A FRAMEWORK FOR REAL-TIME PATH PLANNING IN CHANGING ENVIRONMENTS

BY

PETER J. LEVEN

B.S., Michigan Technological University, 1993
B.S., Michigan Technological University, 1993
M.S., University of Illinois at Urbana-Champaign, 1997

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2001

Urbana, Illinois

© Copyright by Peter J. Leven, 2001

ABSTRACT

We present a new method for generating collision-free paths for robots operating in changing environments. Our approach is closely related to recent probabilistic roadmap approaches. These planners use preprocessing and query stages, and are aimed at planning many times in the same environment. In contrast, our preprocessing stage creates a representation of the configuration space that can be easily modified in real time to account for changes in the environment. As with previous approaches, we begin by constructing a graph that represents a roadmap in the configuration space, but we do not construct this graph for a specific workspace. Instead, we construct the graph for an obstacle-free workspace, and encode the mapping from workspace cells to nodes and arcs in the graph. When the environment changes, this mapping is used to make the appropriate modifications to the graph, and plans can be generated by searching the modified graph.

As part of our presentation, we discuss the construction of the roadmap, including how samples of the configuration space may be generated and how to connect the samples to form the roadmap. We then discuss the mapping from the workspace to the roadmap, explaining efficient techniques for its generation and representation. We continue with some robustness measures and how these may be used to enhance the robustness of the roadmap. We then present an extension of our method for mobile robots. Finally, we evaluate an implementation of our approach for serial-link manipulators with up to 20 joints.

Dedicated to my parents, Peter J. and Marie E. Leven

ACKNOWLEDGMENTS

I would like to thank my parents for their patience, support, and encouragement while I toiled seemingly endlessly in school.

I would like to thank my advisor, Professor Seth Hutchinson, for providing a rich, creative environment in which to work. His ideas and guidance have been invaluable.

I would like to thank the robot vision research group of the Lehrstuhl für Realzeit-Computersysteme at the Technische Universität München for their invitation to spend nine months in their lab. I am especially grateful to Alexa Hauck for making the trip possible. I would also like to thank Christoph Zierl, Darius Burschka, Norbert Stöffler, Siegfied Wieler, Christof Eberst, Franz Fischer, Thomas Hopfner, Stefan Blum, and Bernhard Kuhn whose extra effort made the visit much more enjoyable.

I would also like to thank my peers in the Computer Vision and Robotics research group for their many helpful discussions. Without the many discussions, this research would not have been nearly as interesting as it has been. I would particularly like to thank Kevin Nickels, Yakup Genc, Mark Hyman, Jeff Ho, Fred Rothganger, and Ricardo Swain.

TABLE OF CONTENTS

CHAPTER **PAGE**

1 INTRODUCTION 1

2 CONSTRUCTING THE ROADMAP 5
2.1 Generating Sample Configurations 6
2.1.1 Previous sampling approaches 6
2.1.2 Our sampling schemes 8
2.2 Local Planner 11
2.2.1 Local planners used in traditional PRM methods 11
2.2.2 Local planner for our roadmap 12
2.3 Distance Function 13
2.4 Efficient Nearest-Neighbor Computation 18
2.4.1 Our approaches for nearest-neighbor computation 19
2.5 Summary 20

3 WORKSPACE TO CONFIGURATION SPACE MAPPING 21
3.1 Computation of ϕ_n and ϕ_a 22
3.2 Efficient Representations 25
3.2.1 Evolution of data compression 27
3.2.2 Efficient encoding of $\phi(w^*)$ 29
3.2.3 Encoding ϕ over neighborhoods in $\mathcal{W}$ 29
3.2.4 Encoding the labels 31
3.2.5 Summary 35

4 ROADMAP ENHANCEMENTS 37
4.1 Previous Enhancement Methods 37
4.2 Connectivity-based Enhancement 39
4.3 Workspace-based Enhancement 41
4.3.1 Pathological cases 42
4.3.2 Application of ϵ-robustness to robot design 43
4.3.3 Detection of pathological cases and other algorithm details 45
4.4 Summary 47

5 MOBILE ROBOTS 48
5.1 Implementation Details 49
5.1.1 Search procedure 50
5.1.2 Search heuristics 50
5.1.3 Collision avoidance 50
5.2 Other Applications 50

6 RESULTS . . . 52
6.1 Experimental Setup . . . 52
6.2 Roadmap Construction . . . 54
6.2.1 Sampling of configuration space . . . 54
6.2.2 Computing nearest neighbors . . . 55
6.3 Computing ϕ . . . 58
6.3.1 Computing ϕ_n . . . 58
6.3.2 Computing ϕ_a . . . 59
6.3.3 Compression results . . . 62
6.4 Graph Enhancement . . . 71
6.4.1 Enhancement based on minimum cut sets . . . 71
6.4.2 Enhancement based on ϵ-robustness . . . 72
6.5 Some Planning Examples . . . 80
6.5.1 Fixed-base robots . . . 80
6.5.2 Mobile robots . . . 82

7 CONCLUSION . . . 85

APPENDIX A PLANAR ROBOTS WITHOUT JOINT LIMITS . . . 87
A.1 Experimental Setup . . . 87
A.2 Results for Planar Robots without Joint Limits . . . 87
A.2.1 Configuration space sampling . . . 88
A.2.2 Graph construction . . . 88
A.2.3 Computing ϕ_n . . . 90
A.2.4 Computing ϕ_a . . . 90
A.2.5 Some compression results . . . 92
A.2.6 Some planning examples . . . 95
A.3 Problems . . . 95

APPENDIX B MANIPULABILITY OF ROBOTS . . . 98

REFERENCES . . . 106

VITA . . . 113

LIST OF TABLES

Table **Page**

3.1 Examples of variable-length codes for integers. These codes do not include a representation of zero. . . . 33
3.2 More examples of variable-length codes for integers. These codes do include a representation of zero. . . . 34
3.3 Golumb codes for $b = 11, 17$, and 23. . . . 35
3.4 Number of kilobytes required using the various codes for ϕ for a six-DOF robot example in a 3D environment. The roadmap has 2048 nodes and ϕ was encoded using the region-growing approach described in Section 3.2.3. . . . 36

6.1 Definition of the labels used on figures for distance functions. . . . 60
6.2 Definition of the labels used on figures for the different encoding models. . . . 70
6.3 The effect on ϵ-robustness of increasing the number of arcs per node in the initial roadmap for 3D workspace robots and roadmaps with different numbers of nodes. . . . 75
6.4 Results for ϵ-robustness for a 2048-node roadmap created for the type A robots. . . . 76
6.5 Results for ϵ-robustness for a 2048-node roadmap created for the type B robots. . . . 76
6.6 Number of arcs in 2048-roadmap for which the robot collides with itself while following the path corresponding to the arc. . . . 77
6.7 Arcs rejected due to self-collision during ϵ-robustness enhancement. . . . 78
6.8 The results for creating roadmaps with the number of arcs per node proportional to the number of joints in the robot. . . . 79

LIST OF FIGURES

Figure **Page**

2.1 Planar robot example. 9
2.2 Sample distributions for a two-joint planar robot: (a) uniform, (b) higher density in regions of low manipulability, (c) higher density in regions of high manipulability. 10
2.3 Probability distribution functions for the manipulability of a robot with six joints: (a) a planar robot and (b) a 3D workspace robot. 11
2.4 The roadmaps that result from using the four distance functions. 16
2.5 (a) Two configurations considered close by the $\mathcal{D}_2^{\mathcal{W}}$ function and (b) the area swept by the robot following a straight-line in $\mathcal{C}$ connecting the two configurations. 17
2.6 Illustration of d_x, d_y, and d_z. 17

3.1 (a) The workspace of a two-link robot with an obstacle at (0, 10), and (b) the corresponding configuration space of the robot. 22
3.2 An example of (a) $\phi_n^{-1}(\mathrm{q})$ and (b) $\phi_a^{-1}(\gamma)$. 23
3.3 Algorithm for computing the set of cells in $\phi_n^{-1}(\mathrm{q})$. 24
3.4 Illustration of the subdivision method for computing $\phi_a^{-1}(\gamma)$. 25
3.5 Size of ϕ_a using a naive encoding. 25
3.6 The two neighborhoods w^*: $\eta_1(w^*)$ is the light region containing w^* and $\eta_2(w^*)$ is the darker region. 30

4.1 A roadmap (a) before and (b) after enhancement. The dashed lines in (b) correspond to the new arcs. 39
4.2 Effect of an occupied cell in the workspace on the connectivity of the roadmap. The configuration shown in the workspace corresponds to the enlarged node on the right side of configuration space. 41
4.3 The result of ϵ-robustness processing on the roadmap shown in Figure 4.2. Note that the obstacle no longer breaks the roadmap. 42
4.4 An example where the local planner failed to reconnect the roadmap. 43
4.5 Three cases of cells that disconnect $\mathcal{C}_{free}$ and the corresponding $\mathcal{CB}$. The problem cell is shaded and the solid dots represent the rotation axes of the joints. 44
4.6 A problem cell for planar robots. 45
4.7 A robot with and a robot without some pathological cells in the workspace. 46

6.1 Two robots with 3D workspace. 53
6.2 Two mobile robots with 3D workspace. 53
6.3 Number of samples of C-space generated per node accepted: fixed-length planar robots (a), constant length-to-width ratio planar robots (b), and fixed-length robots with 3D workspace (c,d). 55
6.4 Nearest-neighbor computation time results for the $\mathcal{D}_2^{\mathcal{W}}$ distance function: (a) brute force, (b) ANN with $\epsilon = 0$. 56

6.5 Computation time for determining nearest neighbors for constructing the roadmap for the $\mathcal{D}^{\mathcal{W}}_{m2}$ distance function: (a) brute-force exact approach, (b) approximate approach. 57
6.6 The effect of varying ϵ with ANN and a 16 384 node roadmap: (a) $\mathcal{D}^{\mathcal{C}}_{2}$ distance function, (b) $\mathcal{D}^{\mathcal{W}}_{2}$ distance function. For comparison, the brute-force computation time for $\mathcal{D}^{\mathcal{C}}_{2}$ is shown in (a). 58
6.7 Computation times per node for ϕ_n using the two different algorithms: (a) original algorithm based on V-Clip, (b) new algorithm. 59
6.8 Size of ϕ_n for 3D workspace robots. 59
6.9 Size of ϕ_a for planar robots with joint limits and for roadmaps with 16 384 nodes and five distance functions. 60
6.10 Average swept volume per arc in ϕ_a: (a) planar robots without joint limits, $\mathcal{D}^{\mathcal{W}}_{2}$ distance function; (b) planar robots with joint limits, $\mathcal{D}^{\mathcal{W}}_{2}$ distance function; (c) planar robots with joint limits, $\mathcal{D}^{\mathcal{W}}_{m2}$ distance function; (d) 3D workspace robots, $\mathcal{D}^{\mathcal{W}}_{m2}$ distance function. . . . 61
6.11 Computation time for ϕ_a for planar robots (a) and for 3D workspace robots (b). The roadmaps in (a) have 16 384 nodes. In (b), the roadmaps are constructed with the $\mathcal{D}^{\mathcal{W}}_{m2}$ distance function. 62
6.12 Compression results using run-length graph encoding for ϕ_n. The savings are in terms of the reduction of the number of labels. 63
6.13 Compression results for ϕ_n for planar robots (a, c, e) and robots with 3D workspace (b, d, f). The neighborhoods are region-growing in (a) and (b), "union" hierarchical in (c) and (d), and "best" hierarchical in (e) and (f). 65
6.14 Compression results for ϕ_a for planar robots: $\mathcal{D}^{\mathcal{W}}_{2}$ distance function for robots without joint limits (a, c, e) and $\mathcal{D}^{\mathcal{W}}_{m2}$ distance function for robots with joint limits (b, d, f). The neighborhoods are region-growing in (a) and (b), "union" hierarchical in (c) and (d), and "best" hierarchical in (e) and (f). 66
6.15 Compression results for ϕ_a for robots with 3D workspace: roadmap constructed with the $\mathcal{D}^{\mathcal{W}}_{m2}$ distance function (a, c, e), and roadmap constructed with the $\mathcal{D}^{\mathcal{W}}_{\mathcal{F}}$ distance function. The neighborhoods are region-growing in (a) and (b), "union" hierarchical in (c) and (d), and "best" hierarchical in (e) and (f). 68
6.16 File sizes and compression ratios for ϕ_n and a 16 384 node roadmap with efficient label encodings and different neighborhood models. The results for planar robots are in (a) and (b), and for 3D workspace robots in (c) and (d). 69
6.17 File sizes and compression ratios with efficient label encodings and different neighborhood models for ϕ_a for a 16 384 node roadmap for planar robots without joint limits. 70
6.18 File sizes and compression ratios with efficient label encodings and different neighborhood models for ϕ_a for a 16 384 node roadmap for planar robots with joint limits. 71
6.19 Compressed file sizes for ϕ_a and a 16 384 node roadmap for 3D workspace robots. The roadmaps were constructed with the $\mathcal{D}^{\mathcal{W}}_{m2}$ distance function (a) and the $\mathcal{D}^{\mathcal{W}}_{\mathcal{F}}$ distance function (b). 72
6.20 Test for ϵ-robustness for roadmaps constructed with different distance functions: (a) planar robots and (b) 3D workspace robots. 73
6.21 Arcs added and resulting connectivity at different steps during processing for ϵ-robustness for 2048-node roadmaps constructed with the $\mathcal{D}^{\mathcal{W}}_{m2}$ distance functions: (a) and (b) planar robots, and (c) and (d) 3D workspace robots. 74
6.22 An example plan for a 20-joint planar robot with joint limits. The sequence of steps is from left to right, top to bottom. 81
6.23 An example plan for a six-joint robot with a 3D workspace. The sequence of steps is from left to right, top to bottom. 83
6.24 An example plan for a 15-joint mobile robot. The sequence of steps is from left to right, top to bottom. 84

A.1 Number of samples of configuration space generated per node accepted. 88

A.2 Time in seconds to compute the k nearest neighbors for the different distance functions. . 89
A.3 Size of ϕ_a. 91
A.4 Computation time for ϕ_a for two distance functions. 92
A.5 Compression ratios. 92
A.6 $|\phi_n(w)|$ per cell for different robots and a graph with 2048 nodes. 94
A.7 $|\phi_a(w)|$ per cell for a two-link robot. 95
A.8 Plans for two robots passing through a (relatively) narrow corridor. The dark blocks are the obstacles. 96
A.9 A problem with the straight-line in configuration space local planner combined with the $\mathcal{D}_2^{\mathcal{W}}$ or the $\mathcal{D}_\infty^{\mathcal{W}}$ distance functions. 97

B.1 Three example robots with 15 joints: (a) planar robot, (b) 3D workspace robot class A, (c) 3D workspace robot class B. 99
B.2 Special case for planar robots: two joints. 99
B.3 Planar robot, 3–8 joints. 101
B.4 Planar robot, 9–14 joints. 102
B.5 Planar robot, 15–20 joints. 103
B.6 Robot class A with 3D workspace. 104
B.7 Robot class B with 3D workspace. 105

LIST OF SYMBOLS

$\mathcal{C}$	The configuration space of a robot
$\mathcal{CB}$	The obstacle region of $\mathcal{C}$
$\mathcal{C}_{free}$	The collision-free region of $\mathcal{C}$
$\mathcal{A}$	The robot
$\mathcal{W}$	The workspace of a robot
w	A cell in the workspace
$\mathcal{G}$	The graph (roadmap) in $\mathcal{C}$
$\mathcal{G}_n$	The nodes of $\mathcal{G}$
$\mathcal{G}_a$	The arcs of $\mathcal{G}$
q	A configuration, also a node in $\mathcal{G}$
γ	An arc in $\mathcal{G}$
ϕ	The mapping from $\mathcal{W}$ to $\mathcal{C}$
ϕ_n	The mapping from $\mathcal{W}$ to $\mathcal{G}_n$
ϕ_a	The mapping from $\mathcal{W}$ to $\mathcal{G}_a$
η	A neighborhood in $\mathcal{W}$
$\mathcal{D}_2^{\mathcal{C}}$, $\mathcal{D}_\infty^{\mathcal{C}}$	Distance functions based in $\mathcal{C}$
$\mathcal{D}_2^{\mathcal{W}}$, $\mathcal{D}_\infty^{\mathcal{W}}$	Distance functions based in $\mathcal{W}$
$\mathcal{D}_{m2}^{\mathcal{W}}$	An extension to $\mathcal{D}_2^{\mathcal{W}}$ incorporating the path midpoint
$\mathcal{D}_{\mathcal{F}}^{\mathcal{W}}$	A distance function based on the end-effector coordinate frame in $\mathcal{W}$

CHAPTER 1

INTRODUCTION

One of the main problems in robotics is that of geometric path planning. For this problem, we are given as input a geometric description of the robot and the obstacles in its environment. We are also given a pair of configurations of the robot, one of which is the initial configuration and the other is the goal. The task is to find a path connecting the two configurations such that the robot avoids the obstacles while transforming its state from the initial configuration to the goal configuration.

Path planning is one of the requirements for creating robots that are autonomous. Path planning can be as simple as moving a polygonal object in a static polygonal workspace, such as pushing a cart through an office environment. It also can be arbitrarily complex, as in the case of a robot consisting of many segments, some of which may be connected in loops whereas others may be flexible, operating in an unknown environment with possibly many moving obstacles, where the positions and velocities of the objects may not be known with high precision.

This thesis addresses the need for practical methods for geometric robot path planning in changing environments. In particular, we introduce a framework for allowing a path planner to adapt to changes in its environment while maintaining a quick response to planning queries. In many ways, our approach represents a logical next step in the evolution of path planning algorithms.

The earliest work in path planning produced exact algorithms (for example, [36, 112]) and methods that build an approximate representation of the full volume of configuration space (for example, [35,69,93]). In the former case, the best known algorithms have exponential complexity and require exact descriptions of both the robot and its environment, whereas in the latter case, the size of the representation of configuration space grows exponentially in the dimension of the configuration space.

The fact that real robots rarely have an exact description of the environment, coupled with a desire for real-time planning, led to the development of potential field approaches [66,77,81]. The idea of potential field approaches is to construct a scalar function over the configuration space that represents the goal region as the global minimum in the field and the obstacles as local maxima. Path planning is then reduced to following the gradient of the potential function until the goal is reached. The advantage of this

approach is that the potential functions are easy to compute, making the planner fast. Unfortunately, it is difficult to create potential function with a single global minimum at the goal; therefore, these planners are easily trapped by local minima.

The problems of local minima in potential field planners led to the development of randomized planning [17]. In this approach, when a local minimum is detected, a random motion is performed to try to escape the local minimum. Planning then can be considered a graph search, where the nodes of the graph are the sequence of local minima encountered when searching for the goal.

The randomized motion planners proved effective for a large range of problems, but required extensive computation time for some robots in certain environments [63,76]. This limitation, together with the idea that a robot will operate in the same environment for a long period of time, led to the development of the probabilistic roadmap planners [7,60,74,103,122]. The idea that the cost of planning will be amortized over many planning episodes justifies spending extensive amounts of time during a preprocessing stage, provided the resulting representation can be used to generate plans very quickly during a query stage.

The two stages of the probabilistic roadmap planners can be described as follows. In the preprocessing stage, a roadmap is constructed in the free configuration space. The nodes of the roadmap are created by some random sampling scheme, and pairs of nodes are interconnected using a simple local planner. After construction, this roadmap may contain more than one connected component, in which case an enhancement operation may be performed to try to connect the different components together. In the second stage, planning queries are performed. For each planning query, the initial and goal configurations are connected to the network and the network is searched for a path.

The PRM planners that use this two-stage processing mechanism are targeted toward environments in which the obstacles are stationary and whose positions are known in advance. For environments for which this assumption does not hold, *single-query* variants were developed that build the roadmap as they search for a path [63,82,118].

Our new approach is a direct descendant of the probabilistic roadmap methods. Our goal, like theirs, is a real-time planner that uses approximate representations such as those provided by computer vision or range sensors. However, unlike probabilistic roadmap methods, our method is aimed at changing environments and therefore cannot exploit the premise that planning will occur many times in the same environment. Note that we are not concerned here with the problem of motion planning among moving obstacles whose trajectories are known a priori. We are interested in motion planning for environments in which obstacle locations are unknown a priori, or in which obstacle motion is unpredictable, such as would be the case for a mobile robot equipped with a manipulator operating in a previously unknown environment.

The development of our ideas has been driven by the recent and continuing explosion of available computational power, both increased processor speeds and increased memory sizes. It is not, of course,

reasonable to expect algorithms with exponential space or time requirements to become feasible merely by increasing computational power, but it is quite reasonable now to encode fairly fine discretizations of the environment (which is always 3D) and to encode a representation of the mapping between workspace obstacles and obstacle regions in the configuration space, provided the configuration space is represented by a combinatorial data structure that does not grow exponentially with the dimension of the configuration space (for example, the graph representations used by probabilistic roadmap methods). These newly realized abilities form the basis for our new approach.

Our method begins, as do the probabilistic roadmap planners, by constructing a graph that represents the configuration space. Nodes are generated by a random sampling scheme, and connections between nodes are generated using a simple, straight-line planner. Unlike the probabilistic roadmap planners, we generate a roadmap that corresponds to an obstacle-free environment. Then, in a second phase of the preprocessing stage, we generate a representation that encodes the mapping from cells in the discretized workspace to nodes and arcs in the graph. These two phases are specific to the robot, but are independent of the environment in which the robot will operate. The fact that the preprocessing is completely independent of the robot's target environment removes constraints on preprocessing time. Indeed, with our approach, it is feasible that when a new robot is designed, an extended period of preprocessing could be performed, at the end of which the robot would essentially be preprogrammed to construct motion plans in any environment that it might encounter.

In the online planning phase, the planner first identifies the cells in the discretized workspace that correspond to obstacles, and then deletes the corresponding nodes and arcs from the graph, using this encoded mapping. Planning is then reduced to connecting the initial and final configurations to the graph (again, as is also the case with the probabilistic roadmap planners), and then searching the graph for a path between these newly added nodes. Of course it is possible to add obstacles to the environment in such a way that the graph becomes disconnected. This is true for any of the probabilistic roadmap planners (once one knows how samples are selected, and how these samples are connected by local planners, it is fairly straightforward to construct environments that will thwart them), but, as we will describe in subsequent chapters, there are a number of steps that can be taken to cope with this problem during the preprocessing stages.

In the on-line planning stage, our method runs in real time; plans are generated in less than 1 s. Thus, it is feasible to use the planner even in the case when obstacles are moving in the environment with unknown trajectories, provided a sensing system can identify in real time those regions of the workspace that are occupied by the obstacles.

There have been a number of previous approaches to path planning in changing environments. In some cases, planners have execution times that make it feasible to directly use them in some kinds of changing environments with no modifications. This is the case, for example, for the Ariadne's Clew

algorithm reported in [26,96]. The Ariadne's Clew algorithm operates by generating landmarks (during an exploration phase) and then connecting them to the existing network (the search phase). Variations of this algorithm can be obtained by varying the search phase, and by using different optimization criteria to select candidate landmarks [4,97]. The idea of incrementally expanding a network for single query planning has also been used in [63] and [82]. In both of these, networks are grown from both the initial and goal configurations until they can be connected, though the details for expanding the network differ. The incremental expansion method in [63] has also been used in a dynamic environment [79]. In [118] an adaptable approach that uses multiple local planners is described. At run time, characteristics of the problem are used to determine which (combination of) local planners will be most effective.

We also note here that in two of these previous approaches ([60] and [97]), the idea of somehow representing the mapping from the workspace to the configuration space was incorporated. In [60], during the off-line planning stage, the planner is aware of a set of obstacles that might be present in the environment (in their experiments, a single obstacle was used). The locations of these obstacles are specified a priori, and at run time the robot sensor system determines which, if any, of the obstacles is present in the environment. During the off-line planning stage, the trajectories in the paths tree that cause collisions with each of these obstacles are determined. When objects are detected at run time, the corresponding arcs are deleted from the graph. In [97], the paths tree (created by the modified Ariadne's Clew algorithm) is augmented to generate a graph. Then, for each path in the graph, the corresponding workspace cells are identified. Thus, when a new obstacle is added to the environment, the set of paths that intersect that obstacle can be deleted from the graph. Because these authors are primarily interested in domains for which extensive preprocessing is not viable, they construct relatively small graphs (fewer than 100 landmarks, with fewer than 400 corresponding paths). Therefore, it is fairly easy to add obstacles that would disconnect the graph, even though these obstacles might not cause the free configuration space to become disconnected. In their experimental evaluation of their planner, a single obstacle was added, and the addition of this obstacle resulted in disconnecting the graph into five components. In such cases, their algorithm resorts to the Ariadne's algorithm to reconnect the graph, which can take time long enough to prohibit the planner from being used in environments with moving obstacles.

In the remainder of this thesis, we discuss the features of our algorithm. We start with a discussion in Chapter 2 of the construction of the roadmap to be used for path planning, followed by a discussion of the construction of the mapping from the workspace to the roadmap in Chapter 3. We continue with a discussion of methods for enhancing the roadmap in Chapter 4 and how roadmaps we construct may be applied to mobile robots in Chapter 5. We then discuss some results using this approach with serial-link robots in 2D and 3D environments in Chapter 6, and finish with some concluding remarks in Chapter 7.

