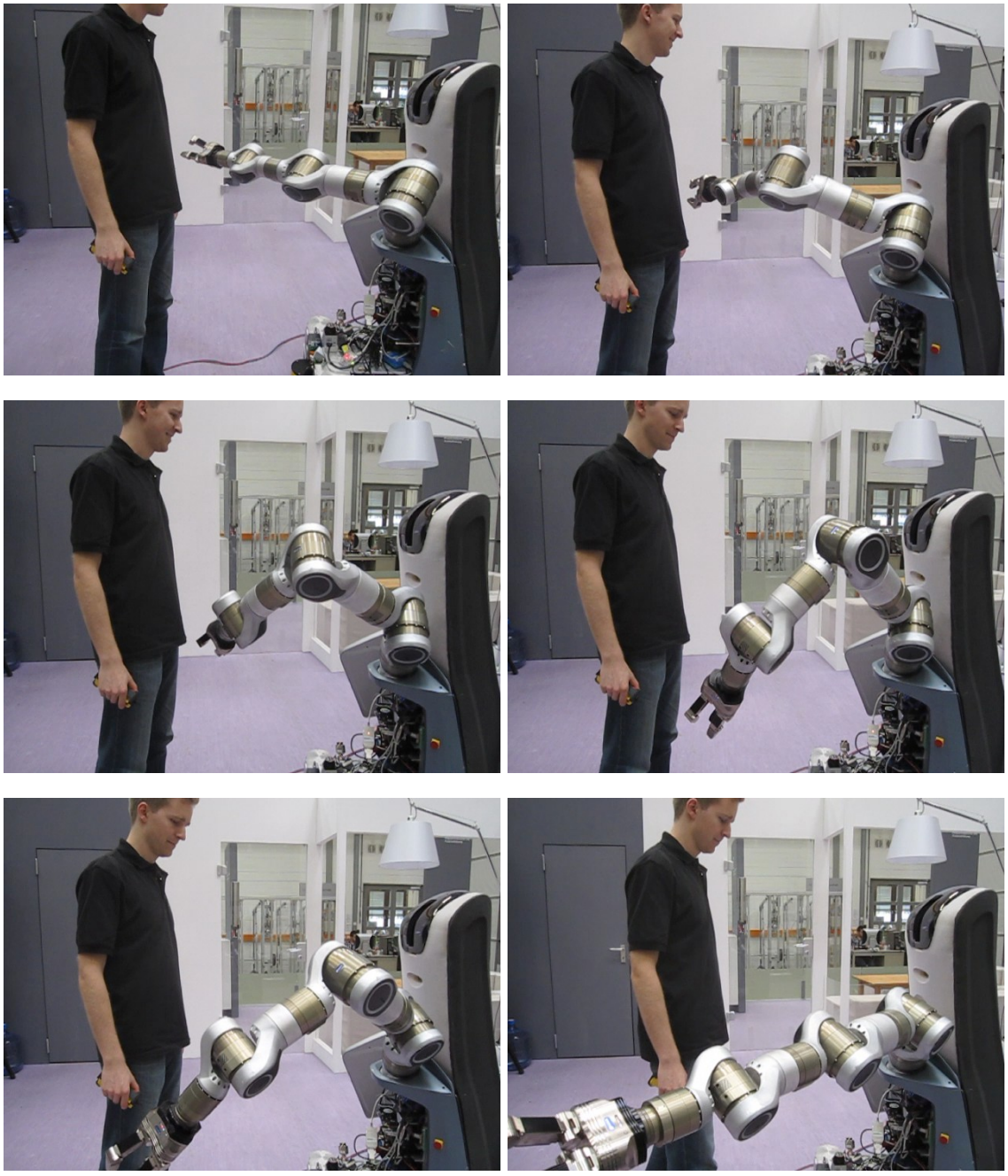


Figure 6.12: The robot planning around a human obstacle that was detected by the SwissRanger sensor.