CHAPTER 2

CONSTRUCTING THE ROADMAP

Our construction of a roadmap of the configuration space is very similar to methods used in previous probabilistic roadmap (PRM) planners [74]. Nodes are generated by generating sample configurations, and these nodes are then connected to form a roadmap. We will denote this roadmap by $\mathcal{G} = \langle \mathcal{G}_n, \mathcal{G}_a \rangle$, in which $\mathcal{G}_n$ is the set of nodes in the roadmap and $\mathcal{G}_a$ is the set of arcs in the roadmap.

Generating samples to construct $\mathcal{G}_n$ is potentially more complex than for the traditional PRM planners, since we cannot exploit the geometry of the obstacle region in the configuration space $\mathcal{C}$ to guide the sampling of $\mathcal{C}$. In traditional PRM planners, the goal is to find enough collision-free samples of the configuration space so that the connectivity of collision-free portion of $\mathcal{C}$ can be captured by the roadmap. In our approach, all samples are collision free because there are no obstacles; therefore, our goal is to create samples such that our roadmap remains connected despite the motion of obstacles in the workspace.

Once the nodes have been generated, they must be connected to form the roadmap. There are two issues to be addressed in this phase of the roadmap construction. The first issue is to decide which nodes to attempt to connect together. For this purpose a distance function is defined that provides a measure of the expected difficulty that might be encountered when attempting the connection. The second issue to be addressed when constructing the roadmap is to check that a pair of nodes can be connected. This is the role of the local planner, which must test that a collision-free path exists joining the two nodes. Traditionally, this is the most expensive part of the PRM methods, as these paths involve many calls to collision testing libraries to determine the feasibility of the path. With our approach, connecting the nodes to generate $\mathcal{G}_a$ is simple, since our roadmap is constructed without the presence of obstacles in the workspace. For example, for holonomic robots, a simple straight-line planner could be used.

The remainder of this chapter is organized as follows. We begin with a discussion on the sampling of $\mathcal{C}$, including techniques others have used and those that are appropriate for our planner. We follow this with a discussion of the local planner that is used to connect pairs of configurations. Then, we discuss

the distance functions used to determine the pairs of nodes to connect together in the roadmap. We conclude with a discussion of nearest-neighbor algorithms and the efficiency of these algorithms.

2.1 Generating Sample Configurations

The first part of building a roadmap is to generate the nodes. We first discuss sampling methods used by other probabilistic roadmap methods, and then discuss our approaches.

2.1.1 Previous sampling approaches

For most other PRM methods, the roadmap is constructed in the presence of obstacles. The goal of sampling is then to generate sample configurations in the free configuration space (denoted $\mathcal{C}_{free}$). For this purpose, a variety of techniques have been developed. The simplest of these is to sample the configuration space uniformly at random, and discard the samples that lead to a collision. This technique makes no assumptions about the distributions of the obstacles and is relatively easy to analyze [72]. Unfortunately, the number of samples this technique places in any particular region of $\mathcal{C}_{free}$ is proportional to its volume; therefore, uniform sampling is unlikely to place samples in narrow passages. Most recent PRM sampling schemes have been developed to address this problem.

One means to address the problem of few samples in narrow passages in $\mathcal{C}_{free}$ is to add samples in regions where the roadmap has few nodes. More samples near these nodes can be taken by randomly bouncing off obstacles: choose a random direction, travel in the direction until an obstacle is encountered, choose another direction, and continue until the path length reaches some threshold [60, 75].

Another technique for sampling $\mathcal{C}_{free}$ is to concentrate the samples near the surfaces of the obstacles in configuration space. One such approach is to locate samples on the surfaces of the obstacles themselves. This approach works by taking an arbitrary sample in collision and then searching for the boundary of the collision region of $\mathcal{C}$ on rays directed away from the collision point, uniformly distributed on a hypersphere [9]. The hope is to locate the center point close to the center of the obstacle region such that the samples are uniformly distributed over the surface of the obstacle. A modified version of this approach also adds samples near the surfaces, in some cases building shells of samples around the $\mathcal{C}$-obstacles [7]. Another approach for concentrating the samples near the obstacle surfaces is to generate pairs of points, one uniform at random in $\mathcal{C}$ and the other a small distance away (with the distance guided by sampling a Gaussian distribution) [29]. In this case, the collision-free sample of the pair is added to the roadmap only if the second sample is in collision.

There are other techniques that use the geometry of the obstacles to define sample points. One such approach that works well for rigid-body robots in 2D environments is to use the geometry of the obstacles and the robot in the workspace to define the sample nodes in $\mathcal{C}_{free}$ [103]. In this case, the axis of the robot is placed parallel to obstacle surfaces with the robot a small distance away from the

surface (a similar position is defined for vertices). Another geometric approach is to generate samples along the medial axis, either in $\mathcal{C}_{free}$ or $\mathcal{W}$. To generate samples using the medial axis in $\mathcal{W}$, the idea is to first compute the medial axis of the workspace, and then take random configurations and move the robot from those configurations until some subset of reference points defined on the robot lie on (or as close as possible to) the medial axis [55, 58]. To generate samples in the medial axis in $\mathcal{C}_{free}$, the idea is to take random configurations and transform them to the medial axis [122]. Each random configuration falls into one of three cases. In the first case, the sample is on the medial axis and nothing further need be done. Second, the sample could be in the $\mathcal{C}_{free}$ but not on the medial axis, in which case the point is translated away from the nearest obstacle until it is equidistant from two obstacles. Third, the sample could be in an obstacle region. In this case, the configuration is translated to the nearest obstacle surface, and then to the medial axis.

Some techniques are designed in particular for single-query path planning. One of these involves choosing a node at random from the current roadmap, generating samples around that node, and adding some of these new samples to the roadmap [63]. In this case, the node selection is biased towards the nodes with fewer neighbors, and new samples that have too many neighbors in the roadmap are rejected. Another technique in a similar vein is to generate a sample at random of $\mathcal{C}$, find the node nearest to it in the roadmap, and generate a new node in the roadmap by moving toward the random sample [82]. This use of the random sample should bias the tree built with this approach to explore $\mathcal{C}_{free}$ more rapidly. The Ariadne's Clew algorithm also falls in this category, though it uses the endpoints of Manhattan paths as its samples [4–6]. Another approach for single-query path planning used a vision sensor to detect the free portions of the workspace, and a sampling algorithm that placed new nodes to explore further the free workspace [128].

Some sampling techniques for single-query planning have been developed to solve the more difficult kinodynamic planning problem, in which the velocities of the robot must also be taken into account in addition to the positions. In these cases, a tree or pair of trees is built in the state space of the robot, where each tree is incrementally expanded. As above in the case for holonomic robots, there are two techniques for expanding the trees. The first of these is to choose a random state from the state space of the robot, find the closest state in the tree built so far, and derive an input which will transform the state from the tree closer to the random state [85, 86]. The other approach is to choose a node from the tree with probability that is greater for nodes with fewer neighbors, and then select several random inputs to transform the state from the tree to several new states [79].

For certain types of planning problems, sampling is decomposed into two or more parts. For the case of flexible objects, either surfaces [59, 73] or volumes [10], the idea is to decompose the rigid object transformation from the parameters defining the deformation. The deformations are calculated separately from the transformation because they are calculated using energy minimization criteria, which

are expensive to compute. To generate samples, a deformation is determined at random, and this deformation is replicated at random locations in the workspace. Decomposition of the sampling process has also been used for robots with closed chains [56]. In this case, a roadmap is constructed for the internal motion of the robot, and this roadmap is replicated over the workspace, with parts that intersect obstacles deleted. The roadmap for the internal motion itself is decomposed into two parts: one for which configurations are generated at random, and the other for which inverse kinematics is used to generate valid configurations that maintain the closed chains. A different form of sampling was used in [87], also for robots with closed chains. In this case, a configuration was generated at random from the entire configuration space, and a randomized gradient descent was used to modify the configuration until it satisfied the closure constraint.

Some iterative forms of sampling have also been proposed. One of these was designed to allow a human to demonstrate a path for a 3D rigid object, one that may violate some collision constraints [18]. In this case, the planner samples the path provided by the user and modifies it until it satisfies the collision constraints. A different iterative sampling approach that pushes a roadmap into $\mathcal{C}_{free}$ was proposed in [62]. In this case, the idea is to sample an expanded $\mathcal{C}_{free}$, where the robot was allowed to penetrate the obstacles to a prespecified distance and then iteratively refine the resulting roadmap until it is entirely in $\mathcal{C}_{free}$.

2.1.2 Our sampling schemes

For our planner, since there are no obstacles to consider, it is fairly easy to generate samples in the configuration space. The only hard constraint is that self-collision (that is, collision between distinct links of the robot) is prohibited. One method for generating configuration nodes is to sample from a uniform distribution on the configuration space. This approach reflects a complete absence of prior knowledge about the environment in which the robot will ultimately operate. If prior knowledge, either about the environment or the set of tasks that the robot will perform, were available, an appropriate importance sampling scheme, or even a deterministic scheme (if the existence of certain obstacles were known in advance) could be used.

Even without prior knowledge, there are ways to improve the quality of the set of sample configurations. We could, for example, attempt to maximize the portion of the workspace that is covered by the sampled configurations. In a similar spirit, we could attempt to maximize the minimum distance between samples as in [26]. In fact, each of these criteria can be posed as optimization problems with deterministic solutions; however, in each case, the complexity of the resulting problem would prohibit its direct solution.

In addition to uniform sampling of the configuration space of the robot, we have investigated an importance sampling scheme that is based on the manipulability measure associated with the manipulator

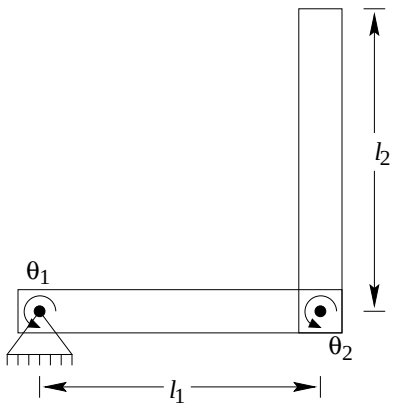


Figure 2.1 Planar robot example.

Jacobian matrix [127]. The manipulability is a quantitative measure of the ability of a robot to position and orient its end-effector. The formula for the manipulability is $\omega = \sqrt{\det J(\mathrm{q}) J^T(\mathrm{q})}$, where $J(\mathrm{q})$ is the relationship between the joint velocities and the end-effector velocities.

The basic idea for using the manipulability as a bias for sampling is the following. In regions of the configuration space where manipulability is high, the robot has great dexterity, and therefore relatively fewer samples should be required in these areas. Regions of the configuration space where manipulability is low tend to be near (or to include) singular configurations of the arm. Near singularities, the range of possible motions is reduced, and therefore such regions should be sampled more densely.

Consider the robot shown in Figure 2.1 as an example. The manipulability for this robot is $\omega = l_1 l_2 |\sin \theta_2|$, where l_1 and l_2 are the lengths of the two links [127]. The configuration shown in Figure 2.1 corresponds to one of the configurations at which the manipulability is highest for this robot (the other configuration has the second link pointing down). Notice that for this robot the manipulability does not depend on the position of the first joint.

Three different sample distributions for the robot are shown in Figure 2.2. As shown in the figure, concentrating the sampling in regions of low manipulability results in more samples near $\theta_2 = -\pi$, 0, and π at the bottom, middle, and top of the views of configuration space, respectively; sampling in regions of high manipulability results in more samples near $\theta_2 = -\pi/2$ and $\pi/2$. Normalizing for the link lengths, the average manipulability of the samples for the uniform distribution is 0.65, the average for low manipulability is 0.49, and the average for high manipulability is 0.82.

We adopt the following strategy for concentrating samples in regions of low manipulability. First, for each robot, we compute a discrete representation of the cumulative distribution function of the manipulability. We create this distribution function using the following approach. First, we sample the configuration space of the robot uniformly at random and compute the manipulability for that configuration. We exclude from this computation configurations in which the robot collides with itself.

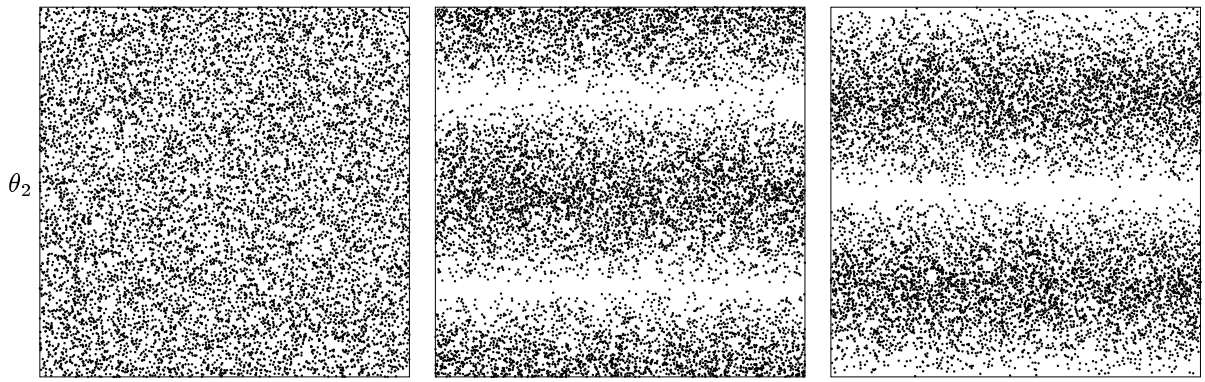


Figure 2.2 Sample distributions for a two-joint planar robot: (a) uniform, (b) higher density in regions of low manipulability, (c) higher density in regions of high manipulability.

We then create a histogram of the manipulability values that we have computed. Finally, we normalize the number in each bucket of the histogram and create the cumulative distribution function from these normalized values. The bucket size of the histogram and the number of samples to take are parameters.

Some example manipulability probability distributions for six-joint robots are shown in Figure 2.3. In this figure, the plots labeled "Not filtered" correspond to sampling the manipulability of the robot without filtering out samples in which the robot is in self-collision; the plots labeled "Filtered" do exclude these samples. In both cases and for both robots, 10 million samples were evaluated for manipulability. In addition, the gnuplot "csplines" function was used to smooth the plots. See Appendix B for more examples. An explanation for the shift that can be seen for both robots in the probability distribution when filtering out self collisions is that configurations in which the robot is in collision with itself tend to be configurations for which the manipulability is low.

We then use these manipulability distributions to guide the sampling as follows. First, we compute a self-collision free sample of the configuration space uniformly at random. We then compute the manipulability measure for this configuration and use this value as an index into the manipulability cumulative distribution table of the robot. The value in the table at this index is then used as the probability of rejection for this configuration. This approach was used to generate the sample distribution in Figure 2.2(b). In Figure 2.2(c), we use the table value as a probability of acceptance.

One shortcoming of the manipulability measure for our purposes is that it does not reflect joint limits. When the robot is near a joint limit, its movement is restricted. In an effort to include samples near joint limits we adopt the following convention: at configurations in which some joint is near a limit, the

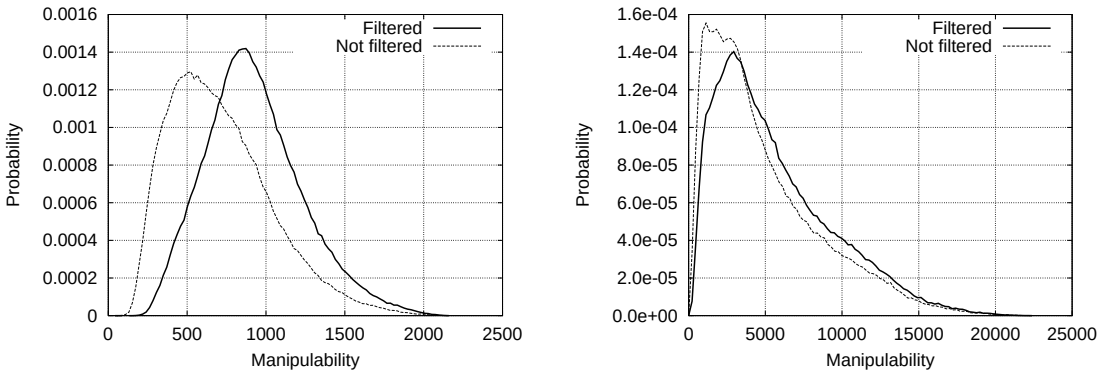


Figure 2.3 Probability distribution functions for the manipulability of a robot with six joints: (a) a planar robot and (b) a 3D workspace robot.

manipulability is defined to be zero. The nearness of a joint to its limit is a parameter of our sampling algorithm.

2.2 Local Planner

The local planner determines how the nodes of the roadmap are connected and whether the connection is feasible, given the presence of obstacles. In this section, we will discuss local planners used by traditional probabilistic roadmap methods, and follow with a discussion of the requirements for a local planner for our roadmap.

2.2.1 Local planners used in traditional PRM methods

The local planner used most often with the PRM methods for holonomic robots is the straight line in configuration space because of its speed, simplicity, and generality [8, 63, 72]. It will also work for any holonomic robot. For articulated robots, a variant of the straight-line planner is to have the robot follow sets of lines in the workspace [74]. In this case, the robot is partitioned into pairs of joints. The base of the first joint of each pair is translated on a straight line in the workspace, and the parameters of motion for each joint are determined by inverse kinematics to follow the straight lines.

Another general planner is based on Manhattan paths, in which one joint of the robot is moved at a time. Manhattan paths form the basis of the Ariadne's Clew algorithm because they are well suited to the genetic algorithms used to produce them [26]. A typical Manhattan path specifies several sequences of motion, in which each joint of the robot is moved in turn. In this version, the endpoint of the Manhattan path is the maximum distance traveled using the path until an obstacle is encountered or the end of the path is reached. To allow the entire path to be used, a "bouncing" technique was introduced, in which

the idea is to reverse direction along a path segment when an obstacle is encountered, and to continue to do the same for every collision until the endpoint of the path is reached. This method has been shown to allow the planner to navigate narrow passages in some cases, and to improve the overall success of the planner [4, 5, 97].

Some other general planners have used randomization techniques to generate paths. One such planner is one based on potential fields, though with limits on how long the planner is allowed to run [29, 103]. This planner will, in general, allow more nodes to be connected together, as it can generate paths that go around corners. Another randomized local planner used a randomized gradient descent technique to find paths for a robot with closure constraints [87].

Several PRM methods have also been developed for use with robots with kinematic and dynamic constraints, where, in general, a straight line path is not feasible. The earliest of these were applied to car-like robots operating in 2D environments [103, 104]. For these types of robots, configurations are connected using RTR paths (also TRT paths in [103]), where R denotes a turn and T denotes a straight-line translation and RTR specifies a path consisting of a turn, followed by a translation and another turn. For the turn part of the path, the robot turns at its minimum turning radius. These types of paths allow the planner to connect arbitrary configurations. A more general technique for robots with kinematic and dynamic constraints is to integrate the equations of motion for the robot with some input [79, 85, 86]. These planners operate in the state space of the robot, which includes the robot velocities in addition to the robot positions. A disadvantage to these latter approaches is that they tend to be restricted to single-query planning, as the tree built from one planning instance may be unusable for another planning instance.

Other PRM methods have separated the local planner into two or more parts. This is the approach used for the planners for flexible objects, such as flexible surfaces [59, 73] and flexible volumes [10]. For these methods, the rigid body motion (for [73] and [10], the rigid motion is defined by manipulation constraints on two opposite sides of the object) of the object is separated from the deformation. The local planner first generates a linear interpolation of the rigid body motion, and then generates a linear interpolation of the deformation. The primary reason for this separation is that the deformation is much more expensive to compute, and, when a collision is found during the rigid body transformation, it is not necessary to compute the deformation.

2.2.2 Local planner for our roadmap

In our approach, the roadmap is built without obstacles in the workspace. Therefore, the requirements for the local planner are very modest. It should be reasonably fast, as that reduces the time needed to construct the data structures, but this is not a primary concern. It must always return the same path when given the same two configurations as input. This is required because we compute a discrete

approximation to the volume swept by the robot as it traverses this path; this swept volume is stored implicitly as part of the mapping from the workspace to the roadmap. In addition, the local planner should consider self-collisions of the robot when determining whether two nodes can be connected.

Other objectives for the local planner could also be considered. In our case, the primary objective of interest is that the planner should generate paths that cover the least possible amount of the workspace when connecting two configurations. This has two purposes. First, reducing the volume of the workspace covered by the path reduces the likelihood that the path will be blocked when an obstacle is introduced into the workspace. Second, since we store the swept-volumes of the paths, this results in a savings in the size of the resulting data structure. Other, control-oriented, objectives can also be considered.

In our implementation, we use the simple straight line planner (that is, for each arc, a straight line trajectory in the configuration space is used to connect the two configurations whose connection is implied by that arc). In addition, using the straight-line planner simplifies the calculation of some of the distance functions described below.

2.3 Distance Function

The purpose of the distance function is to provide a measure of the difficulty the local planner would have in connecting two nodes in the roadmap. This measure of difficulty is then used to select which pairs of nodes to connect together in the roadmap. In some cases, the distance measure is also used to determine the order in which connections are attempted. The distance functions may be defined in either the configuration space or in the workspace. Distance functions defined in the workspace first transform the configuration to coordinates in the workspace.

Since the role of the distance function is to parameterize the difficulty in connecting two configurations, an ideal distance function would be the swept volume in the workspace of the trajectory connecting the two configurations. The reason for this is that, intuitively, trajectories with larger swept volumes are more likely to be blocked by obstacles in the environment. Unfortunately, as noted by others [8, 76], this distance function is very expensive to compute; therefore, most PRM methods use approximations based solely on the two configurations that are to be connected.

Several distance functions have been defined on the configuration space of the robot. These distance functions typically view the configuration space as a Cartesian space and define the distance function accordingly. For example, the Euclidean distance, which is the 2-norm of the differences of each configuration variable, is used in a few planners [9,26,56]. The 1-norm has also been used [87]. A problem with these distance functions is that they weight the configuration parameters equally, when some parameters may have a larger effect than others. A solution to that problem is to add weights to the different configuration parameters, and this has been used in a number of planners [7, 28, 86, 104].

Workspace distance functions attempt to measure the motion of the robot in the workspace. One method that has been used for articulated robots is to take the 2-norm of the Euclidean distances between the joint positions in the workspace [74]. Two methods for rigid objects in 3D that were tested in [8] are the distance between the center of mass of the object at two configurations and the maximum distance between any vertex of the bounding box at one configuration and its corresponding vertex at the other configuration. For flexible surface objects, a workspace-defined distance function is the sum of the translation distance, scaled rotation, and maximum displacement of a control point [73]. Another distance function is defined as the sum of the distances between unit vectors of the coordinate frame of the end-effector of the robot. This distance function was used for manipulation planning [4] and for inverse kinematics of redundant robots [3]. Another variant on a workspace distance function is defined for nonholonomic robots in 2D workspaces [104]. In this approach, the distance function is defined as the length of the minimum RTR path connecting two configurations.

For some methods of constructing the roadmap, there is no explicit distance function. Some of these methods involve incremental expansion of the roadmap, and the distance function is implicit in how new samples are generated around a node. Connections are then attempted between all new samples and the chosen node. The tree expansion method described in [63] uses this approach. In addition, tree expansion methods that derive new nodes by integrating the motion equations with an input automatically connect the new nodes with the old, and no distance function is needed in this case [79]. Other methods that do not use a distance function when constructing the roadmap are those that attempt to connect every pair of nodes [61]. This approach is particularly useful when analyzing the performance of the roadmap [72].

For the initial evaluation of our approach, we selected four distance functions:

the 2-norm in C-space

$$\mathcal{D}_2^{\mathcal{C}}(\mathrm{q},\mathrm{q}') = \|\mathrm{q}' - \mathrm{q}\| = \left[\sum_{i=1}^{n}(q_i' - q_i)^2\right]^{\frac{1}{2}}$$

the ∞-norm in C-space

$$\mathcal{D}_\infty^{\mathcal{C}}(\mathrm{q},\mathrm{q}') = \max_n |q_i' - q_i|$$

the 2-norm in workspace

$$\mathcal{D}_2^{\mathcal{W}}(\mathrm{q},\mathrm{q}') = \left[\sum_{\mathrm{p}\in\mathcal{A}} \big\|\mathrm{p}(\mathrm{q}') - \mathrm{p}(\mathrm{q})\big\|^2\right]^{\frac{1}{2}}$$

the ∞-norm in workspace

$$\mathcal{D}_\infty^{\mathcal{W}}(\mathrm{q},\mathrm{q}') = \max_{\mathrm{p}\in\mathcal{A}} \big\|\mathrm{p}(\mathrm{q}') - \mathrm{p}(\mathrm{q})\big\|.$$

In each of these equations, the robot has n joints, q and q$'$ are the two configurations corresponding to different nodes in the roadmap, q_i refers to the configuration of the ith joint, and p(q) refers to the

workspace reference point p of the set of reference points $\mathcal{A}$ at configuration q. Versions of $\mathcal{D}_\infty^\mathcal{W}$ and $\mathcal{D}_2^\mathcal{W}$ were also used in [76].

Figure 2.4 shows the roadmaps that result from using the four distance functions with a collection of 50 nodes. As can be seen in the figure, each of the roadmaps is somewhat different from the others, particularly as the distance between nodes increases. The advantage of the roadmaps generated using the workspace distance function is that the links of the robot are relatively closer together in the workspace; therefore, the volume swept by the robot as it traverses a path is, on average, smaller than the volume swept by paths defined by the configuration-space distance functions.

Most of the distance functions defined above try to capture the cost of a connection by using a measure defined only on the endpoints of the path. While this allows the value of the distance function to be calculated quickly, it does not sufficiently penalize the motion of the robot as it follows the path generated by the local planner. An example of this is shown in Figure 2.5, where Figure 2.5(a) shows two configurations that are considered close by the $\mathcal{D}_2^\mathcal{W}$ function, and Figure 2.5(b) shows the region of the workspace swept out by the straight-line in $\mathcal{C}$ planner. For this robot, the first joint has unlimited range of motion, whereas the latter nine have limited range. To incorporate a measure of the motion of the robot into the distance function, we add the midpoint $\mathrm{q}_m = (\mathrm{q} + \mathrm{q}')/2$ of the path into the distance function. Specifically, we have evaluated two new distance functions. The first of these is similar to $\mathcal{D}_2^\mathcal{W}$ defined above but with the addition of the midpoint of the path:

$$\mathcal{D}_{m2}^\mathcal{W}(\mathrm{q}, \mathrm{q}') = \left[\sum_{\mathrm{p} \in \mathcal{A}} \left\| \mathrm{p}(\mathrm{q}') - \mathrm{p}(\mathrm{q}_m) \right\|^2 + \sum_{\mathrm{p} \in \mathcal{A}} \left\| \mathrm{p}(\mathrm{q}_m) - \mathrm{p}(\mathrm{q}) \right\|^2 \right]^{\frac{1}{2}}.$$

The second distance function that we will investigate is based on the coordinate frame of the end-effector. A form of this distance function was used in [3] for use in a motion-planning based approach to inverse kinematics. The idea for this function is to connect nodes of the roadmap with similar end-effector positions, which may then be used during path smoothing to limit the total motion of the end-effector when connecting two configurations. This function will be defined as follows. Let $\mathcal{F}_a$ and $\mathcal{F}_b$ denote two coordinate frames. The distance between these two frames is the sum of the distances between the unit vectors along the coordinate axes, that is,

$$d(\mathcal{F}_a, \mathcal{F}_b) = d_x + d_y + d_z,$$

where d_x, d_y, and d_z are the Euclidean distances between the unit vectors along the x, y, and z axes, respectively (see Figure 2.6). Using this definition, the distance between two configurations q and q′ is

$$\mathcal{D}_\mathcal{F}^\mathcal{W}(\mathrm{q}, \mathrm{q}') = d(\mathcal{F}(\mathrm{q}), \mathcal{F}(\mathrm{q}_m)) + d(\mathcal{F}(\mathrm{q}_m), \mathcal{F}(\mathrm{q}')),$$

where $\mathcal{F}(\mathrm{q})$ is the end-effector frame corresponding to the configuration q and $\mathrm{q}_m = (\mathrm{q} + \mathrm{q}')/2$.

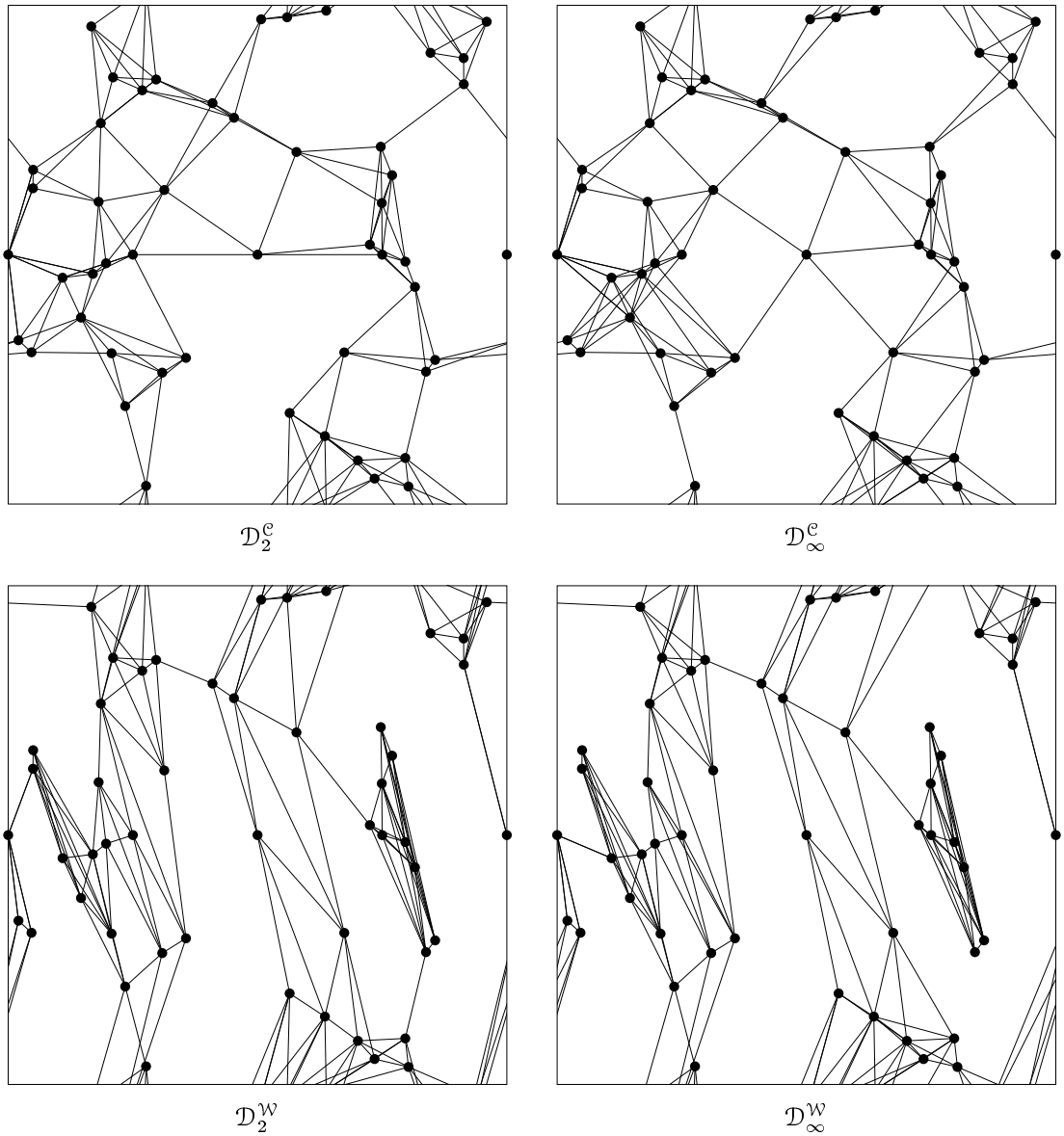
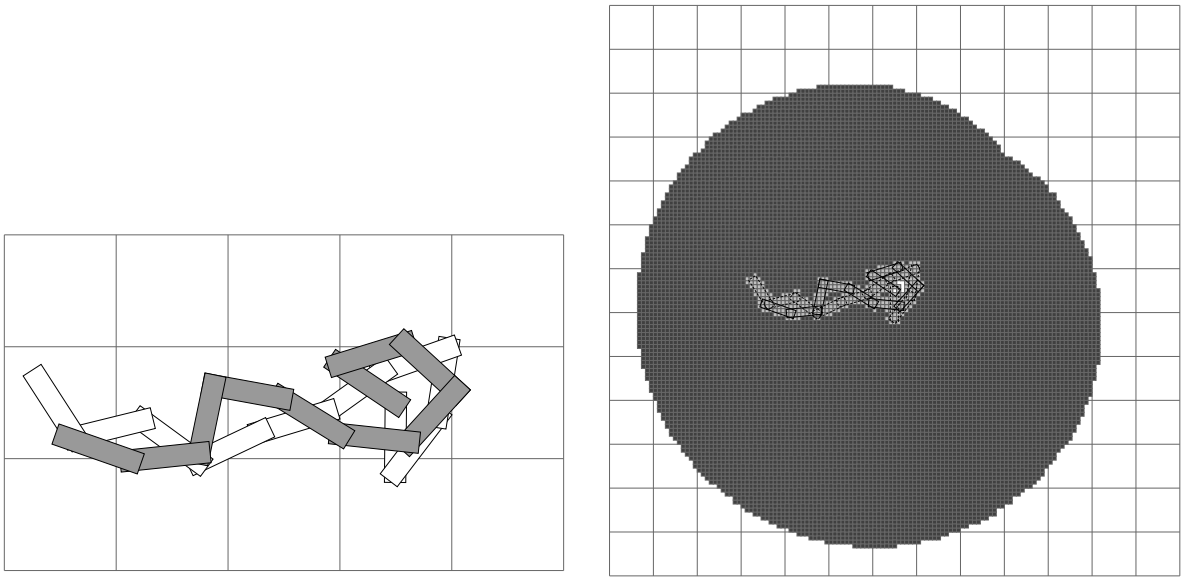


Figure 2.4 The roadmaps that result from using the four distance functions.



(a) (b)

Figure 2.5 (a) Two configurations considered close by the $\mathcal{D}_2^{\mathcal{W}}$ function and (b) the area swept by the robot following a straight-line in $\mathcal{C}$ connecting the two configurations.

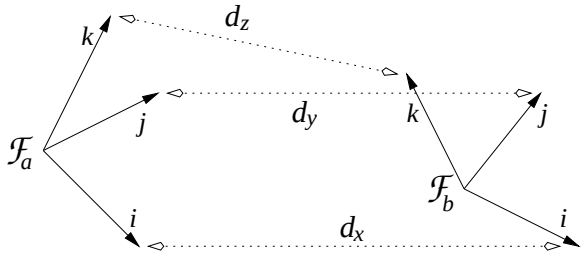


Figure 2.6 Illustration of d_x, d_y, and d_z.

It is important to note that, in general, neither of these two functions satisfies the triangle inequality. This restricts the type of nearest-neighbor algorithms that can be used when constructing the roadmap (see below) as well as its use for searching for paths in the planning phase. Also, because of the cost of computations associated with using the midpoint, using either of these two functions will increase the time needed to compute the nearest neighbors during roadmap construction, though this may lead to time savings elsewhere.

2.4 Efficient Nearest-Neighbor Computation

After generating the samples of the configuration space for the nodes of the roadmap, the set of nodes to which each node of the roadmap is connected must be determined. For this task, we compute for each node the nearest neighbors, with the distance between the nodes defined using one of the distance functions described above. The simplest approach for this task is to compute for each node the distance to the other nodes and keep the k nearest, where k is a parameter for roadmap construction. This approach has $O(dn^2)$ complexity, where n is the number of nodes and d is the dimension of the configuration space. This approach has the advantage that it works for arbitrary distance functions that can be computed in $O(d)$ time.

The version of nearest-neighbor algorithms described above is the all-k nearest neighbor problem in computational geometry. There are several approaches that reduce the complexity for this computation. One is a multidimensional divide-and-conquer technique with computation time $O(n \log^{d-1} n)$ [20]. Using quadtree-like approaches, this bound was improved by Clarkson [40] to $O(n \log n)$ and by Vaidya [117] to $O(3^d n \log n)$. Although these algorithms have good performance in terms of n, the exponential dependence on the dimension of space for the other factors in the computation time limits the usefulness of these approaches.

The all-k nearest neighbor problem can also be solved using a k nearest neighbor algorithm for each of the points, provided that the sum of the preprocessing time and search time is less than using the brute-force technique. There has been extensive work to create faster algorithms for locating nearest neighbors for a given query point, particularly for database applications where there are large databases that do not fit in main memory [24, 70, 90, 107, 110, 121]. For these algorithms, computation cost is calculated in terms of the number of disk accesses.

More appropriate for our purposes are the algorithms designed to work on in-memory data. One simple approach is to break the space containing the points into a grid of uniform cells [21, 42, 129]. These algorithms work well for uniformly distributed points, where each cell contains a small number of the points. For these distributions, the expected search time is constant [23]. With a suitable definition of the cells, the size of the data structure can be linear in the number of points [23].

A number of approaches have been proposed that use as a basis the kd-tree, a version of a binary tree extended to higher dimensions. In the basic version of the kd-tree, an axis-aligned partition hyperplane for a node is chosen based on the median value of the coordinate with the largest range [50]. Other approaches have been proposed for choosing the splitting criterion, such as arbitrary, not necessarily axis-aligned hyperplanes [113]; random orthonormal basis vectors [126]; and a clustering technique [45]. The kd-tree has also been extended to handle semidynamic point sets [22] and to report nearest neighbors

incrementally [34]. Other tree-based approaches with higher branching factors have been proposed, such as such as quadtrees [49], multi-split tree [51], and depth-limited multi-branching factor tree [78].

Another approach to decomposing the space of points into regions is to use the points themselves as the basis for forming a partition tree [31–33, 48, 106, 116, 125]. The basic idea is to use one or more "sentinel" points for each internal node of the tree, and to classify the remaining points into two or more partitions based on their distance to the sentinel points. During search, the distance between the query point and the sentinel points is computed, and the next node in the tree to visit is selected based on these distances. An advantage of many of these approaches is that they can be applied to more general metrics than the standard Minkowski metrics. A difficulty with these approaches is the choice of the points to use as the sentinels, as this choice can have a significant impact on search performance [106].

A problem with most of the methods described above is that there are factors that are exponential in the dimension in either the query time or storage space. This has lead to the development of approximate search techniques, where instead of returning the exact nearest neighbor, a point is returned whose distance from the query point is no more than $1 + \epsilon$ times the distance to the nearest neighbor. Another justification for this approach is that, for some domains, the distance function itself is an approximation of the similarity between two points; therefore, an exact nearest neighbor algorithm is not necessary. As in the exact case, these algorithms have been based on trees [11, 14, 25, 38, 67, 68], neighborhood roadmaps [11, 12, 41], hashing [44, 54], and projections on random lines [80, 83].

Unfortunately, the worse-case analysis of many of these algorithms still have constant factors that are exponential in the dimension, though the algorithms tend to perform well in practice. Several attempts have been made to explain why the performance is better than expected. For example, taking into account boundary effects of the point set, the base of the exponent can be reduced [13]. Another analysis technique is to analyze the fractal dimension of the data [105]. For some data sets, this shows that the dimension of the data can be much lower than the embedding space.

Of the above algorithms, those that are of greatest interest to us are those that decompose the set of points into a tree whose size is linear in the number of points, as these are the algorithms that can be used to compute the nearest neighbors faster than the brute-force approach. The exponential dependence of some of these algorithms, however, means that, for robots with more than a certain number of joints, the brute force method will be faster. The exact number of joints for which the brute force method will be faster depends on the distance function and the particular nearest-neighbor algorithm.

2.4.1 Our approaches for nearest-neighbor computation

When constructing most of our roadmaps, we have used the brute-force approach of computing the distance between all pairs of nodes and keeping the k closest. One reason for using this approach is that

two of our distance functions do not satisfy the triangle inequality, a property that is required by the faster nearest-neighbor algorithms discussed above.

We also have developed another approximation technique for faster k nearest neighbor computation that uses two distance functions: a slow function that produces the desired result and a fast function that provides a suitable approximation to the desired result. The basic idea is to use the fast function to select a subset of the points and then apply the slow function to select the closest k of this subset. Take for example the workspace-based distance functions defined above that use reference points on each link of the robot, and consider the two versions, one that uses the midpoint (the slow function) and the other that does not (the fast function). Using these two functions, the alternative approach to nearest neighbors can be viewed as first selecting pairs of configurations where the reference points are close together, and then selecting from these pairs those where the midpoint is also close.

A side effect of this approach is that, if the fast distance function is a metric, then we can use the faster nearest-neighbor algorithms to compute the subset of points to test for each point. This approach can also be extended to much more expensive distance functions, such as using the swept volume. We have not implemented such an approach, however.

2.5 Summary

To construct our roadmap, we have examined several different approaches. First, for generating the nodes, we have tested uniform random sampling of the configuration space, rejecting samples for which the robot is in self-collision. We have also tested distributions that are biased toward configurations of low manipulability. Second, to select the the pairs of nodes to connect with an arc, we have evaluated six different distance functions. We have found that the distance functions that better approximate the workspace volume swept by the path connecting two configurations produce better results. Finally, we have also evaluated several techniques for computing nearest neighbors. More details about our experiments can be found in Chapter 6.

CHAPTER 3

WORKSPACE TO CONFIGURATION SPACE MAPPING

A key element of our path planning approach is a mapping from the workspace to the roadmap in configuration space. The basic idea is to be able to determine, for each obstacle in the workspace, those portions of the roadmap that are no longer valid due to the robot colliding with the obstacle. The portions of the roadmap that remain after this step are then used for path planning. To be able to adjust to changes in the environment in real time, the mapping must be simple to evaluate, so that the roadmap can be updated as the environment changes. In this chapter, we discuss how the mapping is computed, and how it may be efficiently represented.

To represent the workspace, we use a uniform, rectangular decomposition, which we denote by $\mathcal{W}$. We denote the configuration space by $\mathcal{C}$, and define the mapping $\phi : \mathcal{W} \rightarrow \mathcal{C}$ as

$$\phi(w) = \{\mathrm{q} \mid \mathcal{A}(\mathrm{q}) \cap w \neq \emptyset\}, \tag{3.1}$$

in which w is cell of the workspace and $\mathcal{A}(\mathrm{q})$ denotes the subset of $\mathcal{W}$ occupied by the robot at configuration q. We note that $\phi(w)$ is exactly the configuration space obstacle region (often denoted by $\mathcal{CB}$) if w is considered as a polyhedral obstacle in the workspace. In our approach, we do not explicitly represent the configuration space, but instead use the roadmap $\mathcal{G}$. Therefore, we define two additional mappings, one from the workspace to the nodes in the roadmap, and one from the workspace to the arcs in the roadmap:

$$\phi_n(w) = \{\mathrm{q} \in \mathcal{G}_n \mid \mathcal{A}(\mathrm{q}) \cap w \neq \emptyset\}, \tag{3.2}$$

$$\phi_a(w) = \{\gamma \in \mathcal{G}_a \mid \mathcal{A}(\mathrm{q}) \cap w \neq \emptyset \text{ for some } \mathrm{q} \in \gamma\}. \tag{3.3}$$

An example of this mapping is shown in Figure 3.1 for a two-link robot in a 2D workspace. Shown in Figure 3.1(a) is the robot in its workspace, with an obstacle (1×1 dark square) located at $(0, 10)$ with respect to the origin of the robot (each link of the robot is 15 units long and 2.1 units wide). Shown in Figure 3.1(b) is the configuration space for this robot. The configuration of the robot shown corresponds to the dot at the center of the configuration space view. The roadmap was generated using the $\mathcal{D}_2^{\mathcal{W}}$ distance function.

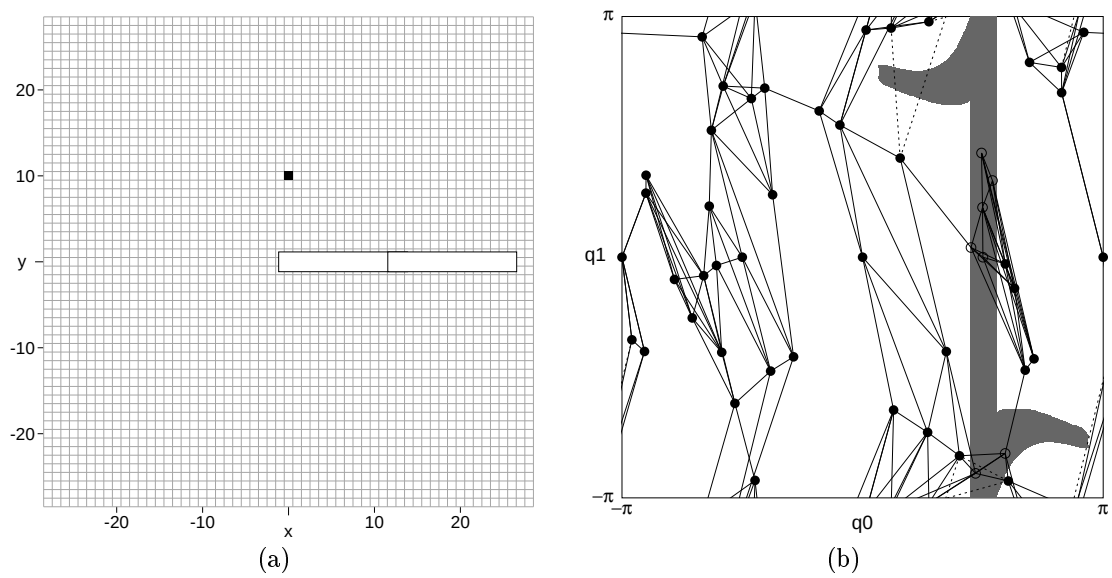


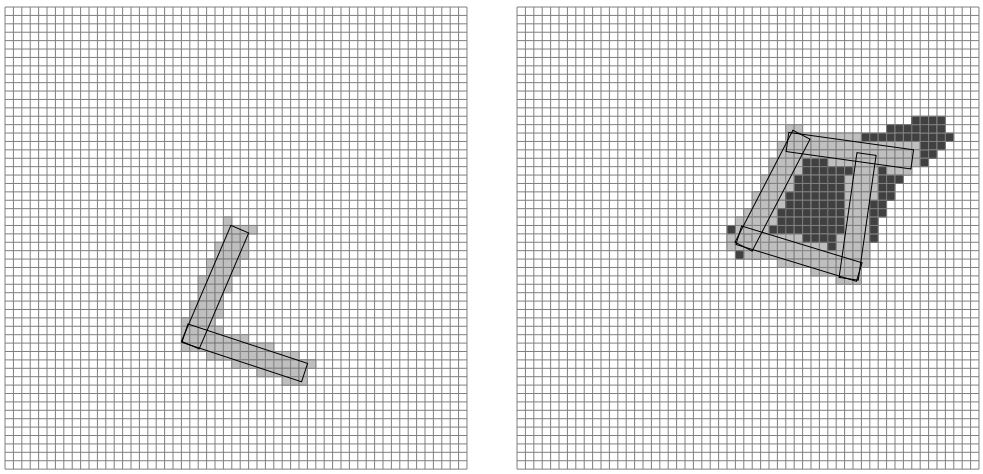
Figure 3.1 (a) The workspace of a two-link robot with an obstacle at (0, 10), and (b) the corresponding configuration space of the robot.

The shaded region in Figure 3.1 is the $\mathcal{CB}$ region created by the obstacle in the workspace. The mapping $\phi_n(w)$ for this obstacle w includes the nodes of the roadmap shown with empty circles, and $\phi_a(w)$ includes the arcs shown as dotted lines. Note that we need not include the arcs connected to the nodes in $\phi_n(w)$ in the representation of $\phi_a(w)$, because having one of the endpoints of the arc blocked by an obstacle means that the arc itself is also blocked. For this reason, the arcs connected to nodes in ϕ_n are not shown as dotted lines in Figure 3.1.

3.1 Computation of ϕ_n and ϕ_a

In terms of the implementation, it is much easier to compute the inverse maps ϕ_n^{-1} and ϕ_a^{-1}, and, for that reason, we use the inverse maps for the construction of our representation of the mapping. The inverse map $\phi_n^{-1}(\mathrm{q})$ corresponds to the set of cells in $\mathcal{W}$ that intersect with the robot at configuration q, and the map $\phi_a^{-1}(\gamma)$ corresponds to the cells in $\mathcal{W}$ that intersect with the swept volume of the robot as it follows the path γ in the roadmap. In our definition of $\phi_a^{-1}(\gamma)$, we exclude the cells in ϕ_n^{-1} for both of the endpoints of γ. An example of $\phi_n^{-1}(\mathrm{q})$ is shown in Figure 3.2(a) and $\phi_a^{-1}(\gamma)$ is shown in Figure 3.2(b). The lightly shaded cells correspond to those in ϕ_n^{-1} and the darkly shaded cells to those in ϕ_a^{-1}.

The construction of the representation for ϕ_n is straightforward. For each $\mathrm{q} \in \mathcal{G}_n$ we note the mapping from each $w \in \phi_n^{-1}(\mathrm{q})$ to the corresponding q. The set of cells in $\phi_n^{-1}(\mathrm{q})$ is computed by a voxelization algorithm inspired by the voxelization algorithm for polyhedra described in [71]. The vox-



(a) (b)

Figure 3.2 An example of (a) $\phi_n^{-1}(\mathrm{q})$ and (b) $\phi_a^{-1}(\gamma)$.

elization algorithm used for computer graphics cannot be used directly in our case because, for collision avoidance, we require all voxels that intersect with the robot to be in the voxelization output, whereas the requirements for rendering in computer graphics are less strict. We also require this algorithm to be efficient, as it is used very often: once for each convex polyhedron in the geometric description of the robot at each configuration in the roadmap. It is also used as a basis in computing ϕ_a (see below).