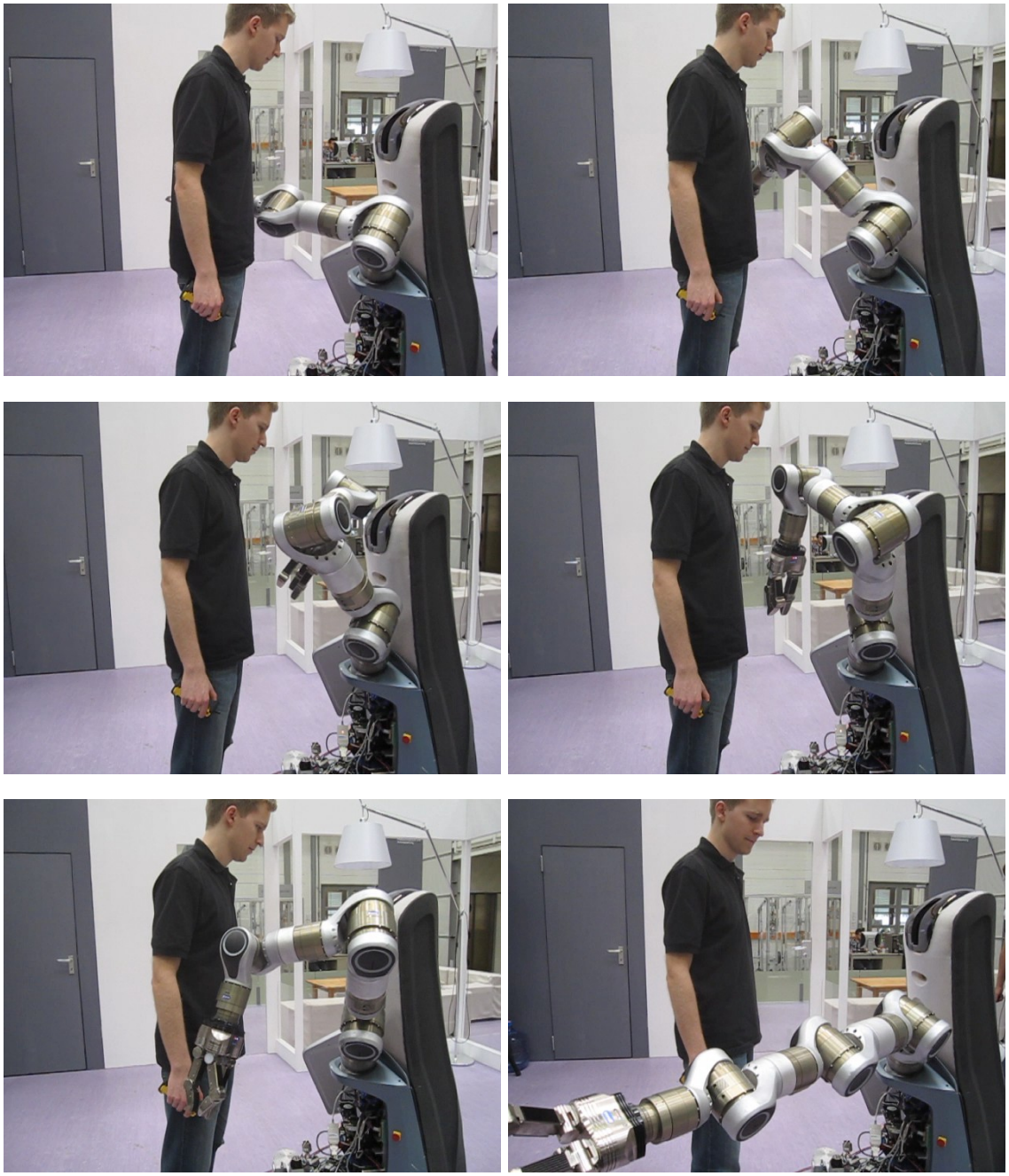


Figure 6.13: The robot planning around a close human obstacle.

6.4 Removing Robot from SwissRanger Data

As mentioned earlier, removing the arm from the sensor data did not work well enough. Figure 6.14 shows the result of the removal. The basic method only removes points that fall within workspace grid cells that are occupied by the robot arm. The enhanced method is more tolerant and removes more points including some that represent real obstacles, e. g. a human hand.

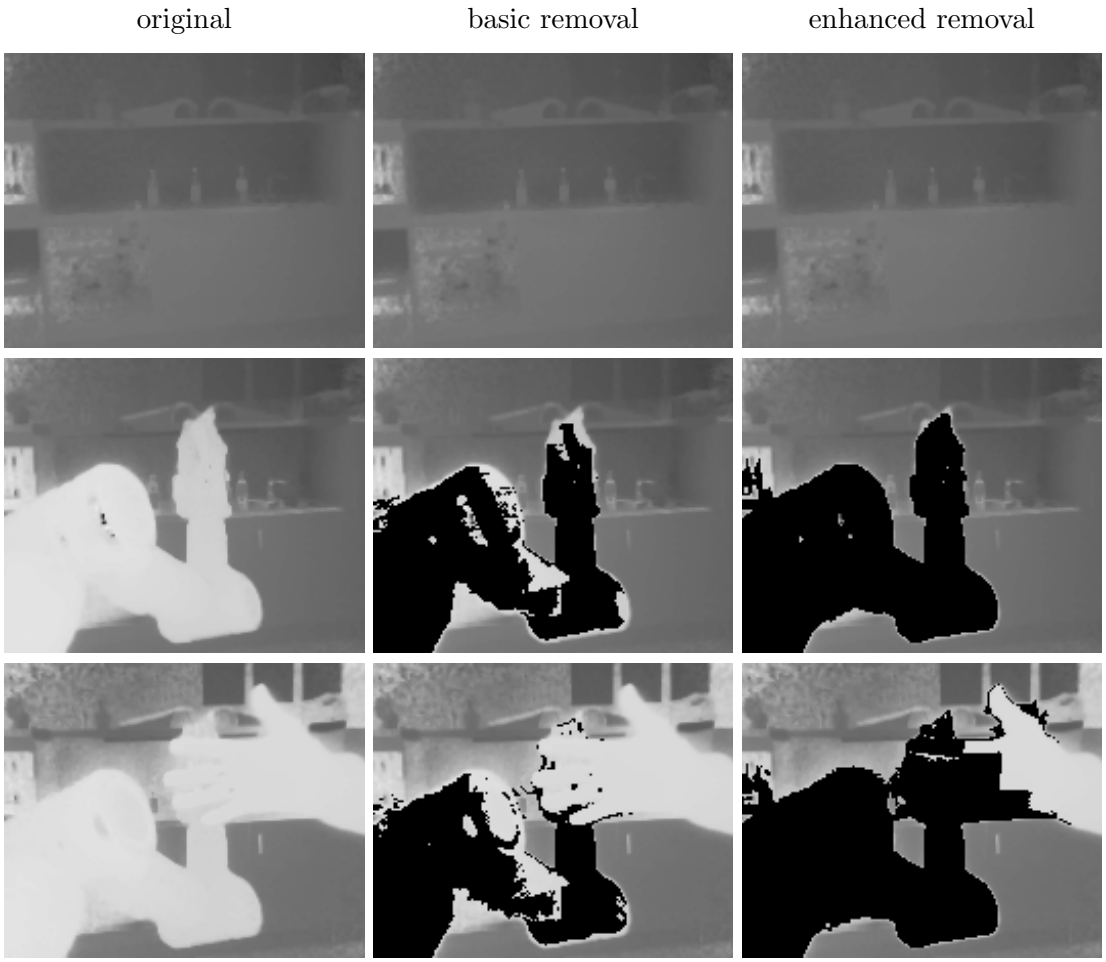


Figure 6.14: Trying to remove the robot from the sensor data. Light areas are close, dark areas are far away. Black parts have been attributed to the robot arm and are subsequently removed from the sensor data. The enhanced removal is more tolerant in attributing sensor points to the arm.

Chapter 7

Future Work

This section gives ideas on how my work could be continued and improved.

7.1 Roadmap Enhancements

The roadmap could be enhanced by replacing the uniform sampling of configurations with a manipulability-based sampling as described by Leven and Hutchinson [6]. Among other things, this would create more roadmap nodes at the border of the arm's workspace. This includes configurations in which the arm is nearly fully extended. This could be especially helpful for grasping objects that are close to the border of the arm's workspace. Also taking more samples close to a self-collision could create more optimal movements of the arm closely around the robot body.

The roadmap could also be enhanced by changing the way edges are created. Currently candidates for connection are determined solely based on distance. But the direction of the candidates is also important. To plan short and smooth paths it is advantageous for a node to have connections in all directions. Currently the high number of 20 connection candidates per node make it probable that all directions are covered well enough. If we could bias the selection of connection candidates towards a good distribution in the direction of the candidates, we would need to create fewer edges in order to obtain the same quality paths. This improvement is suggested in [7] with the remark that it has not been given much attention, yet.

Leven and Hutchinson [6] apply several enhancements to the roadmap after generating it to make it more resilient against becoming disconnected. This includes adding edges in loosely connected areas of the roadmap. These enhancements are currently not implemented but could be in the future.