The algorithm, which is designed for convex polyhedra, is shown in Figure 3.3. The first step of the algorithm is to determine an axis-aligned bounding box for the polyhedron. Assume, without loss of generality, that the smallest dimension of this bounding box is aligned with the Z-axis and that the largest dimension is aligned with the X-axis. The next step is to cut the polyhedron into slices along the Z-axis, using the cell boundaries as the cutting planes. These slices can be computed incrementally, by starting with the vertex having the minimum Z coordinate, and then walking along the edges until the cutting planes are reached.

Now, for each of the slices along the Z-axis, we have a convex polyhedron that is one cell thick. Each of the vertices of these slices is then projected onto the XY-plane, and the planar convex hull of these points is computed [43]. This convex hull corresponds to the silhouette curve of the polyhedral slice, and this will enable us to determine all of the cells intersected by the polyhedron in that slice.

The next step of the algorithm is a 2D scan-conversion algorithm, similar to those used for rasterization of polygons in computer graphics [108]. The basic idea is to walk along the boundary of the polygon on both the left and right sides, starting at the lowest vertex, to locate the ends of runs of cells (pixels in the graphics case). We augment the standard algorithm so that for each row the minimum and maximum X coordinates of the polygon are calculated, and these values are used to determine the

Input: convex polyhedron $\mathcal{P}$
Output: $\{w | w \cap \mathcal{P} \neq \emptyset\}$

1. Compute the bounding box for the polyhedron
2. Cut polyhedron into slices along the Z-axis
3. For each slice z_i
4. Compute the set of vertices that fall within the slice
5. Compute the set of intersection points between line segments of the polyhedron and the planes that define the slice
6. Project the vertices and intersection points onto the XY-plane
7. Compute the 2D convex hull of the projected points
8. Cut convex hull into rows along the Y-axis
9. For each row y_i
10. Compute the set of vertices that fall within the row
11. Compute the set of intersection points between line segments of the convex hull and the lines that bound the row
12. Compute the minimum x_{min} and maximum x_{max} of the intersection points and vertices
13. Add the cells $(\lfloor x_{min} \rfloor, y_i, z_i)$ to $(\lceil x_{max} \rceil, y_i, z_i)$ to the output

Figure 3.3 Algorithm for computing the set of cells in $\phi_n^{-1}(\mathrm{q})$.

extent of the row of cells (this allows us to find all cells that intersect with the polyhedron along the row).

With the use of incremental algorithms to track both the line segments that cross the cutting planes that define each slice and the lines that bound each row, the running time of the algorithm is dominated by the time required to compute the 2D convex hulls of the projected points and the time to list each cell. This gives a running time that is $O(P|V| \log |V| + p)$, where P is the number of slices along the Z-axis, $|V|$ is the number of vertices in the polyhedron, $|V| \log |V|$ is for the computation of the convex hull, and p is the number of cells in the output.

The computation of ϕ_a^{-1} is more complex and time-consuming. It involves computing the swept-volume of the robot as it traverses a path computed by the local planner between two configurations. Computing the swept volume for a robot trajectory is not a trivial problem [1, 2, 27, 30, 92]. There exists a simple solution for convex polyhedra that only translate [124], but the general case for motion with rotation is much more complex. Because of this complexity, methods for approximating the swept volume have been proposed, such as a polyhedral approximation [1, 2], or B-splines in [92].

We have developed an approximate method for computing $\phi_a^{-1}(\gamma)$ for each $\gamma \in \mathcal{G}_a$ as follows. First, ϕ_n^{-1} for the two endpoints q_a and q_b of γ is computed. Then, the path corresponding to γ is sampled using a recursive bisection method, which proceeds as follows. First, the configuration q_m corresponding to the midpoint of the segment connecting the two nodes is computed, and $\phi_n^{-1}(\mathrm{q}_m)$ for that configuration is computed. Cells in $\phi_n^{-1}(\mathrm{q}_m)$ that are not in either $\phi_n^{-1}(\mathrm{q}_a)$ or $\phi_n^{-1}(\mathrm{q}_b)$ are added to $\phi_a^{-1}(\gamma)$. In the current implementation, the subdivision of the path continues until no new cells are added to $\phi_a^{-1}(\gamma)$ by the robot at configuration q_m. An example of this subdivision process is shown in Figure 3.4.

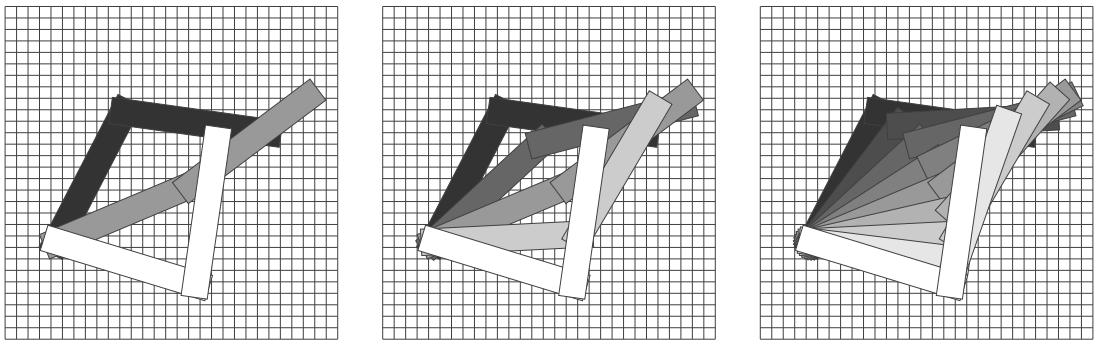


Figure 3.4 Illustration of the subdivision method for computing $\phi_a^{-1}(\gamma)$.

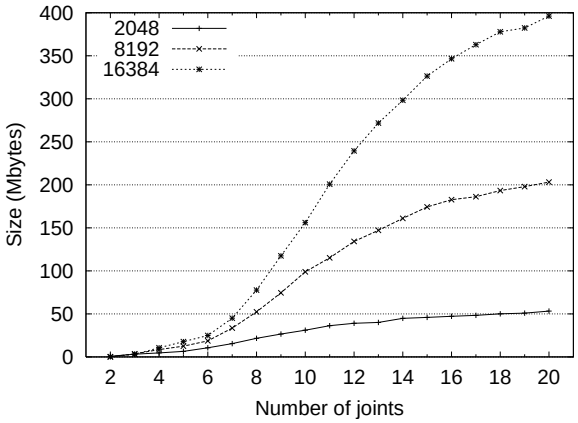


Figure 3.5 Size of ϕ_a using a naive encoding.

3.2 Efficient Representations

With the recent rapid increase in computer memories, it is feasible to use a naive encoding to represent ϕ_n and ϕ_a for a fairly large roadmap and a fairly fine discretization of the workspace. An example showing the size of a naive encoding of ϕ_a for several roadmap sizes and different numbers of joints for planar robots without joint limits is shown in Figure 3.5. Nevertheless, it is beneficial to find more efficient encodings of ϕ_a and ϕ_n, provided that the encoding does not drastically increase the computation required to compute plans online. Reducing the size of the representation will also enable us to consider larger roadmaps $\mathcal{G}$, which will increase the efficacy of the online planning.

From an information theoretic viewpoint, compression of a data set involves the reduction of redundancy in that data set. The amount of compression that can be performed is limited by the information content of the data set, which, in turn, is related to the degree of unexpectedness, or randomness, in the data set [57]. The limit of compression can also be viewed as the limit to which the data in the data set

can be generated using some model. In general, the more complex the model, the greater the amount of compression that can be achieved [19, 57]. Since the model must also be stored, there is a trade-off between the model complexity and compression result. There is also a trade-off between the complexity of the model and the time needed for compression and decompression. Concerning these trade-offs, we prefer algorithms that give the best compression with low model complexity and fast decode time.

In terms of compression efficiency, we already have available the optimal compression model: in the online phase, for each occupied cell in the workspace, compute $\mathcal{CB}$ and its intersection with the roadmap. Using this model, no storage space is required for either ϕ_a or ϕ_n. On the other hand, the online computation time required by this approach is prohibitive, so this model is of little use to us.

What remains for us to do is to derive models to exploit the redundancy available in ϕ_n and ϕ_a. There are three main sources of redundancy in the representation of ϕ_n and ϕ_a that can be exploited: (1) the spatial coherence of the set $\phi(w)$ in $\mathcal{C}$ for a specific w in $\mathcal{W}$, (2) spatial coherence of $\phi(w)$ for neighboring w's in $\mathcal{W}$, and (3) the representation of the labels of the nodes and arcs in the roadmap.

The spatial coherence of $\mathcal{CB}$ has been exploited in previous collision checking approaches (for example, [91, 99, 120]). In our case, spatial coherence derives from the continuity of ϕ, namely that small changes in the location of w will cause only small changes to $\phi(w)$. Because of this, for some cell, say $w^* \in \mathcal{W}$, we expect that $\phi(w)$ will be very similar to $\phi(w^*)$ for $w \in \eta(w^*)$, with $\eta(w^*)$ some appropriate neighborhood of w^*. We also expect that, because the robots we consider consist of a small set of convex polyhedra, for many $w \in \mathcal{W}$, $\mathcal{CB}(w)$ will contain a small number of connected components, and that these may be exploited to derive a compact representation of $\phi(w)$.

The discussion above suggests the following approach: partition $\mathcal{W}$ into a set of neighborhoods, and for each neighborhood (a) choose a representative w^*, (b) derive a compact representation of $\phi(w^*)$, and (c) for all $w \in \eta(w^*)$, express $\phi(w)$ in terms of $\phi(w^*)$. In some cases, we may be able to improve upon this by selecting some reference set in step (c) other than $\phi(w^*)$, and we discuss this below.

Given the above, we can formulate the corresponding optimization problem. For specific choice of neighborhood system we have the cost functional

$$\mathcal{L}(\mathcal{W}^*, \eta) = \sum_{w^* \in \mathcal{W}^*} \left\{ \text{cost}[\phi(w^*)] + \sum_{w \in \eta(w^*)} \text{cost}[\phi(w)] \right\}, \tag{3.4}$$

in which $\mathcal{W}^*$ is a set of representative cells in $\mathcal{W}$, $\eta(w^*)$ is the set of neighbor cells for w^*, and cost$[\phi(w)]$ denotes the cost of encoding the representation. This leads to the optimization problem

$$\begin{cases} \text{minimize} & \mathcal{L}(\mathcal{W}^*, \eta) \\ \text{subject to:} & \bigcup_{w^* \in \mathcal{W}^*} \eta(w^*) = \mathcal{W} \quad \text{and} \\ & \eta(w_i^*) \cap \eta(w_j^*) = \emptyset, i \neq j. \end{cases} \tag{3.5}$$

This particular formulation of the cost suggests an algorithm that first selects representatives in $\mathcal{W}$, builds the appropriate neighborhoods, and then efficiently encodes the representatives and the neighborhoods.

In the remainder of this section, we first discuss some previous compression approaches and the ideas we use in deriving our approach. We then discuss an approach to the encoding of $\phi_n(w^*)$. We continue with a discussion of the definition, construction, and encoding of neighborhood structures in $\mathcal{W}$. We conclude with a discussion on the encoding of the labels.

3.2.1 Evolution of data compression

The evolution of text compression is a good example of the extent to which good models increase the compressibility of text. The earliest models used the statistical distributions of the letters in the source text to compute representations for characters, such as the Huffman code [64], that minimize the number of bits needed to represent the text [19, 94]. Arithmetic coding is a more recent method for coding characters that can achieve much greater compression than Huffman coding, particularly when one of the characters occurs much more often than 50% in the text being coded (Huffman coding requires at least one bit for each symbol, whereas arithmetic coding can use less than one bit per symbol) [19]. There is a trade-off, though, in that a special additional code must be added to arithmetic coding to indicate that the end of the data stream has been reached.

Models were then developed to take into account the relationships between characters, such as the letter 'q' almost always being followed by the letter 'u' (at least in English). These models are combined with the encoders described above to generate the final result. These context-based models achieved better compression results, particularly as the length of the context increased [19]. The next step in compression was the introduction of adaptive models that could take into account the nature of the data being compressed to generate a model that can compress it better. The best known of these to date is the predictive partial match method, which builds context models of varying lengths as the input is examined, and encodes the current symbol with respect to the current context [19, 46]. Dictionary-based methods also achieve good compression results. In these methods, a dictionary is constructed of sequences of characters, and the compressed result includes pointers to elements in this dictionary [19, 84]. The Burrows-Wheeler transform, which rearranges the text to make it more compressible, is a more recent technique that also achieves good compression results [123].

A similar situation has also occurred for image compression, with more complex models being developed over time. Some of the earliest methods for image compression were developed for fax transmission. These methods included run-length encoding, in which a run of pixels of the same color are encoded as a pixel count and a value. These methods worked well for faxes, since faxes are generally only two-color

images [108, 123]. More recent standards for bilevel images, such as JBIG [123], have included more complex models based on the neighborhood of each pixel, with better compression as a result.

For gray-scale and color images, there are a number lossless compression techniques, including GIF, PNG, FELICS, CALIC, and JPEG-LS [37,123]. All of these methods use neighboring pixels to develop context models that improve compression. GIF and PNG both use variants of dictionary-based encoding on the pixels in scan-line order, with PNG being able to encode more colors and generally achieving better compression ratios than GIF. FELICS was developed for gray-scale images and uses two neighbors as the encoding context for each pixel. It can achieve better compression than either GIF or PNG on most of types of gray-scale images. CALIC is a context-adaptive method for image compression with better compression performance than most lossless image compression schemes to date (an algorithm called TMW achieves slightly better compression performance at a high computation cost [98]). JPEG-LS is a new standard for lossless compression of images that is simpler to implement than CALIC, but achieves nearly the same amount of compression. More image model examples include the piecewise-constant image model [15], targeted toward images based on color palettes, and the adaptive linear prediction and classification method [101], targeted toward gray-level images.

The MPEG standards play a similar role for lossy encoding of image sequences. MPEG attempts to exploit the similarity between adjacent images in a sequence to achieve high compression ratios for the entire sequence [100]. In MPEG-1, the motion of blocks of pixels between images of a sequence is computed, and a number of algorithms have been developed to efficiently compute this motion [52]. Later MPEG standards include higher-level models, such that, eventually, a video sequence may include just geometric models of the scene, the actors, and the motions of objects within the scene.

One of the key ideas used in image compression is the exploitation of the similarity of the image over small neighborhoods of pixels. Well known examples include using quadtrees [65, 109], run-length coding [108,109,123], and shaped groups of neighboring pixels [123] to compress images. In each case, the compression exploits an efficient representation of neighborhood structure and then encodes homogeneous neighborhoods. For quadtrees, neighborhoods are defined by recursive partitions of the image, and, in run-length coding, neighborhoods are defined in terms of individual horizontal rows in the image. For shaped groups of neighboring pixels, the neighborhoods are defined in terms of the number of neighboring pixels and the scanning order of the image.

For our compression approach, we borrow techniques from text encoding for the encoding of numbers and sorted sequences of numbers. From image encoding, we borrow ideas for the representation of neighborhoods. We also use the idea of differential encoding of neighbors from MPEG.

3.2.2 Efficient encoding of $\phi(w^*)$

As described above, a set of cells in the workspace are designated as representatives to be used for differential encoding of the neighbors. We have tested the following idea for compressing the representation of $\phi_n(w^*)$: for cells where $|\phi_n(w^*)|$ is relatively large, there is some graph traversal algorithm that efficiently separates the nodes inside $\phi_n(w^*)$ from those outside. In the ideal case, this graph traversal algorithm crosses from the nodes inside $\phi_n(w^*)$ to those outside only once; in this case, only the starting node and the point at which the boundary is crossed need be stored. For example, if $\phi(w^*)$ were a hypershere in an n-dimensional configuration space, then using a breadth-first search to trace the graph would require only the storage of the root node and a single integer that denotes the crossing point to encode ϕ_n. Using the same traversal algorithm starting at the specified start node can then reconstruct the exact set of nodes in $\mathcal{G}$ that correspond to the cell w^*. In our implementation, we have tested the depth-first and breadth-first search algorithms to evaluate the efficacy of this approach with our roadmaps.

This approach is analogous to run-length encoding. In run-length encoding, the lengths of strings of ones and zeros are stored (for binary images) [123]. This approach essentially works by imposing an ordering on the pixels in the image (raster scan ordering), and then encoding when a region of ones is exited or entered. In our approach, the graph traversal is used to impose an ordering on nodes and arcs in $\mathcal{G}$. We note here that approaches analogous to 2^n–trees are not appropriate in our case because these methods are very sensitive to small perturbations [65, 109, 111].

3.2.3 Encoding ϕ over neighborhoods in $\mathcal{W}$

The encoding of the neighborhoods is based on the idea discussed above that there will be only minor variations in ϕ over local neighborhoods of $\mathcal{W}$. Assuming for the moment that the neighborhoods have been determined, one way to encode ϕ over a neighborhood $\eta(w^*)$ is to first determine a representative $\phi^*(w^*)$ and to then specify $\phi(w)$ relative to $\phi^*(w^*)$ for each $w \in \eta(w^*)$. This is essentially a differential encoding and can be specified as $\phi'(w) = \phi(w) \oplus \phi^*(w^*)$, where $\phi'(w)$ is the encoded representation of $\phi(w)$ and $\oplus$ is the set symmetric difference operator.

We have investigated two methods for defining the neighborhoods in $\mathcal{W}$. The first method is a region-growing approach, in which seed cells are selected and regions are grown around them. The second method for defining neighborhoods is hierarchical, based on an octree decomposition of the workspace. Both of these will be discussed in more detail below.

In the region growing method for defining neighborhoods, we are confronted with two issues: how to choose seed cells, and when to stop expanding the neighborhood. Choosing the optimal seed cells is a difficult combinatorial optimization problem. Therefore, we have applied a greedy selection approach:

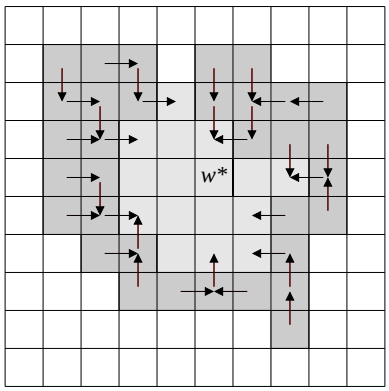


Figure 3.6 The two neighborhoods w^*: $\eta_1(w^*)$ is the light region containing w^* and $\eta_2(w^*)$ is the darker region.

at each iteration, the algorithm selects as the seed cell the unencoded workspace cell with the largest representation of $\phi(w)$.

We expand the neighborhood around the seed cell in two stages. Thus, for a seed cell w^*, we actually construct two neighborhoods, $\eta_1(w^*)$ and $\eta_2(w^*)$. As we discuss below, we use a different encoding method for each of these neighborhoods. An example of the neighborhood structure is shown in Figure 3.6; note that, though this example is 2D our algorithm works with 3D neighborhoods corresponding to the 3D workspace of the robot.

The neighborhood $\eta_1(w^*)$ is constructed as follows. Let $\phi^*_{n_1}$ be the representation that minimizes

$$\left|\phi^*_{n_1}\right| + \sum_{w \in \eta_1} \left|\phi(w) \oplus \phi^*_{n_1}\right|,$$

in which $|\cdot|$ denotes set cardinality. That is, $\phi^*_{n_1}$ is such that if every $w \in \eta_1(w^*)$ is encoded by a symmetric difference with $\phi^*_{n_1}$, then the cost of the representation of $\eta_1(w^*)$ is minimized. Beginning with the seed w^*, we add cells to η_1 until it becomes more efficient to encode the additional cells using the encoding method for η_2. Cells that are assigned to $\eta_1(w^*)$ are then encoded by their symmetric difference with $\phi^*_{n_1}$.

Each cell in $\eta_2(w^*)$ is encoded by the symmetric difference with the representation of one of its neighbors, w_N, that minimizes the resulting representation size. This is illustrated in Figure 3.6, where the arrows originating in the darkly colored cells point to the neighbor that is used for the differential encoding. Of course, care must be taken to avoid loops in this referencing scheme.

Construction of the neighborhood η_2 relies on two parameters: a recursion depth and a scale factor. The recursion depth limits the length of the chain of references between adjacent cells. For example, the longest such chain in Figure 3.6 is 4, with two cells having chains of this length. The scale factor is used to ensure that the encoding of the new cell is sufficiently small; that is, the cell w is added to η_2 if $|\phi(w) \oplus \phi(w_N)| < s|\phi(w)|$, where s is the scale factor. For cells where there is no 6-neighbor such that

$|\phi(w) \oplus \phi(w_N)| < s|\phi(w)|$, we add the cell to η_2 if $|\phi(w) \oplus \phi(w_N)|$ is the best we can do for this cell, provided $|\phi(w) \oplus \phi(w_N)| < |\phi(w)|$.

Cells where $|\phi(w)| \leq 2$ are treated separately. First, we cluster together arbitrary cells w_a and w_b $\mathcal{W}$ where $\phi(w_a) = \phi(w_b)$, creating one reference set for each cluster. For each of the remaining cells with $|\phi(w)| = 2$, each of its 6-neighbors is examined to find one where $|\phi(w) \oplus \phi(w_N)| = 1$. If such a cell w_N is found, it is used as the reference set for w. The remaining cells are encoded by themselves.

The second method for defining the neighborhoods in $\mathcal{W}$ is based on an octree-based decomposition of the workspace. The result is a hierarchical set of references for neighborhoods, at the bottom level, composed of eight cells. An advantage of defining the neighborhoods in this manner instead of using region growing is that the reference number for each cell is implicit with the location of the cell, and need not be stored. A disadvantage is that the region-growing method can define neighborhoods that more closely match the regions in the data.

Once the neighborhoods have been defined by the hierarchy, we are left with the choice of what set to store at the parent of each set of eight children in the hierarchy. To simplify the notation, we label the parent p and the children c_i. We have evaluated two such choices, which we label "union" and "best". In the union approach, the set $\phi^*(p)$ stored with the parent is $\cup_{i=1}^{8} \phi(c_i)$. This approach is equivalent to a multiresolution decomposition of the workspace. In the "best" approach, the set $\phi^*(p)$ is chosen to be the set that minimizes

$$\left|\phi^*(p)\right| + \sum_{i=1}^{8} \left|\phi(c_i) \oplus \phi^*(p)\right|.$$

For both choices of the parent set, the set encoded with each child is the set $\phi(c_i) \oplus \phi(p)$.

These two neighborhood models are options to use for the encoding of ϕ; the particular neighborhood model to use for a given ϕ is a choice of the user. We will show in Chapter 6 the effect of using these models for different robot examples.

3.2.4 Encoding the labels

Each entity in the representation of ϕ (including the nodes in ϕ_n, the arcs in ϕ_a, and the cells in $\mathcal{W}$) requires a label. For simplicity, we use nonnegative integers for the labels. Since there are many such labels, we can benefit from previous work that has been done in data compression, particularly in the area of efficiently encoding positive integers [19]. We can also impose some additional constraints on the labels, such as storing the lists of labels in sorted order. This allows us to use differential encoding of the labels, which can reduce the number of bits required to represent the list [123].

Given a finite set of integer labels and the frequency count for each label, one encoding procedure that one can use is that due to Huffman [64]. The advantage of this procedure is that the resulting code uses the fewest bits to encode the entire set of labels, assuming that no information about the

distribution of the labels is known other than their frequency. A disadvantage of the Huffman codes is the need to store the decoding table.

Extensions to Huffman codes have been proposed to allow them to represent infinite sets of positive integers. Each of these codes uses a variable-length representation, and each implies a different underlying probability distribution on the numbers being encoded. The simplest of these is the unary code, which grows in length by one bit for each increment in the magnitude of the number (for example, $0 \Rightarrow 0, 1 \Rightarrow 10, 2 \Rightarrow 110, 3 \Rightarrow 1110$, etc.). Such a code is suitable when the magnitudes of numbers to be encoded are all small.

A number of other encoding schemes for arbitrary positive integers are described in [19] and [47]. All of these codes combine a representation of the number of bits needed to encode a number, along with the bits of that encoding. One code is the γ code, which represents with a unary code the number of bits of the binary representation of the number with the leading 1 removed. The bits of this code alternate between bits encoding the length and bits of the number. A second code is the δ code, which uses the γ code to represent the length of the number, followed by the number in binary, also with the leading 1 removed. A third code, the ω code, uses a recursive encoding of the length of the number followed by the binary representation of the number. The recursion terminates with a two-bit code to represent the length. The ω' code is similar, except that the base case is three bits, and the number zero can be encoded. The ω'' code is an extension of the ω' code, starting the code with 1 instead of 0 and removing a redundant bit from the first four codes. Some examples of the γ, δ, ω, and ω'' codes are shown in Table 3.1, and the ω' code is shown in Table 3.2.

Yet another code for positive integers is the Golumb code [53]. This code was designed for sequences of numbers that have a geometric distribution. Its representation is derived as follows. For a parameter b, a number $n > 0$ is encoded in two parts: first, a unary code for $q + 1$, where $q = \lfloor (n-1)/b \rfloor$; then a binary code for the remainder $r = n - 1 - qb$. The remainder requires either $\lfloor \log b \rfloor$ or $\lceil \log b \rceil$ bits. For example, with $b = 7$, there are seven possible remainders, which can be encoded as 00, 010, 011, 100, 101, 110, 111. Examples of the Golumb code for $b = 11, 17$, and 23 are shown in Table 3.3.

There are two ways in which the Golumb codes can be used. One way is to compute the best b value globally over the set of labels to be encoded. A more space-efficient approach is to compute the b value individually for each cell. This has the added cost of storing the label count for each cell, though these are often required for other reasons, such as efficient decoding of the labels in each cell.

We have devised another encoding scheme, which we call Γ_b because of its similarity to the γ code described above. In this code, a number expressed in binary is split into pieces b bits in length. The number of pieces is encoded in unary, with one bit of the length appended to each piece. For example, to encode the number 11 using the Γ_3 code, first split the binary representation of 11 into two pieces, 001 and 011. To the first piece, prepend a 1 bit, and to the second, prepend a 0 bit. The resulting code is

Table 3.1 Examples of variable-length codes for integers. These codes do not include a representation of zero.

i	$\gamma(i)$	$\delta(i)$	$\omega(i)$	$\omega''(i)$
1	1	1	0	000
2	001	0010	100	001
3	011	0011	110	010
4	00001	01100	101000	011
5	00011	01101	101010	1000
6	01001	01110	101100	1010
7	01011	01111	101110	1100
8	0000001	00001000	1110000	1110
9	0000011	00001001	1110010	10010000
10	0001001	00001010	1110100	10010010
11	0001011	00001011	1110110	10010100
12	0100001	00001100	1111000	10010110
13	0100011	00001101	1111010	10011000
14	0101001	00001110	1111100	10011010
15	0101011	00001111	1111110	10011100
16	000000001	000110000	10100100000	10011110
17	000000011	000110001	10100100010	101100000
31	010101011	000111111	10100111110	101111100
32	00000000001	0100100000	101011000000	101111110
33	00000000011	0100100001	101011000010	1101000000
63	01010101011	0100111111	101011111110	1101111100
64	0000000000001	01011000000	1011010000000	1101111110
65	0000000000011	01011000001	1011010000010	11110000000
127	0101010101011	01011111111	1011011111110	11111111100
128	000000000000001	00000010000000	10111100000000	11111111110
129	000000000000011	00000010000001	10111100000010	1001000100000000
255	010101010101011	00000011111111	10111111111110	1001000111111100
256	00000000000000001	00000110000000	1110001000000000	1001000111111110
257	00000000000000011	00000110000001	1110001000000010	10010011000000000

10010011. Encoding and decoding the Γ_7 version of this code is particularly efficient on byte-addressable computers. Examples of the Γ_3 and the Γ_7 code are shown in Table 3.2.

We have also created an extension to the Γ_b code, which we call Γ'_b. This code treats the number zero as a special case, using just the bit 0 to encode it. Other numbers are encoded by prepending a 1 bit to the Γ_b code for the number less 1. As an example, consider encoding the number 11. Since 11 is not zero, the first bit of the code is a 1. We then append the Γ_b code for 10 to this, giving the resulting code 110010010. Examples of the Γ'_2 code are shown in Table 3.2. This code is particularly useful for sequences of numbers in which the number zero is predominant.

Shown in Table 3.4 are the number of bytes required using the different encoding schemes for the different numbers stored in ϕ. In this table, the columns labeled "Labels" correspond to the labels for either the nodes or arcs associated with each cell or reference set. The columns labeled "Count" refer to

Table 3.2 More examples of variable-length codes for integers. These codes do include a representation of zero.

i	$\omega'(i)$	$\Gamma_3(i)$	$\Gamma_7(i)$	$\Gamma'_2(i)$
0	0000	0000	00000000	0
1	0010	0001	00000001	1000
2	0100	0010	00000010	1001
3	0110	0011	00000011	1010
4	1000	0100	00000100	1011
5	1010	0101	00000101	1101000
6	1100	0110	00000110	1101001
7	1110	0111	00000111	1101010
8	10010000	10010000	00001000	1101011
9	10010010	10010001	00001001	1110000
10	10010100	10010010	00001010	1110001
11	10010110	10010011	00001011	1110010
12	10011000	10010100	00001100	1110011
13	10011010	10010101	00001101	1111000
14	10011100	10010110	00001110	1111001
15	10011110	10010111	00001111	1111010
16	101100000	10100000	00010000	1111011
17	101100010	10100001	00010001	1101100000
31	101111110	10110111	00011111	1101111010
32	1101000000	11000000	00100000	1101111011
33	1101000010	11000001	00100001	1110100000
63	1101111110	11110111	00111111	1111111010
64	11110000000	100110000000	01000000	1111111011
65	11110000010	100110000001	01000001	1101100100000
127	11111111110	100111110111	01111111	1101111111010
128	1001000100000000	101010000000	1000000100000000	1101111111011
129	1001000100000010	101010000001	1000000100000001	1110100100000
255	1001000111111110	101111110111	1000000101111111	1111111111010
256	10010011000000000	110010000000	1000001000000000	1111111111011
257	10010011000000010	110010000001	1000001000000001	1101100100100000

the integers that represent the number of labels associated with each cell or reference set. The columns labeled "Reference" refer to the integers that specify the reference set to be used with each cell. Last, the columns labeled "Length" refer to the integers used to specify the number of bytes in the encoded representation of each cell or reference set.

The row labeled "Huffman tree" is the cost of encoding the Huffman tree for the set of numbers, which includes two bits for each node in the tree plus the length of the Γ_3 encoding of each number represented in the tree. For both ϕ_n and ϕ_a in this example, the local Golumb code outperforms the Huffman code for encoding the labels. For encoding the reference numbers, the Γ_3 codes performed best.

Table 3.3 Golumb codes for $b = 11, 17$, and 23.

i	$b = 11$	$b = 17$	$b = 23$
1	0000	00000	00000
2	0001	00001	00001
3	0010	00010	00010
4	0011	00011	00011
5	0100	00100	00100
6	01010	00101	00101
7	01011	00110	00110
8	01100	00111	00111
9	01101	01000	01000
10	01110	01001	010010
11	01111	01010	010011
12	10000	01011	010100
13	10001	01100	010101
14	10010	01101	010110
15	10011	01110	010111
16	10100	011110	011000
17	101010	011111	011001
18	101011	100000	011010
31	1101101	101101	100111
32	1101110	101110	101000
33	1101111	1011110	1010010
63	111101100	1101011	11011001
64	111101101	1101100	11011010
65	111101110	1101101	11011011
127	111111111101010	11111100111	1111010100
128	111111111101011	11111101000	1111010101
129	111111111101100	11111101001	1111010110
255	11111111111111111111110001	1111111111111011111	111111111100001
256	11111111111111111111110010	1111111111111100000	111111111100010
257	11111111111111111111110011	1111111111111100001	111111111100011

3.2.5 Summary

To summarize, we have several options from which to choose when encoding the representation of ϕ for a particular roadmap. For the neighborhood type, we have the choice of none, region-growing, or hierarchical. We leave this choice to the user. For the encoding of the labels, we have chosen to use the local Golumb code, since this gives good compression results and reasonable decoding speed. For encoding the other data, we use the Γ'_b code for counts and lengths, and the Γ_b code for the reference numbers. The particular value of b chosen for each of these is the one that minimizes the the size of the representation; the value of b depends on the choice of the neighborhood model and the particular instance of the mapping from the workspace to the roadmap.

Table 3.4 Number of kilobytes required using the various codes for ϕ for a six-DOF robot example in a 3D environment. The roadmap has 2048 nodes and ϕ was encoded using the region-growing approach described in Section 3.2.3.

	ϕ_n				ϕ_a			
Code	Labels	Count	Reference	Length	Labels	Count	Reference	Length
Γ_1	828	56	266	91	6850	175	494	240
Γ_2	662	64	213	75	5395	174	404	201
Γ_3	614	78	198	82	4989	194	367	210
Γ_4	594	93	200	95	4851	222	381	232
Γ_5	615	110	201	110	4974	250	391	255
Γ_6	565	127	213	127	4835	281	421	283
Γ_7	639	146	216	146	4536	315	436	315
Γ_8	703	164	228	164	5071	354	460	354
Γ_9	729	182	232	182	5567	393	492	393
Γ'_1	868	48	280	93	7134	171	520	262
Γ'_2	703	48	228	88	5680	160	432	236
Γ'_3	655	53	214	99	5274	167	403	246
Γ'_4	634	57	216	113	5135	180	417	269
Γ'_5	656	63	218	128	5258	193	427	294
Γ'_6	606	70	230	146	5119	208	457	322
Γ'_7	680	77	233	164	4821	226	472	354
Γ'_8	744	84	244	182	5356	249	497	393
Γ'_9	769	92	248	200	5852	272	528	433
unary	43 541	59	26 200	103	1 033 146	324	88 871	455
δ	779	130	272	166	5987	325	536	397
γ	869	81	286	122	7135	231	541	310
ω	695	48	228	96	5412	166	423	260
ω'	733	80	231	85	5693	203	416	224
ω''	733	63	229	71	5692	173	408	198
Huffman	448	44	175	55	3591	148	331	168
Huffman tree	4	0.23	30	0.17	15	0.33	61	0.28
Golumb (global)	472				4434			
Golumb (local)	358				2643			

CHAPTER 4

ROADMAP ENHANCEMENTS

As we have discussed in Chapter 2, one of the difficulties with the traditional probabilistic roadmap planners is placing samples in narrow passages in $\mathcal{C}_{free}$. This led to the development of an enhancement phase, in which more sampling is performed to improve the connectivity of the roadmap. Since our roadmap is constructed without obstacles, reducing the number of connected components is not the goal for our enhancement phase. Instead, we are interested in preserving the connectivity of the roadmap when obstacles are added to the workspace.

We have examined two methods for evaluating the connectivity of our roadmap in the presence of obstacles, and from these evaluation methods we have derived mechanisms for enhancing the connectivity of our roadmap in the presence of these obstacles. The first evaluation method is based a traditional measure of graph connectivity: the minimum cut set. The second method derives a quantitative measure of the robustness of the roadmap based on the mapping from the workspace to the roadmap. We discuss both of these evaluation methods and the corresponding enhancement techniques in more detail in the following.

4.1 Previous Enhancement Methods

The roadmap enhancement methods that have been used in other PRM methods can be divided into two classes: true enhancement and incremental construction. In the true enhancement methods, the enhancement phase occurs after an initial roadmap is constructed but before any planning takes place that uses the roadmap. For these methods, the primary concern is to join together different connected components in the initial roadmap. In incremental construction, the roadmap is initialized with the start and goal configurations for each planning query, and the roadmap is expanded until a path is found connecting the start and goal (or a time limit is reached). We have already discussed the incremental construction methods in Section 2.1.1 and will not discuss them further here.

The basis of the true enhancement methods is the identification of "difficult" regions in the configuration space, where "difficult" regions are defined as those regions where the sample density is low, or

"narrow" regions between two or more connected components of the roadmap. Once these regions have been identified, one is selected at each step of the enhancement phase for the placement of new nodes.

The selection criterion used in [60] evaluates each small connected component of the roadmap in an attempt to connect it to the largest component. The node chosen for this connection attempt is the node closest to any node in the large component. The selection criterion used in [59, 74] weights each node of the roadmap with a value inversely proportional to the number of nodes to which it is connected. One of these nodes is then selected at random using the weights to bias the selection.

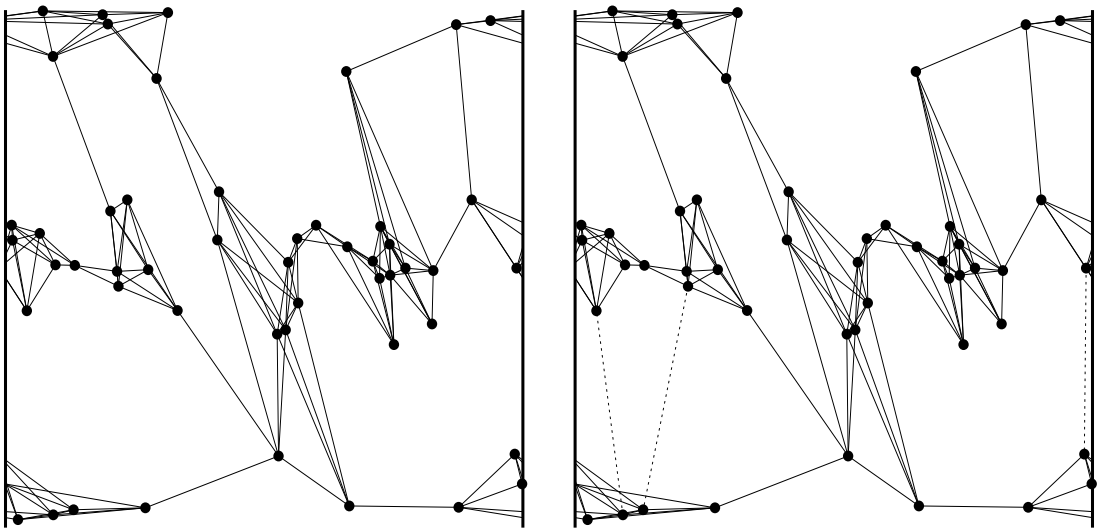
Two methods for resampling have been proposed: neighborhood importance sampling and random walk sampling. The simplest resampling method of the two is the neighborhood importance sampling. In this method, a neighborhood is constructed around a target node by selecting a small interval for each degree of freedom of the robot. After a new configuration is generated in this small region, the local planner is invoked to attempt to connect it to the rest of the roadmap. This resampling method was used in [74] for articulated robots. For these robots, the size of the interval used for resampling was different for each joint, with joints closer to the origin of the robot having smaller intervals than those close to the end. A difficulty with this approach is that there is no guarantee that the new node can be connected to the roadmap.

The random walk resampling method addresses this problem. This method is based on performing a random walk from the selected node and attempting to connect the endpoint of the walk to the roadmap. The random walk is performed as follows: Choose a direction at random and use the local planner to move in this direction until a collision occurs. At the collision point (or a small distance away from it), choose a new random direction and continue the walk until a stopping criterion is reached. The stopping criterion used in [60] is based on the assumption that $\mathcal{C}_{free}$ contains a single connected component: check each step in the walk to determine whether it can be connected to another component in the roadmap, and stop the walk once this happens. An iteration limit is used in [76] and [75].

The purpose of the enhancement methods described above is to improve the connectivity of the roadmap used for planning. We note that there are other possible definitions for enhancement. One such enhancement is described in [114], where a "super-graph" is constructed so that the roadmap can support planning for multiple robots. Each node of the super-graph corresponds to a feasible placement of the set of robots, and each edge corresponds to a feasible motion of one of the robots.

The resampling methods described above require knowledge about the distribution of obstacles in the environment and are designed to connect the various components of a disconnected roadmap. Since we build our roadmap without obstacles, the roadmap will always have a single connected component,[1] and therefore the traditional idea of roadmap enhancement does not apply. Also, unlike previous approaches,

[1] There is one caveat to this: if more than one sampling distribution is used to construct the roadmap, the roadmap may be constructed such that the sets of nodes from each distribution belong to different connected components. In this case, the enhancement phase is needed to ensure that these different components are connected together.



(a) (b)

Figure 4.1 A roadmap (a) before and (b) after enhancement. The dashed lines in (b) correspond to the new arcs.

we are concerned with maintaining the connectivity even if certain nodes or arcs are deleted from the roadmap. We can, however, exploit the mapping from the workspace to the configuration space (discussed in Chapter 3) to use as a basis for evaluating and improving the connectivity of the roadmap.

4.2 Connectivity-based Enhancement

Our first method for enhancement is to examine quantitatively the edge connectivity of the roadmap and add arcs in regions of low connectivity. The edge connectivity of the roadmap is defined as the minimum of the maximum number of arc-distinct paths (that is, paths with no arc in common) between any distinct pair of nodes. The basic idea is that, in regions of low connectivity, it is relatively easy to disconnect clusters of nodes in the roadmap with the addition of a small set of obstacles in the workspace. The goal is to have the edge connectivity of the roadmap be at least equal to the minimum degree of any node in the roadmap.

Some motivation for this can be seen Figure 4.1(a), which shows a 50-node roadmap for a two-joint planar robot. The first joint of the robot has no limit on its range of motion; hence, the left and right borders marked with thick lines in Figure 4.1 correspond to the same configuration. The second joint had limits on its range; these correspond to the top and bottom of the configuration space view. The roadmap was constructed so that each node is connected to at least five of its nearest neighbors. In Figure 4.1(a), there is one cluster of nodes at the bottom that is connected to the rest of the roadmap by only two arcs, and there are two clusters with four arcs connecting them to the rest of the roadmap.

Adding three arcs, as shown in the enhanced version in Figure 4.1(b), ensures that each cluster has at least five arcs connecting it to the rest of the roadmap.

Our approach for this form of enhancement is to add arcs to the roadmap until the edge connectivity of the roadmap is equal to the minimum node degree. To do this, we repeatedly apply the unweighted minimum cut set algorithm from [95] to find the minimum number of arcs that partition the roadmap into two pieces. We then add arcs between these two pieces until the number of such arcs is equal to the minimum node degree. We stop when one of the pieces consists of a single node. A property of the algorithm we use for computing the minimum cut set is that when the number of arcs in the minimum cut set is equal to the minimum node degree, one of the pieces returned by the algorithm consists of a single node.

Once we have found a partition of the roadmap, we use the following approach to add arcs between the two pieces of the partition. First, for each node q in the smaller of the two pieces, we compute the set of cells in $\mathcal{W}$ in $\phi_n^{-1}(\mathrm{q})$, using the inverse of the mapping ϕ_n described in Chapter 3. Next, for each arc γ of the arcs connecting the two pieces, we compute a subset set of cells in $\mathcal{W}$ in $\phi_a^{-1}(\gamma)$ which excludes those cells in $\phi_n^{-1}(\mathrm{q})$ for any node q in the smaller piece. The cells that remain in $\phi_a^{-1}(\gamma)$ for each arc γ connecting the two pieces form a set of potential obstacles in the workspace whose presence would disconnect the roadmap into these two pieces.

We then add arcs in a set of phases, until enough arcs have been added between the two pieces of the partition. For each phase, we select a set of cells from the subset of $\phi_a^{-1}(\gamma)$ described above for each of the arcs connecting the two pieces. These cells will be considered as being occupied by obstacles as we attempt to add more arcs connecting the two pieces. We then take each node of the smaller piece in random order and attempt to add an arc between it and some node in the larger piece. The reason for selecting the nodes of the smaller piece at random is to try to evenly distribute the new arcs over the nodes of the smaller piece. We evaluate each node of the larger piece in order of increasing distance from the test node in the smaller piece, until we succeed in adding an arc or until we have tested all arcs of the larger piece.

We choose the set of cells to mark as occupied in each phase as follows. In the first phase, we select all of the cells in the subset of $\phi_a^{-1}(\gamma)$ for each of the arcs connecting the two pieces. This is the most restrictive case, and arcs added in this phase allow the cluster to remain connected to the rest of the roadmap for the largest set of obstacles. In the next three phases, we select only one cell from each of the subsets of $\phi_a^{-1}(\gamma)$ for the arcs connecting the two pieces. First, we choose the cell closest to the robot base; second, the cell farthest from the robot base; and, finally, a cell at random.

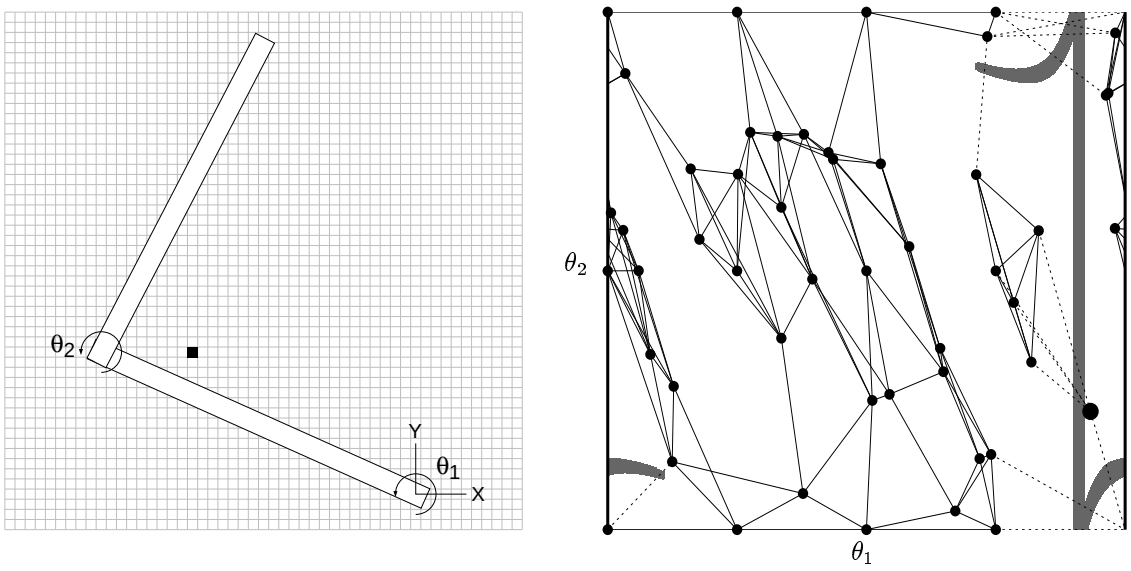


Figure 4.2 Effect of an occupied cell in the workspace on the connectivity of the roadmap. The configuration shown in the workspace corresponds to the enlarged node on the right side of configuration space.

4.3 Workspace-based Enhancement

The second method for roadmap enhancement examines quantitatively the relationship between the roadmap and the workspace. In particular, we derive a property that we call *ϵ-robustness* (inspired by the notion of ϵ-goodness described in [72]).[2] We say that $\mathcal{G}$ is ϵ-robust if no spherical obstacle in the workspace of diameter ϵ (or less) can cause $\mathcal{G}$ to become disconnected. Note that ϵ-robustness is a global property of the roadmap $\mathcal{G}$. Using this concept, it is possible to ensure that, after the enhancement phase, $\mathcal{G}$ can tolerate certain types of obstacles. In particular, for a specified value of ϵ, $\mathcal{G}$ can be tested for ϵ-robustness. This test can used to enumerate the specific workspace obstacles that violate the ϵ-robustness criterion and, for each such obstacle, the roadmap can be patched.

Some motivation for the ϵ-robustness idea can be seen in Figure 4.2, which shows the workspace and configuration space for a two-joint robot with a roadmap with 50 nodes. The first joint of the robot has no limit on its range of motion; hence, the left and right borders of the view of the configuration space correspond to the same configuration for the first joint. The second joint has limits on its range; these correspond to the top and bottom of the view of configuration space. In the figure, the dark cell in the workspace corresponds to the shaded region in configuration space. In this case, the obstacle

[2] Note that we also follow the use of ϵ in ϵ-goodness in that we wish to have the largest possible ϵ. This is contrary to the usual use of ϵ in algorithm design.

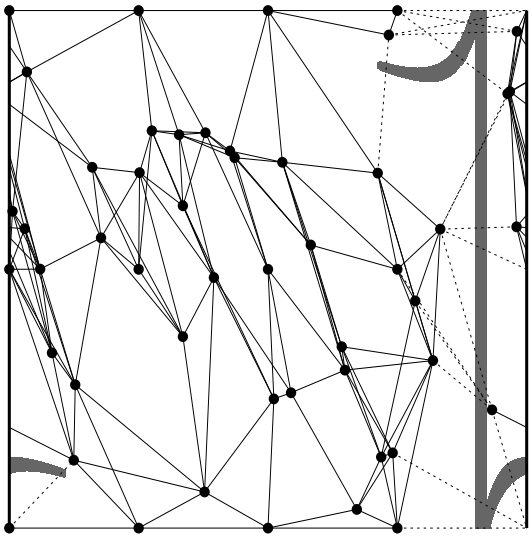


Figure 4.3 The result of ϵ-robustness processing on the roadmap shown in Figure 4.2. Note that the obstacle no longer breaks the roadmap.