7.2 Compression

Leven and Hutchinson [6] devote a large part of their paper to compress the roadmap and the workspace mapping. The implemented path planner currently does not compress the roadmap and workspace mapping. However, if we want to extent the planner to include more degrees of freedom, it might become necessary to consider compression.

The workspace mapping could be compressed and, more importantly, the invalidation process sped up by introducing a hierarchical workspace grid. As it is very likely that

neighboring workspace cells are in collision with the same parts of the roadmap, a superior, larger grid cell could include all nodes and edges of the roadmap that are in collision with all subordinate workspace cells.

7.3 Optimizing Underlying Code

My planner builds on top of existing code on the Care-O-bot and uses the functionality provided. However, most of the code on the Care-O-bot was developed without realtime execution being a design goal. Thus, I am confident that a lot of the underlying code is not as fast as it could be. Examples for underlying functionality that has been used by my planner and possible candidates for improvements are the collision checker and kinematics. Speeding up kinematics would result in a speed up of the workspace metrics.

Retrieving the current sensor data over the network takes about 20 milliseconds. There might also be some potential for improvement.

7.4 A Better Voxelization Algorithm

Currently, voxelization is based on sampling points and then mapping those points into the workspace grid. Leven and Hutchins use a different voxelization algorithm similar to one by Kaufman and Shimony [22]. Replacing the current voxelization algorithm with the one described in [22] or a similar one could yield a speed-up and a more accurate voxelization.

7.5 Integrated Planning

Currently the planning is limited to the arm. The planner can only plan a path for the arm to locations that are within the reach of the arm. Also, obstacles might hinder the robot from reaching a place that it could reach if it moved the robot base. To increase the range of the robot and to give the robot more possibilities to reach places in the presence of obstacles, we need to integrate the movement of the robot base into the planning process. This adds 3 degrees of freedom to the 7 degrees of freedom of the arm.

Integrating the robot base into the planning process would lead to a significantly larger and less dense roadmap. It is probably necessary to improve my implementation before it can be extended to 10 degrees of freedom. The workspace grid would not be fixed on the robot anymore but on the world coordinate frame covering the whole space the robot is supposed to act within. Known, static obstacles could already be considered during the roadmap creation, which would save some time during the online planning stage.

Including the robot base into the roadmap would create some redundancy. For all configurations in which the robot is not close to obstacles, the 7 dimensions of the roadmap representing the arm are very similar and interchangeable. May be, there is a way to exploit this redundancy to keep the roadmap from growing too large.

7.6 Elimination of Robot Parts from SwissRanger Data

In order to allow planning online while the arm is possibly within the field of view of the sensor, we need to be able to reliably remove the robot arm from the SwissRanger Data. In order to achieve good results with the inaccurate sensor, it might be necessary to model the SwissRanger sensor probabilistically. This requires good knowledge of how the sensor works and under which circumstances which inaccuracies occur. This would enable us to assign each sensor point a probability for actually being a real obstacle.

7.7 Update Path Online

If the robot can be removed from the sensor data reliably, updating the path online is rather simple to solve. It needs some small architectural changes to make the path planner run in parallel to the execution of the trajectory. Also the trajectory execution layer might need to be adapted to accept a new changed path during the execution of the current one.

7.8 Integration of GPU-based Sensor Data Modeling

At Fraunhofer IPA a GPU-based sensor data modeling has been implemented. This work is unrelated to the Care-O-bot but also uses a SwissRanger sensor. In this work the robot is removed from the sensor data and a workspace grid is updated with the sensor information. This also takes old sensor data into account in areas that are currently not in the field of view. The GPU is used to speed up the modeling and achieve update rates of about 10 Hz. The fact that this work also uses a workspace grid makes it easier to be integrated with the planner for the Care-O-bot. However, some customizing is probably necessary to make it work with the Care-O-bot.

Chapter 8

Conclusion

Based on the given real-time requirements for a planning rate of 5-10 Hz and the requirement to integrate sensor data, the PRMCE algorithm was chosen as basis for this work. The fast planning process results not only from the chosen algorithm but also from implementing time-critical parts efficiently and making the right choices, especially in regard to the configuration space metrics used and the way roadmap parts are invalidated.