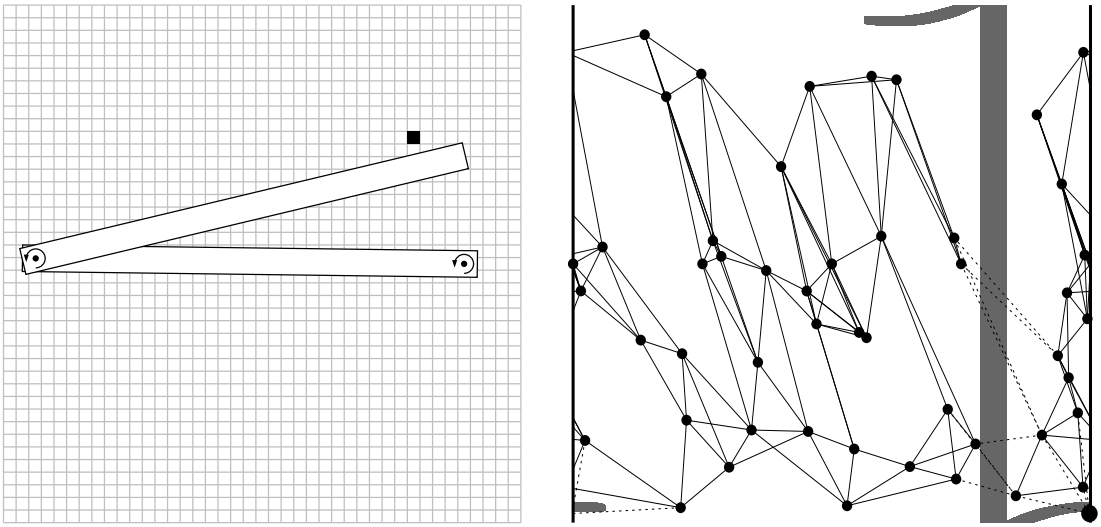
breaks the roadmap into three pieces. Hence, the roadmap in Figure 4.2 is not ϵ-robust for any value of ϵ. Figure 4.3 shows an enhanced version of the roadmap in Figure 4.2 (the enhancement caused 26 arcs to be added to the roadmap). This enhanced roadmap is ϵ-robust for $\epsilon \leq 4$. In other words, no single obstacle of size less than four units can cause the roadmap to become disconnected.

Our approach to enhancement is to add arcs to the roadmap until the desired level of ϵ-robustness is achieved (if possible). Our algorithm proceeds as follows. Starting with a cube of size $1 \times 1 \times 1$, sweep cube-shaped cells over the workspace of the robot. For each location of the cube, compute the set of connected connected components. If there is more than one component, add an arc, if possible, to repair the connectivity of the roadmap, using the local planner to verify the feasibility of each arc tested. Repeat the sweep with larger cubes, $2 \times 2 \times 2$, $3 \times 3 \times 3$, etc., up to a cube of side length $\lceil \epsilon + 1 \rceil$.

This algorithm is not complete, since the local planner may fail to find a path to connect two components. An example of this is shown in Figure 4.4, where the configuration shown in the workspace is isolated from the rest of the roadmap (recall that our example robot has joint limits).

4.3.1 Pathological cases

Unfortunately, it is not always possible to achieve ϵ-robustness, even for $\epsilon = 1$. For many robots (particularly those with joint limits), there are specific cells in the workspace such that if an obstacle is placed in one of these cells, $\mathcal{C}_{free}$ itself may become disconnected, or the passage connecting two regions in $\mathcal{C}_{free}$ may become sufficiently narrow that representing this passage in the roadmap becomes difficult. Our approach is to flag these cells in $\mathcal{W}$ and to eliminate them from consideration during the



Workspace Configuration space

Figure 4.4 An example where the local planner failed to reconnect the roadmap.

enhancement stage. The resulting roadmap will, of course, not be globally ϵ-robust, but will be as nearly ϵ-robust as possible.

Three cases that correspond to $\mathcal{C}_{free}$ becoming disconnected are shown in Figure 4.5. The cells that correspond to case I are those that split the range of the first joint into multiple disconnected pieces. The cell in the example in Figure 4.5 creates three such pieces. The cells that correspond to case II prevent a qualitative transition in the configuration of the robot. In the example, this prevents the robot from switching between the left-arm and right-arm configurations. The cells that correspond to case III are those that trap the robot near a joint limit. In the example shown, there are three components to $\mathcal{C}_{free}$: two small pockets and one large region.

Although the examples shown in Figure 4.5 are for robots with two joints, these same cases also occur for robots with more joints. In addition, if limits are placed on the motion of the first joint, then all cells that can be touched by the first link of the robot will disconnect the roadmap. Other cases are possible for robots with more joints. For example, the regions of $\mathcal{C}$ where the robot collides with itself combined with $\mathcal{CB}$ from a cell in the workspace can also yield isolated pockets of $\mathcal{C}_{free}$. Another example is shown in Figure 4.6. This is a case for planar robots for which the width of a passage can become arbitrarily narrow, such that it is difficult to represent the connectivity of $\mathcal{C}_{free}$ in the roadmap.

4.3.2 Application of ϵ-robustness to robot design

The concept of ϵ-robustness can also be used as an evaluation tool in robot design, with the goal of creating robots with fewer pathological obstacles in the workspace. As an example, consider the robot

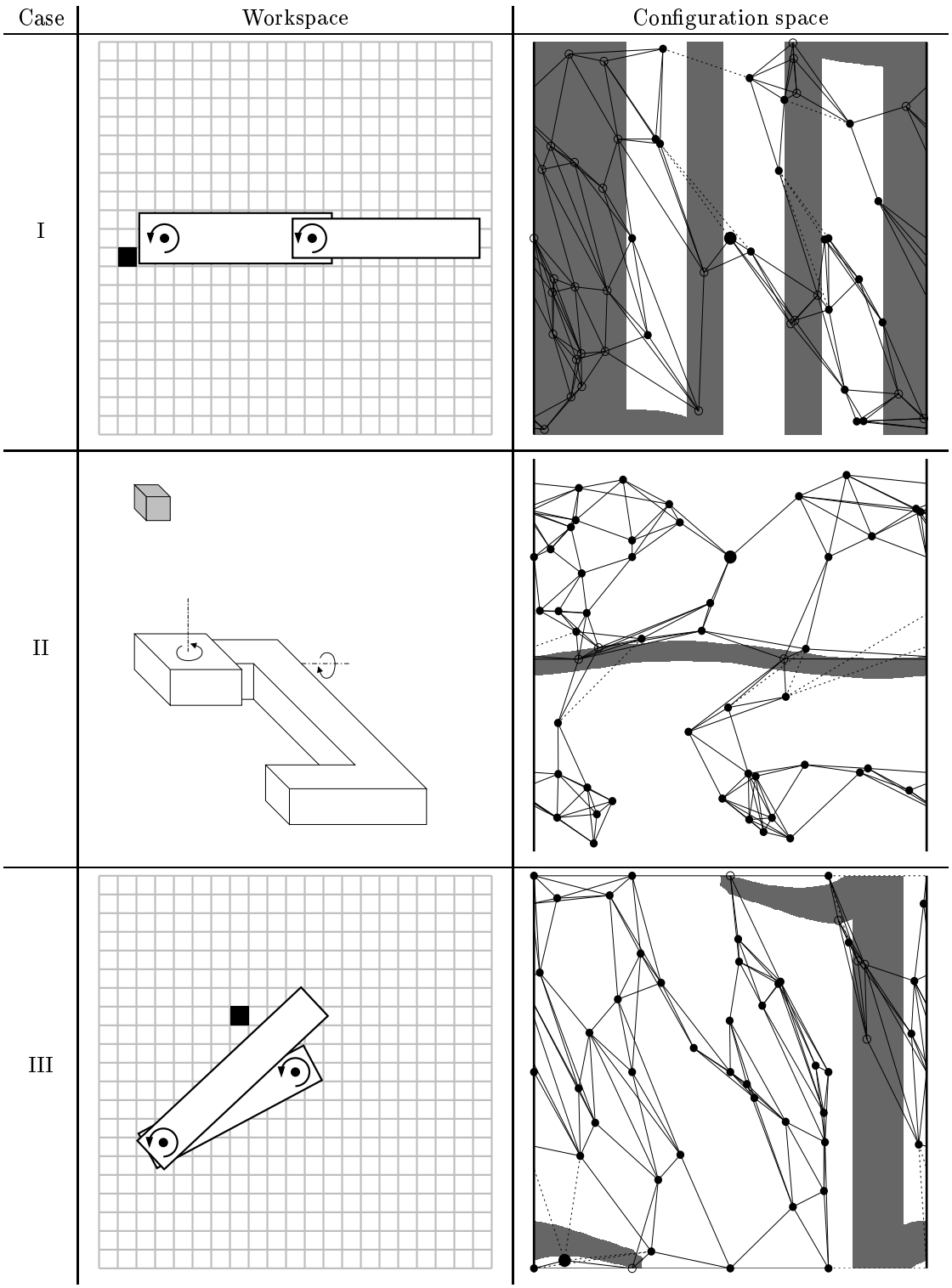


Figure 4.5 Three cases of cells that disconnect $\mathcal{C}_{free}$ and the corresponding $\mathcal{CB}$. The problem cell is shaded and the solid dots represent the rotation axes of the joints.

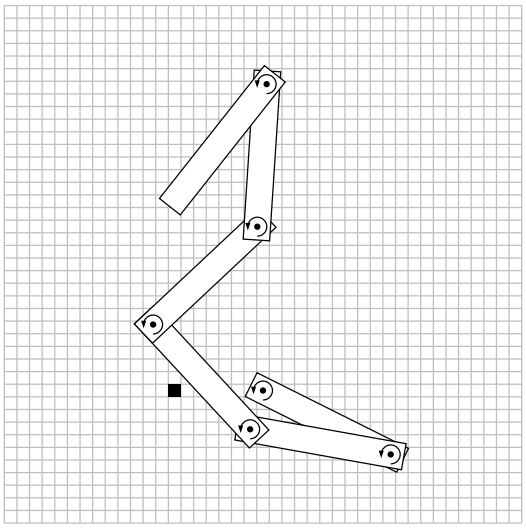


Figure 4.6 A problem cell for planar robots.

used for case II in Figure 4.5 and shown again in Figure 4.7(a). A view of its configuration space is shown in Figure 4.7(b), which also shows $\mathcal{CB}$ for a collection of seven cells in the workspace. Each of these cells generates a $\mathcal{C}$-obstacle that contains a pocket that is separated from the rest of $\mathcal{C}_{free}$, and some of these pockets trap nodes from a roadmap example. A modified version of the robot, shown in Figure 4.7(c), does not have these pockets. A view of its configuration space with the same set of obstacles is shown in Figure 4.7(d). An additional benefit of the modification to the robot is that no single cell in the workspace corresponds to case II in Figure 4.5 (a sufficiently large block will cause this problem, however).

Unfortunately, not all types of problem cells can be avoided. For example, given a sufficiently fine resolution in the workspace, there will always exist cells that correspond to case I in Figure 4.5, at least for robots with rectilinear geometry. Robots with more than two joints will also have problems with isolated regions in $\mathcal{C}_{free}$ resulting from the combination of the $\mathcal{CB}$ from some cells in the workspace, joint limits, and the self-collision region of $\mathcal{C}$.

4.3.3 Detection of pathological cases and other algorithm details

During processing, several types of "problem" cells are detected, many of which correspond to the cases shown in Figure 4.5. Once these cells are found at one level of processing, they are skipped in later stages. The first of these problem cells are those in which all nodes in the roadmap are blocked. These correspond to cells occupied by the robot in all of the configurations in the roadmap, that is, cells that intersect with the stationary portion of the robot. A second set of problem cells consists of cells that break the roadmap into multiple small pieces. These cells are detected by imposing a

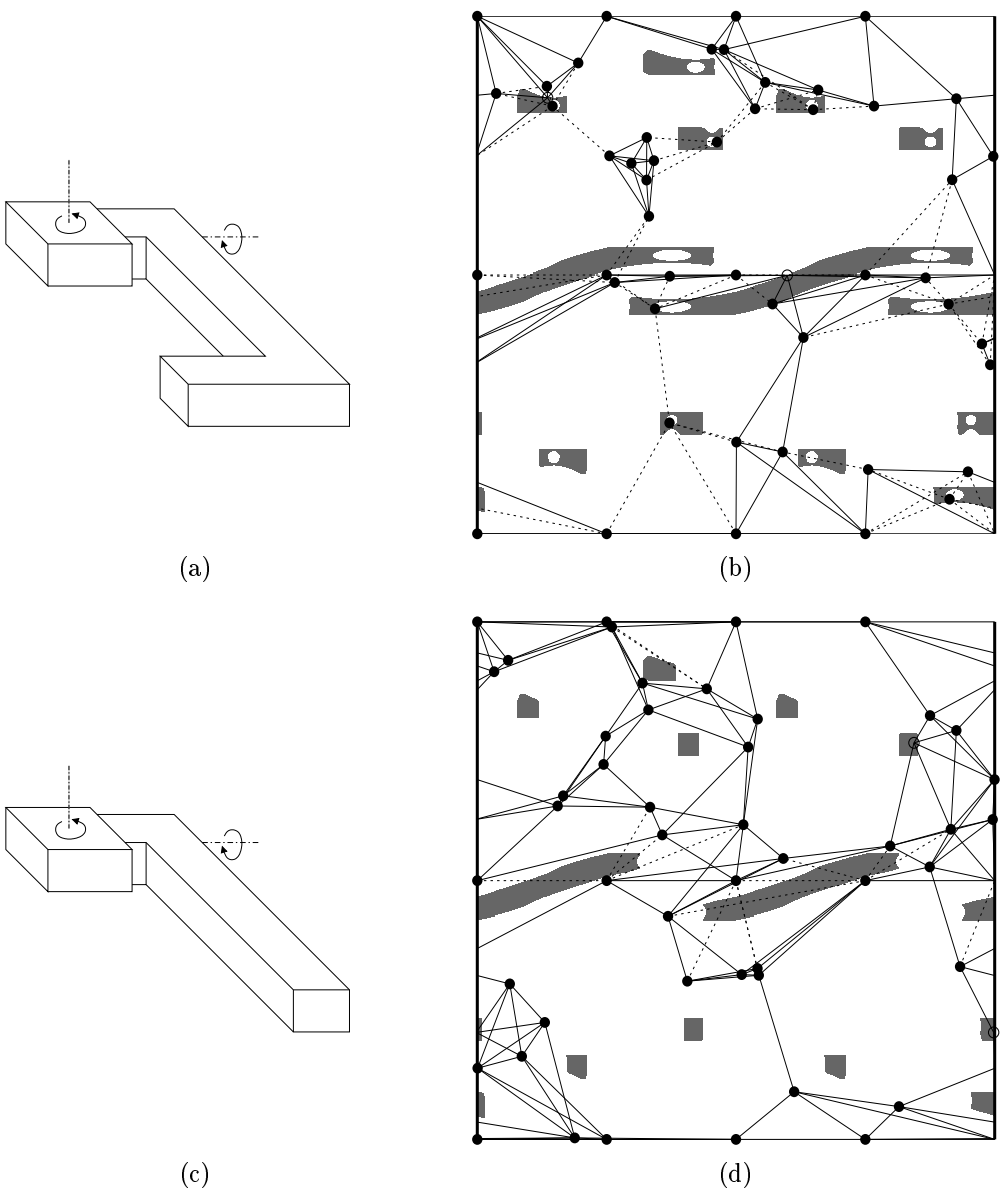


Figure 4.7 A robot with and a robot without some pathological cells in the workspace.

minimum size on the largest cluster, with the idea that, in general, cells that break the connectivity of the roadmap generate one large component and several smaller components. Cells that break the roadmap into multiple small pieces are those that block motion of the first joint of the robot, as in case I in Figure 4.5. The threshold we use is that the largest cluster should contain at least $\frac{1}{3}$ of the nodes in the roadmap. Because these two type of cells can be determined once the geometry of the robot has been specified, a list of these cells can be input as a parameter to skip processing them altogether.

A third class of problem cells consists of those in which the roadmap is split into two or more large pieces, where large is defined as a component having more than $\frac{1}{100}$ of the nodes in the roadmap. In this case, an attempt is made to connect the remaining small pieces to one of the large pieces, but not to connect the large components together. Cells or clusters of cells that split the roadmap into two large pieces are those shown in case II of Figure 4.5.

To connect two pieces together, a set of pairs of nodes is created, with one node from each piece, sorted in order of increasing distance using the $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function. Pairs corresponding to arcs already present in the roadmap are skipped because it is already known that these are blocked. A limit on the number of pairs to test is a parameter, so that the execution time can be bounded. For each pair, the arc connecting them is tested for feasibility—that is, that the robot should avoid the cell in question and self-collision while traversing the arc. Each arc in which the robot collides with itself is stored in a list of arcs to skip for future processing. In addition, if the attempt to connect the small piece to all large pieces fails, the cell is entered into a set for each node of the small piece so that the cell and node combination is not tested in later processing.

4.4 Summary

We have implemented both of the roadmap evaluation methods and the corresponding enhancement techniques described above. We have found that the primary use for the enhancement technique based on the minimum cut set is to ensure that the initial roadmap is connected; these results are discussed in more detail in Section 6.4.1. The enhancement technique based on ϵ-robustness has proved to be more useful, allowing the roadmap to remain connected for a larger class of obstacles. These results are discussed in more detail in Section 6.4.2.

CHAPTER 5

MOBILE ROBOTS

At its core, the mapping from the workspace to the configuration space of a robot is robot-centered, with cells in the workspace located with respect to some coordinate frame associated with the base of the robot. Therefore, if the base of the robot translates by some amount along one of the axes of this frame, the mapping shifts by the same amount. For a mobile robot, this suggests a natural partitioning of the path planning problem: separate the translational motion of the robot base from the remaining degrees of freedom of the robot. We then construct a roadmap for these other degrees of freedom as described earlier and, during the planning process, we effectively replicate this roadmap as needed at various locations over the workspace of the robot.

Planning with this approach is an iterative search process. At each iteration, the state of the roadmap at the current position of the robot is evaluated and a decision is made whether to move the base or to change the internal configuration (for example, the position of joints that control an arm mounted on the robot) of the robot. If the decision is to change the internal configuration, we follow one of the arcs of the roadmap to a node adjacent to the node corresponding to the current configuration. We use the mapping from the workspace to the roadmap to ensure that both the new node and the arc used to reach it are collision free. If, on the other hand, the decision is to move the base of the robot, a representation of the roadmap is constructed at the new location. By using a step size for translating the base equal to the size of a cell in the workspace and changing only one coordinate at a time, it is necessary only to verify that the current internal configuration of the robot is collision free at the new location. We again use the mapping from the workspace to the roadmap to ensure that the current internal configuration at the new location is collision free.

The idea of partitioning the planning problem has been used before by other PRM-type methods. For example, for flexible objects, the parameters describing the deformation can be specified separately from the rigid-body transformation, and the same deformation parameters can be replicated at random rigid-body transformations over the workspace [10, 59]. Another example is a planner used for robots with kinematic chains [56], in which some of the degrees of freedom of the robot were selected to solve

the closed-chain constraint at random configurations of the other degrees of freedom. In both of these cases, the partition is used to select samples of the configuration space when constructing the roadmap.

In our case, we do not include translations of the robot base in the construction of the roadmap. Instead, we fix the location of the base and create a roadmap for the internal configuration of the robot. This internal configuration includes the orientation of the robot base as well as the joint angles of an arm attached to the base. We then process this roadmap as we have for fixed-base robots: we first compute the mapping ϕ and then apply the enhancement techniques.

There are three advantages to using this representation. First, we reduce the number of degrees of freedom of the robot for which we construct the roadmap. This allows the samples we generate for the roadmap to cover a larger portion of the configuration space for the internal parameters of the robot. A second advantage is that the reduction in the number of degrees of freedom also reduces the size of the mapping ϕ from the workspace to the roadmap, as the volume of the workspace covered by the robot at configurations in the roadmap is reduced. The third advantage of this split representation is that the workspace of the robot is no longer restricted by the roadmap.

The primary disadvantage to this representation is that the effective size of the state space used for searching for paths is now much larger, potentially infinite in size with unrestricted workspaces, instead of being constant in size. This requires that the search procedure be guided by suitable heuristics to limit the amount of the state space that is examined. Another strategy to limit the amount of the state space that is examined is to add another planning layer on top of our planner, using, for example, a higher-level planner that works on a topological map of the environment [39, 89, 102]. Such a planner would then use our planner as a local planner in each piece of the global topological map.

We have implemented a planner using the above approach for the case of planar mobile robots, that is, robots for which the base of the robot moves in a plane. The arm attached to the robot is not restricted to planar motion. In the following, we discuss the details of implementing the planner for this type of robot. We end this chapter with an adaptation of this approach to other types of robots.

5.1 Implementation Details

In this section, we discuss the details of implementing a planner for planar mobile robots. For all of these robots, the first step of the process is to build a roadmap for the internal degrees of freedom of the robot, which include the orientation of the robot base and the joint angles of the arm attached to the base. The mapping ϕ from the workspace to this roadmap is computed next, with the location of the cells in this mapping specified with respect to the robot base. In the remainder of this section, we discuss the search procedure used for finding a path and how ϕ is used in this process.

5.1.1 Search procedure

The search procedure we use is based on the A* search procedure. The search starts at the initial configuration and works toward finding a path that reaches the goal. At each step, the search procedure removes the least-cost configuration from a priority queue for examination. Two sets of neighboring configurations are evaluated: those that change the internal configuration of the robot and those that move the base of the robot. For each of these neighboring configurations that has not yet been visited by the search process, we evaluate whether the path to it is collision free. For those neighbors that can be reached, we compute a new cost (see below) for this configuration and insert it into the priority queue. Associated with each configuration in the queue is a back link indicating how the configuration was reached. The search terminates when it reaches the goal, or when it has explored the entire workspace reachable from the initial configuration.

5.1.2 Search heuristics

The search heuristics used by the A* search procedure are captured by the cost associated with each node. This cost consists of two parts: the cost of reaching the node from the initial configuration and the estimated cost for reaching the goal. For these costs, we use the distance functions we defined earlier, modified to incorporate the translation of the robot base. For the distance functions defined on the configuration space, the modification is to add the planar displacements to the joint displacements between the two configurations. For the distance functions defined in the workspace, the current position is added to the reference points for each link. The effect for both of these modified distance functions is to prioritize the movement of the base of the robot over the change in the internal configuration of the robot.

5.1.3 Collision avoidance

For each position of the robot that is evaluated by the planner, we maintain a bitmap for indicating which nodes and arcs result in a collision between the robot and some obstacle in the environment. To update this bitmap, we first find the set of obstacles that affect the roadmap for the robot at this location. A bounding box defined by the maximum reach of an arm mounted on the robot is useful for this purpose. We then evaluate the mapping ϕ for each of these obstacles and mark the associated nodes and arcs as being in collision.

5.2 Other Applications

The same approach described above could also be applied to fixed-base robots whose first joint is revolute. The key to this is to define the cells using cylindrical coordinates instead of rectangular

coordinates and to express the location of these cells with respect to a coordinate frame bound to the first link of the robot. Then, a roadmap can be constructed for the robot, holding the first joint at a fixed position. This roadmap may then be replicated at the angular resolution of the cells in the cylindrical decomposition of the workspace. For some types of robots, the cells could also be defined in spherical coordinates.

CHAPTER 6

RESULTS

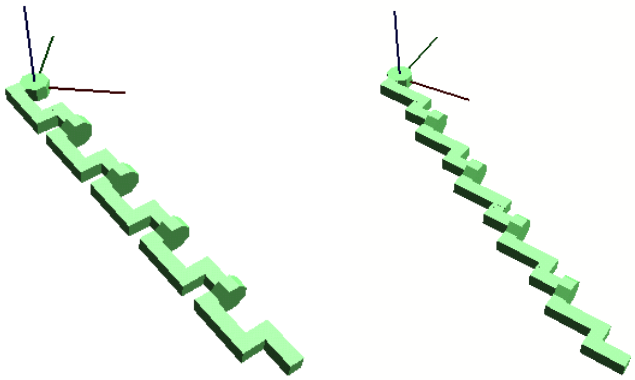
For a real-time path planner, we are interested in three parameters: roadmap update time, planning time, and data structure size. We are also interested in the preprocessing time, though this is less important. In this chapter, we evaluate our planner, studying serial-link manipulators operating in 2D and 3D workspaces. Note that the implementation is not limited to revolute joints; prismatic joints can be used as well. We limit the following discussion to revolute joints for simplicity. For reference, results from a preliminary version of the planner can be found in Appendix A.

6.1 Experimental Setup

We evaluate our planning approach with several different types of robots. The simplest of these is a set of planar serial-link manipulators with two to 20 joints. Each link of this robot is modeled as a rectanguloid with a width of 2.1 units and a height of 1 unit. The total length of these robots is fixed at 70 units, with each link length being scaled appropriately. The first joint of these robots is given full revolute range of motion, and limits are placed on the other joints. Our previous results [88] (see also Appendix A) were from robots with no joint limits; we will show some results from these robots for comparison with robots with joint limits.

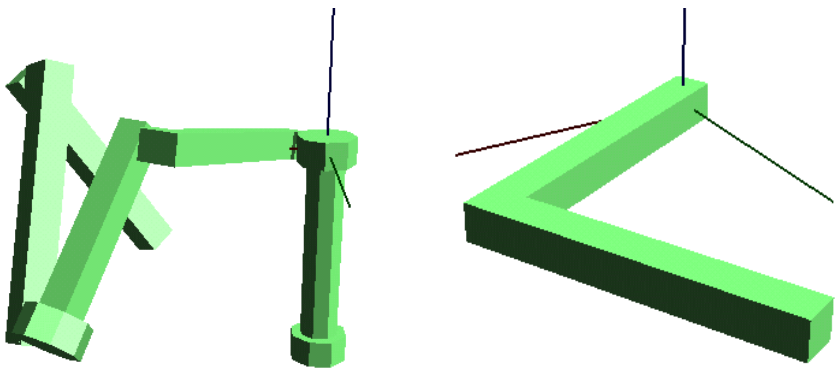
We tested two types of robots that have a 3D workspace. We tested five versions of these robots, with 3, 6, 9, 12, and 15 joints. As with the planar robots, the total length of these robots is fixed at 42 units. The cross section of the links is 1.5 units by 1.5 units. The difference between the two types is in the offset between the second and third link of each triple of links; this difference has an effect on ϵ-robustness, as will be discussed below. Examples of the 15-joint versions of these robot types are shown in Figure 6.1. The coordinate frames shown in the figure coincide with the base of the robot, and the vertical axis is the axis of rotation of the first joint.

We tested two types of mobile robots, examples of which are shown in Figure 6.2. For the first type, we attach a "base" to the type of robot shown in Figure 6.1(b). For these robots, the orientation of the



(a) (b)

Figure 6.1 Two robots with 3D workspace.



(a) (b)

Figure 6.2 Two mobile robots with 3D workspace.

base is fixed; this gives the robot shown in Figure 6.2(a) eight degrees of freedom: two for the translation of the base and six for the arm. The second type is a rigid object with six degrees of freedom.

For all of these robots, a cell in the workspace is a cube of side length one. When constructing the roadmaps, samples of the configuration space where the robot collides with itself are rejected. The same testing was not performed to test the paths connecting the configurations into a roadmap. We use the collision-detection package SOLID [119, 120] for testing for self-collision of the robot, and V-Clip [99] for testing for collisions between obstacles and the robot (both of these packages use the quickhull algorithm [16] for computing convex hulls). To evaluate how the data structures grow with the size of the roadmap, we tested roadmaps with 2048, 8192, and 16 384 nodes.

In addition, all of the results we show for the planar robots with joint limits are from averaging the results from seven runs. For each of these runs, we construct the roadmap, compute ϕ, and then

compress the representation of ϕ. This averaging of results does not include the results for the roadmap enhancement techniques. We also have not performed such an extensive systematic analysis for the 3D workspace robots or for the planar robots without joint limits.

Finally, for all computation times reported in this chapter, we use a computer with 256 Mbytes of memory and two Pentium II processors running at 450 MHz. We make no attempt to use both processors simultaneously, as we have made no effort to parallelize any of our algorithms.

6.2 Roadmap Construction

In this section we discuss results related to the construction of the roadmap used by the planner. We first discuss the issue of sampling the configuration space of the robot, and then discuss determining which of these samples to connect together to form the roadmap.

6.2.1 Sampling of configuration space

We use two different sampling distributions for sampling the configuration space of each robot. The first set of samples comes from a uniform sampling of the first joint for planar robots or the first two joints for robots with a 3D workspace; for both cases, the positions of the other joints are prespecified default values. The purpose of these samples is to touch as much as possible the maximum reach of the robot. The remaining samples come from sampling the configuration space of the robot uniformly at random. Of this second set of samples, those in which the robot collides with itself are rejected.

From the uniform random samples, we may also reject samples with probability based on the value of the cumulative distribution function at the manipulability value for the sample; the probability distribution functions for the robots we used can be found in Appendix B. When we impose the manipulability constraint on the random samples, we observe that about twice as many collision-free samples must be generated.

It is interesting to note that, when sampling $\mathcal{C}$ uniformly, it becomes exponentially more difficult as the number of joints increases to find configurations in which the robot does not collide with itself. This sampling effect can be seen in Figure 6.3, which shows the number of samples generated per node accepted for four different types of robot: two planar robots in (a) and (b), and two robots with a 3D workspace in (c) and (d). Notice that the y-axis scale of these graphs is logarithmic.

There are two main factors that account for the increased difficulty in finding self-collision-free samples. One of these is the combinatorial effect in which each additional joint provides another set of possibilities for self-collision. This is particularly true for the planar robots. The second main factor that accounts for the increased difficulty in finding self-collision-free samples is the length-to-width ratio of the links of the robot. Increasing the width of each link, or, equivalently, decreasing the length of each link, increases the size of the region of configuration space in which self-collision occurs. This effect can

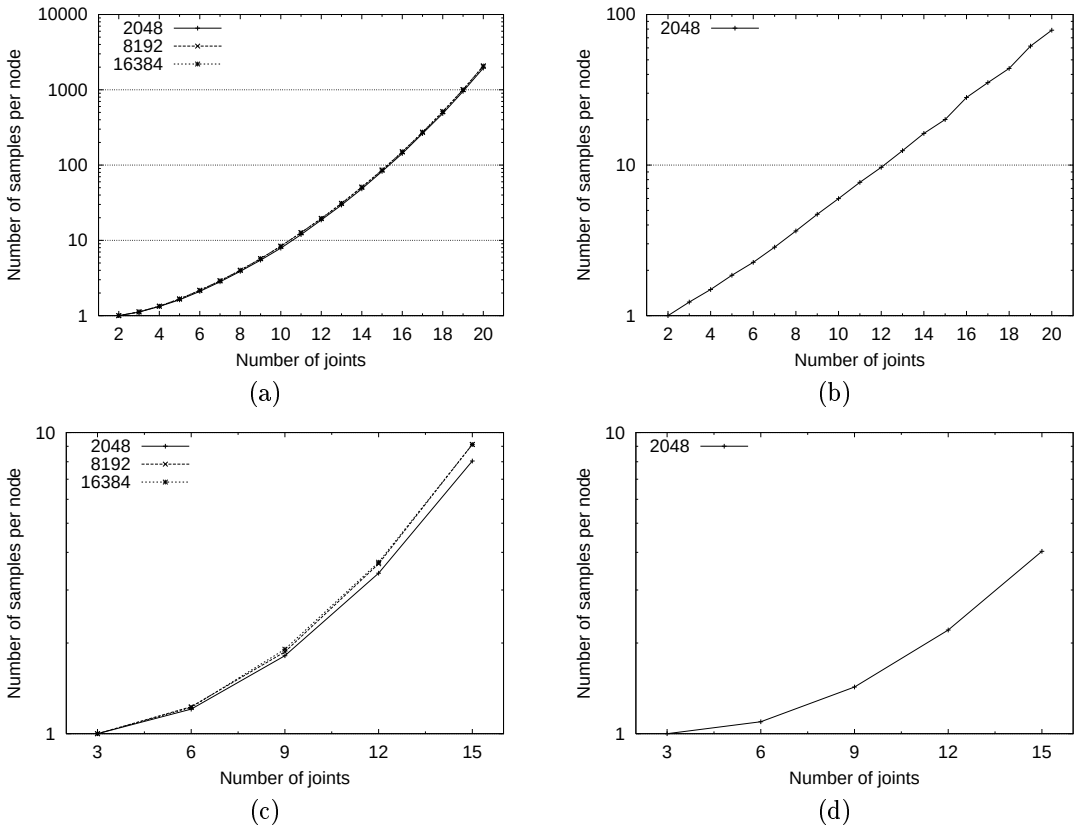


Figure 6.3 Number of samples of C-space generated per node accepted: fixed-length planar robots (a), constant length-to-width ratio planar robots (b), and fixed-length robots with 3D workspace (c,d).

be seen in Figure 6.3(a) and (b). For the robots in Figure 6.3(a), the total length and the link widths are fixed, with the result that the length-to-width ratio of each link is inversely proportional to the number of joints. If, instead, the length-to-width ratio is constant, as is the case in Figure 6.3(b), less of the configuration space corresponds to self-collisions of the robot, so that finding samples is easier.

The geometry of the robot also plays a role in determining the difficulty of finding self-collision-free samples of the configuration space of the robot. This can be seen in Figure 6.3(a) and (b), which correspond to the two robots in Figure 6.1(a) and (b), respectively.

6.2.2 Computing nearest neighbors

As described above, arcs in the roadmap are constructed for the k nearest neighbors of each node. For most of our experiments, we have concentrated on the case of $k = 5$. As is the case with our earlier work, paths connecting these neighbors are not checked for feasibility, that is, whether the robot avoids self-collision while following the paths. The reason for not checking the arcs for feasibility is

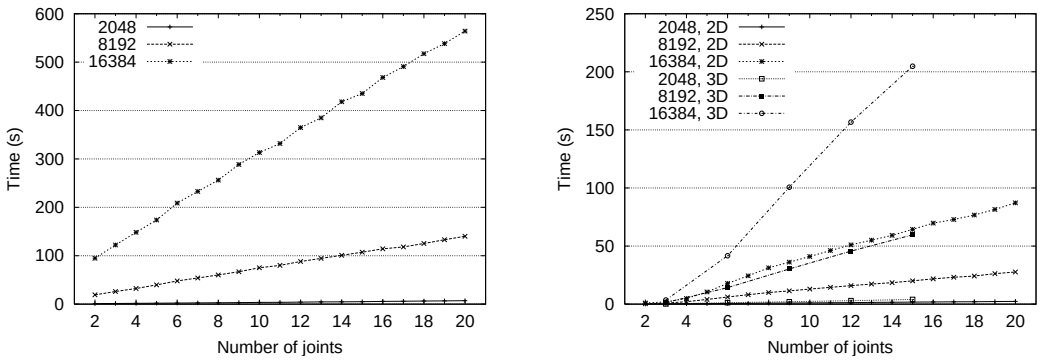


Figure 6.4 Nearest-neighbor computation time results for the $\mathcal{D}_2^{\mathcal{W}}$ distance function: (a) brute force, (b) ANN with $\epsilon = 0$.

that there is no efficient means for integrating this test into the distance functions and nearest-neighbor algorithms used to construct the roadmap. Not checking the arcs is also equivalent to assuming that the local planner can connect any two self-collision-free configurations with a path that avoids self-collision, though this assumption does not hold for our local planner.

In our previous experiments, we had an additional justification for not checking the arcs. In those experiments, we used a set of planar robots without joint limits. To allow these robots full freedom of motion, we allowed the links to pass through each other, which meant that, by definition, there was no self-collision region in the configuration space of these robots. We did, however, use the self-collision constraint when sampling the configuration space for the nodes of the roadmap; the purpose of this was to bias the sampling process so that the robot reached a larger portion of the workspace.

We have evaluated three different methods for computing nearest neighbors: brute-force, which can be used with all distance functions; the approximate nearest-neighbors (ANN) library [14], which can only be used with Minkowski-type metric functions; and the approximation scheme we described in Section 2.4.1 for the distance functions based on the path midpoint. The most basic of these is the brute-force method of computing the distance between all pairs of nodes and keeping the closest k for each node. This algorithm requires time quadratic in the number of nodes. Examples of the computation time for creating the roadmap using this method can be seen in Figure 6.4(a) for the $\mathcal{D}_2^{\mathcal{W}}$ distance function and in Figure 6.5(a) for the $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function.

The second method uses the approximate nearest-neighbors (ANN) library described in [14]. This library will return a neighbor to a point that is within a factor of $1 + \epsilon$ of the closest neighbor. This library has a large number of parameters that control how its search tree is constructed and how the tree is searched. In our data, we have found that the search tree based on the standard kd-tree worked

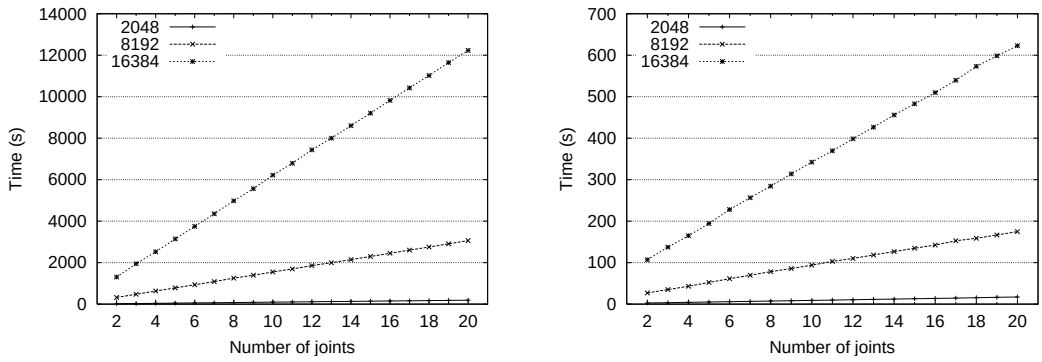


Figure 6.5 Computation time for determining nearest neighbors for constructing the roadmap for the $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function: (a) brute-force exact approach, (b) approximate approach.

well, but, in general, the other tree types performed worse than the standard kd-tree. We also found that using the standard tree search was faster, in general, than using the priority search. The parameter that had the greatest impact was ϵ, where higher values of ϵ resulted in shorter computation times.

Shown in Figure 6.4(b) is the computation time for ANN with $\epsilon = 0$ using the $\mathcal{D}_2^{\mathcal{W}}$ distance function and robots with 2D and 3D workspaces. For both of these robot types, the dimension of the points given to the ANN library is three times the number of joints in the robot (one reference point with 3D coordinates per joint). As can be seen in Figure 6.4(b), the ANN library exploits the lower inherent dimensionality of the points for the planar robots, resulting in faster computation times. For both types of robots, the ANN library is faster than the brute force approach; see Figure 6.4(a) for comparison with the computation times for the planar robots (the computation times for the 3D workspace robots are similar to those for the planar robots using the brute-force approach).

Shown in Figure 6.6 is the effect of varying ϵ with the ANN library. For both graphs, the roadmap has 16 384 nodes. In the case of the $\mathcal{D}_2^{\mathcal{C}}$ distance function with $\epsilon = 0$, there is a crossover point at 19 joints where the brute-force approach becomes faster. In this case, the ANN library is examining most points to find the closest, and the overhead of traversing the tree structure causes the running time to exceed the brute-force approach. For other values of ϵ, the ANN library is much faster. The ANN library is also faster for the $\mathcal{D}_2^{\mathcal{W}}$ distance function for all values of ϵ.

Shown in Figure 6.5 is the computation time for computing the nearest neighbors using the $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function, with the brute-force approach shown in Figure 6.5(a) and the approximate approach from Section 2.4.1 shown in Figure 6.5(b). In the approximate approach shown in Figure 6.5(b), the nearest five neighbors for each node of the roadmap are computed by first using the brute-force approach to find the 50 nearest neighbors using the $\mathcal{D}_2^{\mathcal{W}}$ distance function, and then selecting the five closest of

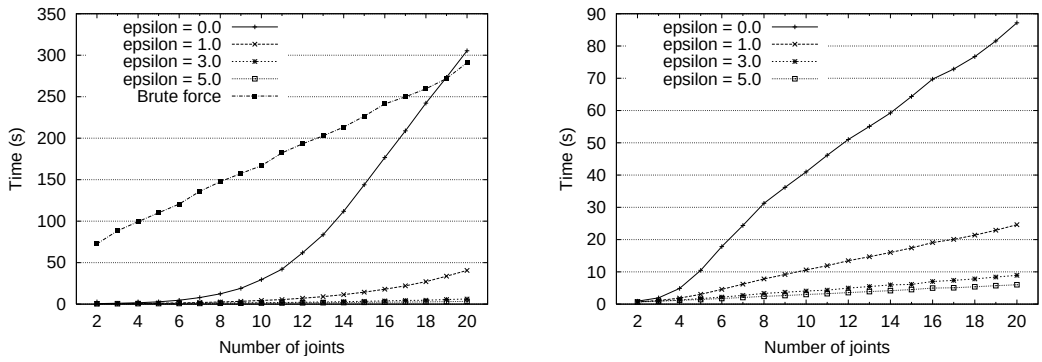


Figure 6.6 The effect of varying ϵ with ANN and a 16 384 node roadmap: (a) $\mathcal{D}_2^{\mathcal{C}}$ distance function, (b) $\mathcal{D}_2^{\mathcal{W}}$ distance function. For comparison, the brute-force computation time for $\mathcal{D}_2^{\mathcal{C}}$ is shown in (a).

these using the $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function. The result is a factor of about 20 improvement in computation time for roadmaps with 16 384 nodes. Since the $\mathcal{D}_2^{\mathcal{W}}$ distance function is also a metric, the ANN library can be used in the first step for a larger improvement in the computation time.

6.3 Computing ϕ

In this section, we discuss the computation of ϕ, the size of the resulting data structures based on a naive encoding, and the results for using a more efficient encoding.

6.3.1 Computing ϕ_n

Shown in Figure 6.7 is the computation time for ϕ_n for planar robots and two different algorithms for computing $\phi_n^{-1}(\mathrm{q})$ for each node q in the roadmap. The original algorithm we used is based on sweeping a cube over the robot and testing for intersection using the V-Clip collision-detection library. The new algorithm is described in Section 3.1. For both algorithms, the computation time increases linearly with the size of the geometric model of the robot, which increases linearly with the number of joints for these planar robots. The new algorithm, though, is much faster, by about a factor of 85 for 20 joints and 16 384 nodes. The difference between the 2048 node case and the other two cases in Figure 6.7 is due to the memory allocation overhead when creating the data structure that represents ϕ_n. This cost is amortized over more nodes in the larger roadmaps.

The overall size of ϕ_n is the product of the number of nodes and the average number of cells the robot intersects in the workspace. For the planar robots with a fixed maximum length, this means that the size of ϕ_n is largely independent of the number of joints. For our experiments, the size of ϕ_n per node in $\mathcal{G}_n$ averages around 1000 bytes. The situation is different for the 3D workspace robots, in which

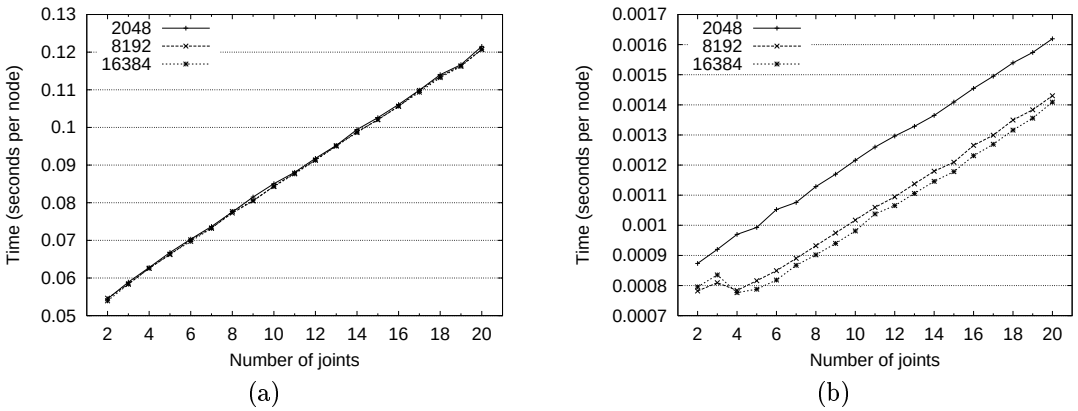


Figure 6.7 Computation times per node for ϕ_n using the two different algorithms: (a) original algorithm based on V-Clip, (b) new algorithm.

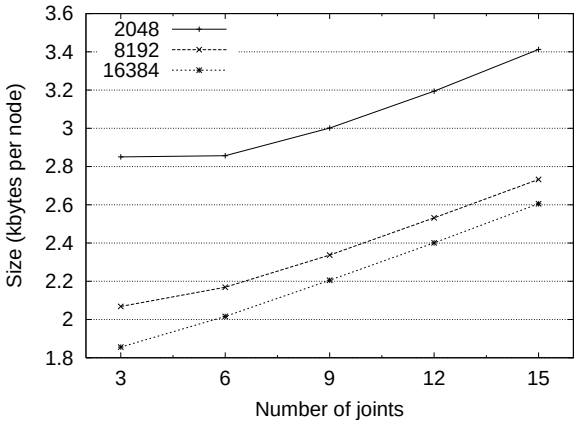


Figure 6.8 Size of ϕ_n for 3D workspace robots.

the number of cells that intersect with the robot increases with the number of joints. The effect of this on the size of ϕ_n can be seen in Figure 6.8, which shows the size of ϕ_n per node in the roadmap. The sublinear increase in the size of ϕ_n with the number of nodes is the result of the amortization of the overhead for each cell in ϕ_n.

6.3.2 Computing ϕ_a

The size of ϕ_a is shown in Figure 6.9 for several distance functions for the planar robots with joint limits; an explanation for the labels can be found in Table 6.1. This figure shows how large the naive representation of ϕ_a can grow for the planar robots. The representation of ϕ_a is too large for the 3D workspace robots to compute with the naive representation; therefore, we defer showing the size of ϕ_a for these robots until Section 6.3.3.

Table 6.1 Definition of the labels used on figures for distance functions.

Label	Description
w	The $\mathcal{D}_2^{\mathcal{W}}$ distance function
W	The $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function
F	The $\mathcal{D}_{\mathcal{F}}^{\mathcal{W}}$ distance function
a	When associated with W or F, indicates that an approximation approach is used to find the nearest neighbors

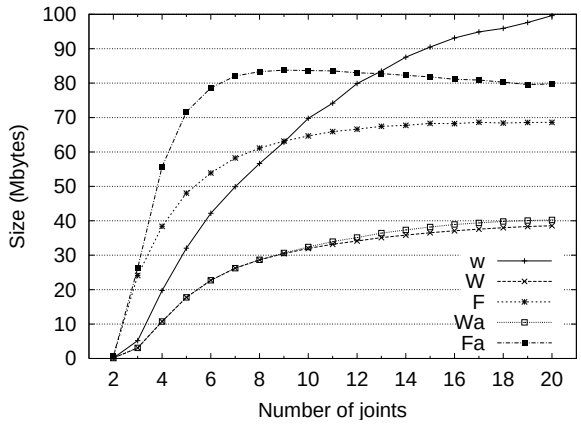


Figure 6.9 Size of ϕ_a for planar robots with joint limits and for roadmaps with 16 384 nodes and five distance functions.

Figure 6.9 also shows the effect of the different distance functions and methods for computing nearest neighbors on the size of ϕ_a. The smallest size of ϕ_a for all number of joints is for the $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function, both for exact and approximate nearest neighbor computation (the approximate form results in a slightly larger size). The distance function defined on the end-effector coordinate frame performs worse, which is expected because this distance function uses less information about the positions of the links of the robot. The large gap between the exact and approximate approaches for computing nearest neighbors using this distance function indicates a poor match between the distance function used in the first step of the approximation and the distance function used in the second step.

Also notice that these sizes for planar robots with joint limits are smaller than those for robots without joint limits; see Figure 3.5 to compare the size of ϕ_a for the $\mathcal{D}_2^{\mathcal{W}}$ distance function for a robot without joint limits. Another way to view this is to look at the average number of cells in the swept volume for each arc, which is shown in Figure 6.10(a) and (b) for the planar robots without and with joint limits, respectively. Incorporating the midpoint of the path followed by the robot into the distance function also reduces the swept volume, as desired. These averages are shown in Figure 6.10(c). In 3D, the swept volumes are much larger; an example is shown in Figure 6.10(d). For all of the robots, the

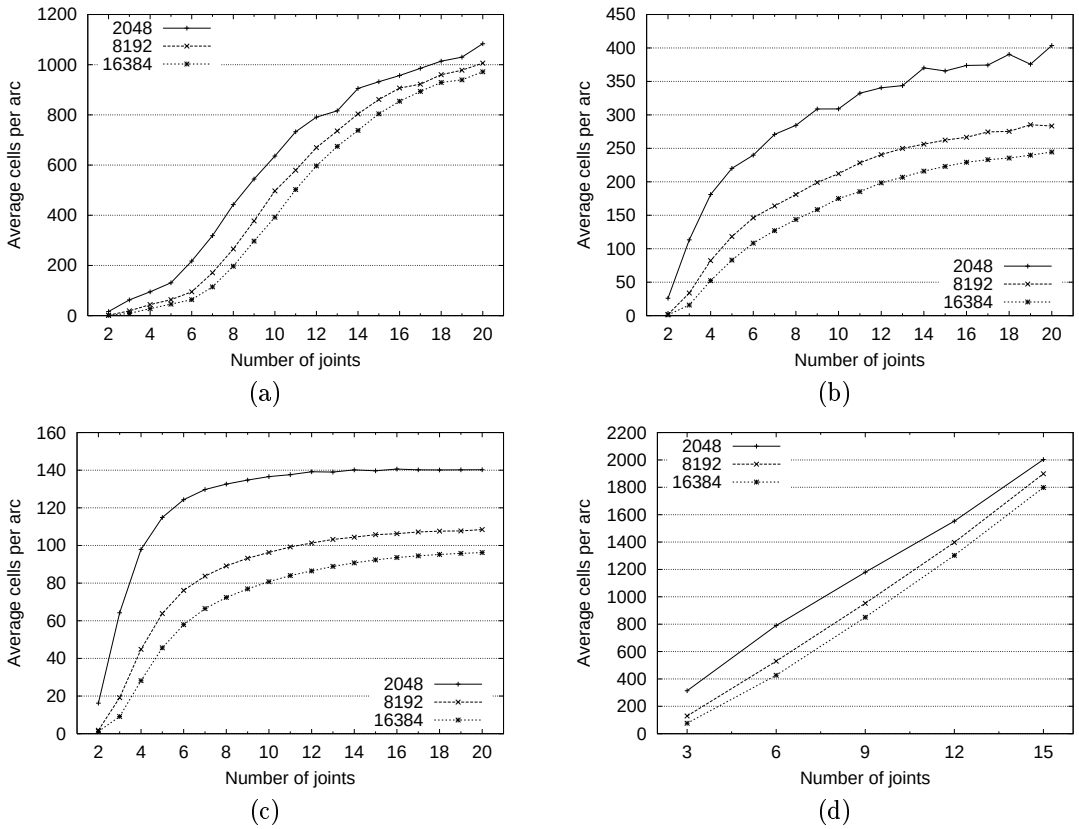


Figure 6.10 Average swept volume per arc in ϕ_a: (a) planar robots without joint limits, $\mathcal{D}_2^{\mathcal{W}}$ distance function; (b) planar robots with joint limits, $\mathcal{D}_2^{\mathcal{W}}$ distance function; (c) planar robots with joint limits, $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function; (d) 3D workspace robots, $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function.

average swept volume per arc decreases as the number of nodes in the roadmap increases: more nodes result in shorter paths for the arcs, and therefore smaller swept volumes.