The path planner presented in this work is able to calculate a collision-free path for the 7-DOF manipulator of Care-O-bot 3 within about 100 milliseconds. This includes the whole perception-action cycle from acquiring sensor data until returning the planned path, which takes sensor data into account. This meets the requirement of a planning rate of 5-10 Hz set up in the problem description and enables us to dynamically replan in case of environment changes.

The main contribution of my work is to demonstrate that the PRMCE algorithm introduced by Leven and Hutchinson [6] can meet real-time requirements on a real robot using real sensor data. Leven and Hutchinson only reported timing simulation results for abstract robots. The fastest planning time they reported was "less than 1 s". My implementation is about ten times faster than that, which might be partially caused by a faster computer. It is also about 10 to 20 times faster than the path planner used on the Care-O-bot before.

In general, there has been little work to combine (real-time) motion planning for a robot arm with real 3D sensor data. This might be caused by the non-availability of well-suited sensors so far. The SwissRanger sensor is a relatively new sensor, which is also still being improved. My work tackles this problem, which is fundamental to the goal of having real robots act in human environments.

Bibliography

[1] S. Quinlan and O. Khatib, "Elastic bands: connecting path planning and control," *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, pp. 802–807 vol.2, May 1993.

[2] S. Quinlan, *Real-time modification of collision-free paths*. PhD thesis, Stanford University, Stanford, CA, USA, 1995.

[3] K. Thurow, "Concept and implementation of the kinematics and path planning algorithms for a 7 degree of freedoms (dof) robot manipulator," Master's thesis, Mid Sweden University, 2007.

[4] R. Volz, "Entwurf und Implementierung einer schnellen Kollisionsdetektion für einen 7 dof Roboter-Manipulator," diplom thesis, Hochschule für Technik und Wirtschaft Berlin, 2007.

[5] G. Snchez and J. claude Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," in *In Int. Symp. Robotics Research*, pp. 403–417, 2001.

[6] P. Leven and S. Hutchinson, "A Framework for Real-time Path Planning in Changing Environments," *The International Journal of Robotics Research*, vol. 21, no. 12, pp. 999–1030, 2002.

[7] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006. Available at http://planning.cs.uiuc.edu/.

[8] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[9] O. Brock and O. Khatib, "Elastic Strips: A Framework for Motion Generation in Human Environments," *The International Journal of Robotics Research*, vol. 21, no. 12, pp. 1031–1052, 2002.

[10] O. Brock, *Generating robot motion: the integration of planning and execution*. PhD thesis, Stanford University, Stanford, CA, USA, 2000.

[11] O. Brock and L. Kavraki, "Decomposition-based motion planning: a framework for real-time motion planning in high-dimensional configuration spaces," *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1469–1474 vol.2, 2001.

[12] O. Brock and L. Kavraki, "Decomposition-based motion planning: Towards real-time planning for robots with many degrees of freedom," 2000.

[13] K. Belghith, F. Kabanza, L. Hartman, and R. Nkambou, "Anytime dynamic path-planning with flexible probabilistic roadmaps," in *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, pp. 2372–2377, May 2006.

[14] Y. Yang and O. Brock, "Elastic roadmaps: Globally task-consistent motion for autonomous mobile manipulation in dynamic environments," in *Proceedings of Robotics: Science and Systems*, (Philadelphia, USA), August 2006.

[15] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE Transactions on Robotics and Automation*, vol. RA-3, pp. 43–53, February 1987.

[16] L. Kavraki and J.-C. Latombe, "Randomized preprocessing of configuration space for fast path planning," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pp. 2138–2145 vol.3, May 1994.

[17] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[18] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic a*: An anytime, replanning algorithm," in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, June 2005.

[19] M. Likhachev, G. J. Gordon, and S. Thrun, "Ara*: Anytime a* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems 16* (S. Thrun, L. Saul, and B. Schölkopf, eds.), Cambridge, MA: MIT Press, 2004.

[20] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Transactions on Robotics*, vol. 21, pp. 354–363, June 2005.

[21] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, pp. 100–107, July 1968.

[22] A. Kaufman and E. Shimony, "3d scan-conversion algorithms for voxel-based graphics," in *SI3D '86: Proceedings of the 1986 workshop on Interactive 3D graphics*, (New York, NY, USA), pp. 45–75, ACM, 1987.

[23] A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," in *ICML '06: Proceedings of the 23rd international conference on Machine learning*, (New York, NY, USA), pp. 97–104, ACM, 2006.

[24] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

(Tobias Kunz)