The computation time for ϕ_a has also benefited from the new algorithm for computing ϕ_n^{-1}. Some computation times using the new algorithm are shown in Figure 6.11 (see Section A.2.4 for some of the previous computation times). Figure 6.11(a) shows the computation times for the planar robots and a 16 384 node roadmap constructed using five different distance metrics (see Table 6.1 for a description of the labels). The increase in the computation time with the number of joints tends to be linear, which corresponds to the linear increase in the size of the geometric description of the robot. The computation times shown here correlate in relative magnitude with the size of ϕ_a shown in Figure 6.9: the distance function that results in a larger size for ϕ_a also results in more time needed to compute ϕ_a.

The computation time shown in Figure 6.11(b) for the robots with a 3D workspace shows an additional factor: the effect on computation time when the main memory of the computer is exceeded and pages of memory used by the program are swapped out to disk. This is particularly apparent for the 16 384

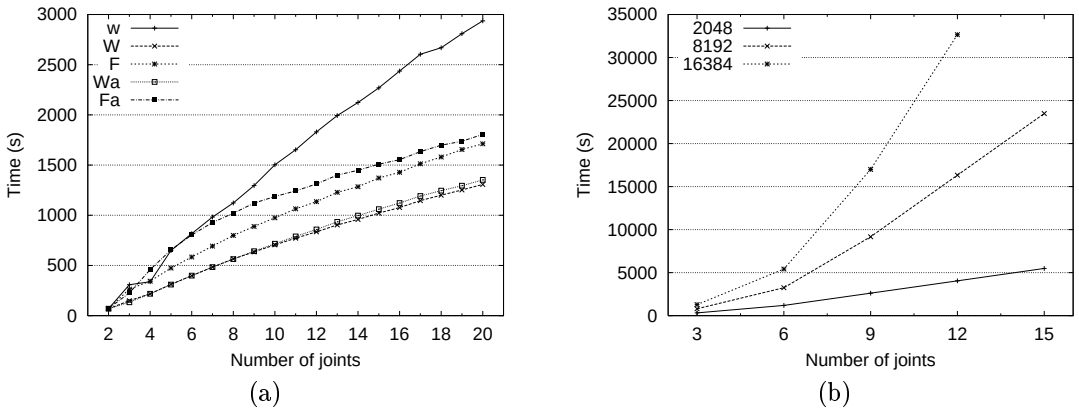


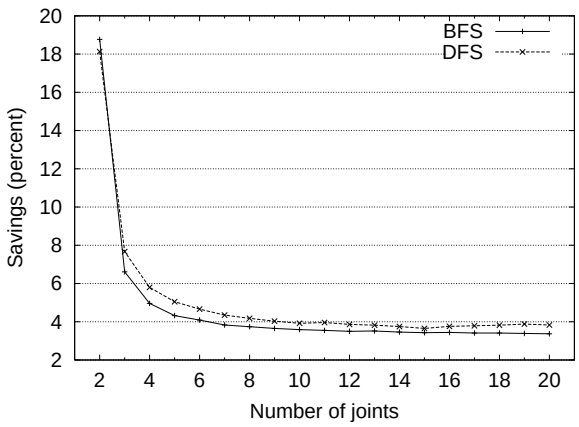
Figure 6.11 Computation time for ϕ_a for planar robots (a) and for 3D workspace robots (b). The roadmaps in (a) have 16 384 nodes. In (b), the roadmaps are constructed with the $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function.

node roadmap; indeed, the reason that the 15-joint case is not shown in the figure is that the computer on which these computations took place ran out of memory in the middle of the computation of ϕ_a. The program that computes ϕ_a uses techniques that improve the memory reference locality, and this reduces the time penalty for swapping.

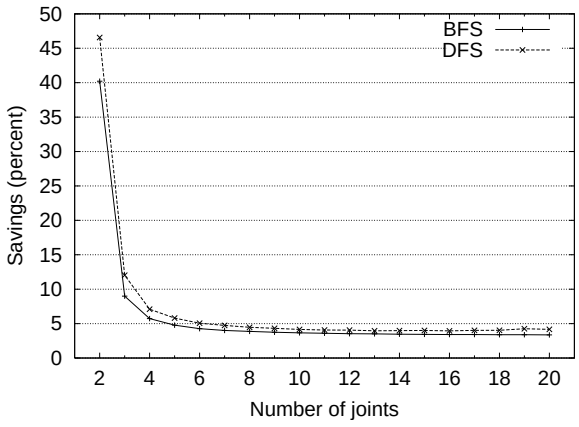
6.3.3 Compression results

We start the discussion on the compression of ϕ with some results for the efficient encoding of $\phi(w^*)$ (see Section 3.2.2). The basic idea is that our roadmap imposes an intrinsic order to the nodes and arcs and that, by choosing the appropriate starting node, this order could be used as a basis for run-length encoding of $\phi(w^*)$. We tested this idea on ϕ_n, testing both depth-first search (DFS) and breadth-first search (BFS) as the method for selecting the order. For each cell in ϕ_n, we tested each node as a candidate starting node, and kept the one that produced the best results. The results of this test are shown in Figure 6.12. As can be seen in the figure, the results are rather disappointing, showing that it is successful for robots with two, or possibly three, joints. The results do not change significantly for different roadmaps constructed using the same set of nodes but different distance functions. What this shows is that the roadmaps we use for planning do not cluster the nodes well enough for the run-length encoding to work well. There may be other roadmaps with the same set of nodes for which this method will work better.

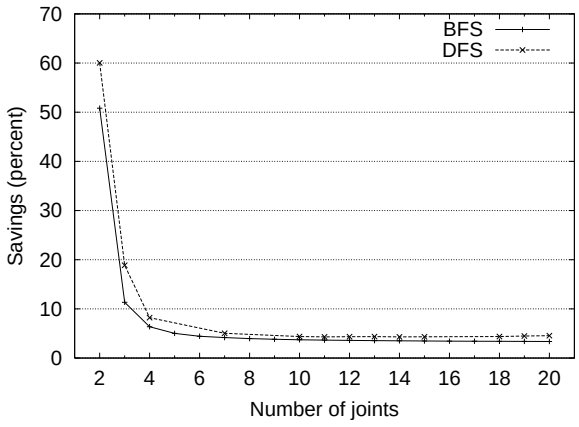
Next, we consider the compression of ϕ that we achieve by encoding ϕ over neighborhoods in $\mathcal{W}$. We have evaluated three neighborhood models: region-growing, "union" hierarchical, and "best" hierarchical. For the two hierarchical neighborhoods, "union" and "best" define the set of labels that is stored with the parent: with "union," the union of the labels of the children is stored, and with "best," the set of



2048 nodes



8192 nodes



16 384 nodes

Figure 6.12 Compression results using run-length graph encoding for ϕ_n. The savings are in terms of the reduction of the number of labels.

labels that minimizes the label count among the parent and children is stored. For the region-growing neighborhood models, the results shown below use the parameters of size factor 2 and depth limit 17. See Section 3.2.3 for more details.

Shown in Figure 6.13 are the savings that we achieve using the different neighborhood models for ϕ_n for both the planar robots and the 3D workspace robots. Using the region-growing definition for the neighborhoods, the amount of savings we achieve is relatively flat with different numbers of joints, with a factor of around three for the planar robots and two for the 3D workspace robots. The savings also tend to be greater for roadmaps with fewer nodes, indicating that the distribution of labels is more uniform for roadmaps with fewer nodes.

Using the hierarchical models for compression of ϕ_n does not yield results as good as those for the region-growing models. Indeed, for the 3D workspace robots, using the "union" model for the neighborhoods results in an expansion of the number of labels (a "savings" of less than one). The hierarchical models tend to perform better as the number of joints of the robot increases. This is due in part to the increase in the average number of cells occupied by the robot at a configuration as the number of joints increases. Another factor is that, with more joints, the links of the robot have a greater tendency to cluster around the origin. With fewer labels in the outer regions of the workspace, the regions of uniformity are larger.

Shown in Figure 6.14 are the savings that we achieve using the different neighborhood models for ϕ_a for planar robots. Shown in Figure 6.14(a), (c), and (e) is the case for robots without joint limits and a roadmap constructed using the $\mathcal{D}_2^{\mathcal{W}}$ distance function. This is the example case for which the region-growing method of compression was designed, and the savings reach a factor of 8.5 at 20 joints using this neighborhood definition. The hierarchical models also perform well in this case, though the "best" model achieves nearly twice the level of savings of the "union" model. The curve for the savings in number of labels with respect to number of joints follows the curve for the average swept volume per arc in the roadmap; see Figure 6.10(a) for reference. The basic idea is that, as the swept volume per arc increases, the amount of coherence in the workspace also increases, increasing the savings that can be achieved.

For a robot with joint limits and using the $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function, the compression results are quite different. This difference can be seen in Figure 6.14(b), (d), and (f). For these robots, the average swept volume per arc converges to some constant value as the number of joints increases, and this same convergence is reflected in the savings achieved. An explanation for the peak in the savings ratio at four joints is that, for robots with fewer joints, the swept volumes of the arcs are more evenly distributed over the workspace, allowing for greater savings. This does not hold for two joints and, to a lesser extent, three joints, because for these two robots, the number of labels stored is already small, allowing for very little savings.

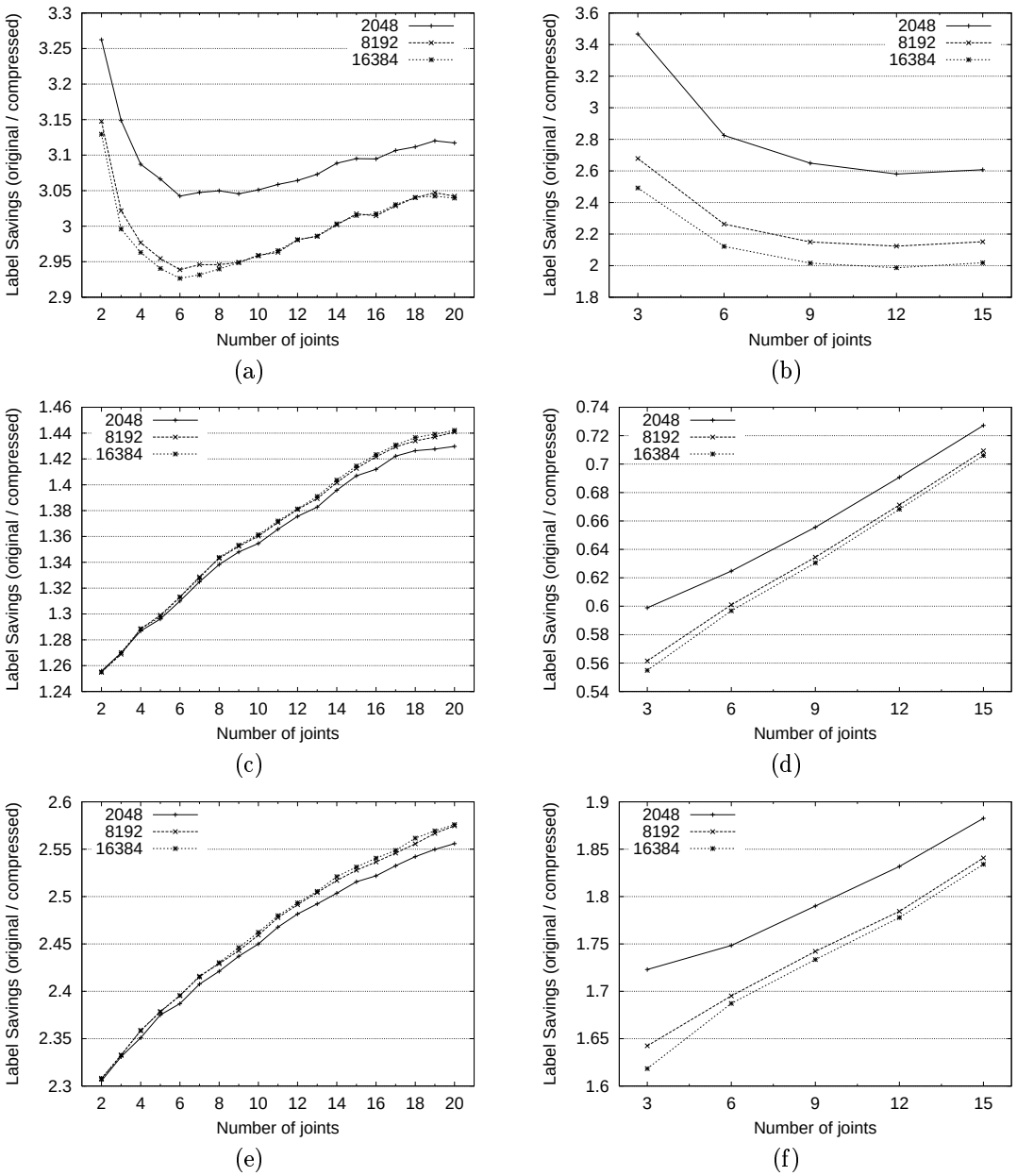


Figure 6.13 Compression results for ϕ_n for planar robots (a, c, e) and robots with 3D workspace (b, d, f). The neighborhoods are region-growing in (a) and (b), "union" hierarchical in (c) and (d), and "best" hierarchical in (e) and (f).

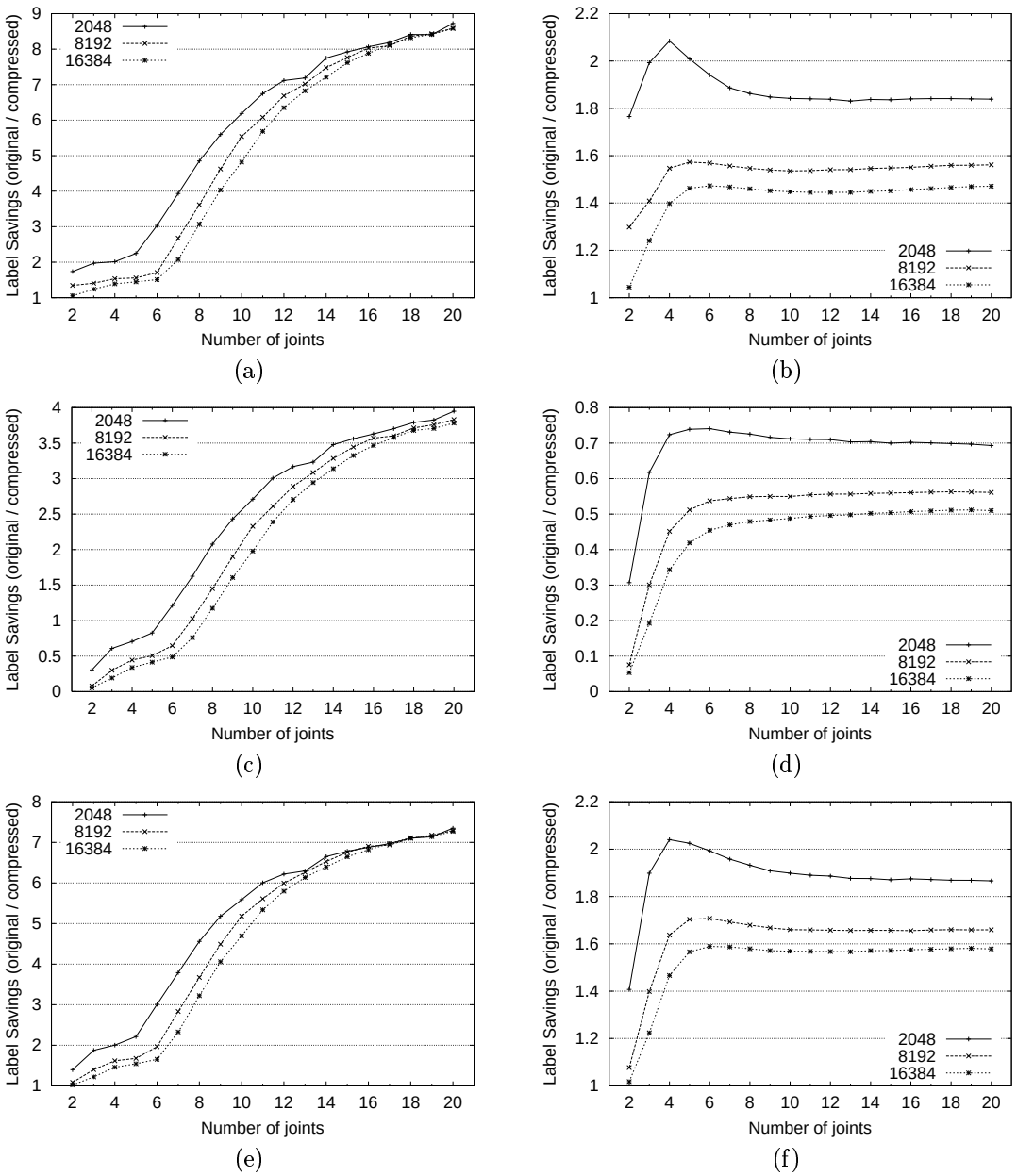


Figure 6.14 Compression results for ϕ_a for planar robots: $\mathcal{D}_2^{\mathcal{W}}$ distance function for robots without joint limits (a, c, e) and $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function for robots with joint limits (b, d, f). The neighborhoods are region-growing in (a) and (b), "union" hierarchical in (c) and (d), and "best" hierarchical in (e) and (f).

Notice that for this roadmap, the "best" hierarchical and the region-growing models produce similar savings, with the "best" model slightly ahead. This is also true for the roadmaps for these robots for the other distance functions, though the region-growing neighborhood results are slightly better for the $\mathcal{D}_2^{\mathcal{W}}$ distance function. This indicates that the heuristics used to define the neighborhoods for the region-growing model are not well suited to this data. In addition, the "union" hierarchical model results in an expansion of the number of labels, which also holds for the other distance functions except for $\mathcal{D}_2^{\mathcal{W}}$. For the $\mathcal{D}_2^{\mathcal{W}}$ distance function, there is a crossover point where the neighborhood model results in a reduction in the number of labels. This crossover point depends on the number of nodes in the roadmap, as is also the case in Figure 6.14(c).

Shown in Figure 6.15 are the savings that we achieve using the different neighborhood models for ϕ_a and for roadmaps constructed with the $\mathcal{D}_{m2}^{\mathcal{W}}$ and $\mathcal{D}_{\mathcal{F}}^{\mathcal{W}}$ distance functions. Results for the second distance function are given to show that the compression ratios do improve for roadmaps constructed with distance functions that result in larger average swept volumes for the arcs. The results for ϕ_a for these robots follow the general trend for the planar robots; that is, the region-growing neighborhoods tend to produce the most savings, followed by the "best" hierarchical, and with the "union" hierarchical resulting in an expansion of the number of labels in many cases.

Finally, we consider the effects of using a more efficient encoding for the labels. We have already examined the effects of using different codes for the different types of data stored as part of ϕ; see Table 3.4. In the remainder of this section, we examine the total space required for ϕ, including the representation of the labels and various sources of overhead, such as the reference number stored in each cell for the region-growing neighborhoods, the count numbers which indicate the number of labels stored with each reference and cell, and the length numbers, which are used to locate a particular cell or reference in the compressed data.

Shown in Figure 6.16 is the result for ϕ_n with a 16 384 node roadmap for both planar and 3D workspace robots; a description of the labels used in this and later figures is given in Table 6.2.

For the planar robots, compression using the region-growing neighborhoods clearly outperforms the other methods. This is due in large part to the reduction in the number of labels that must be stored using this model. The region-growing neighborhoods also have a slight advantage over the hierarchical neighborhoods in terms of the overhead for storing the neighborhood structure; this is due to the hierarchical neighborhoods being defined for 3D workspaces, and this hierarchy is less efficient for the planar case.

For the 3D workspace robots, on the other hand, the "best" hierarchical neighborhoods tie the performance of the region-growing neighborhoods for robots with more joints, even though the region-growing neighborhoods result in fewer labels. This shows the trade-off between arbitrary neighborhoods,

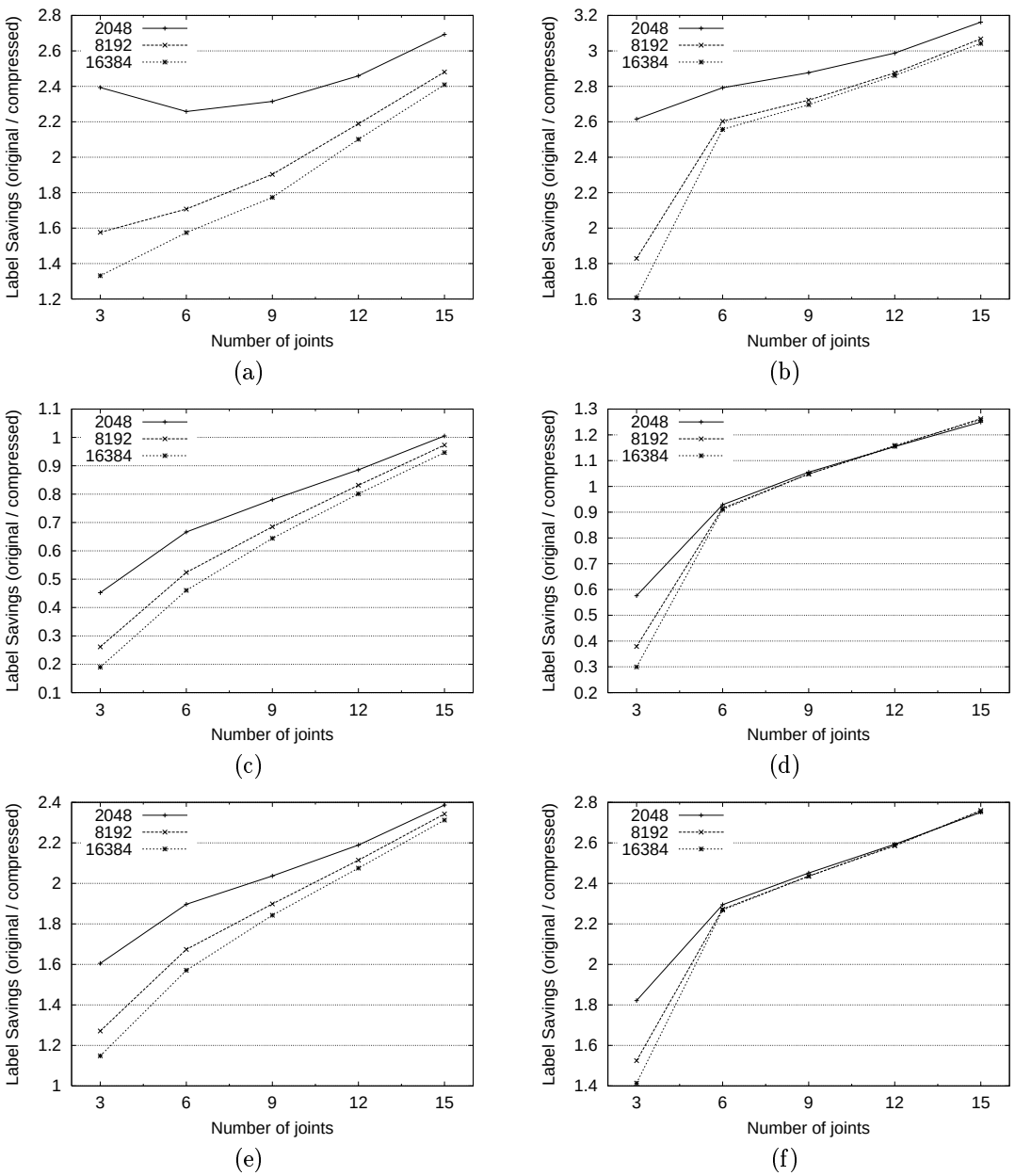


Figure 6.15 Compression results for ϕ_a for robots with 3D workspace: roadmap constructed with the $\mathcal{D}^{\mathcal{W}}_{m2}$ distance function (a, c, e), and roadmap constructed with the $\mathcal{D}^{\mathcal{W}}_{\mathcal{F}}$ distance function. The neighborhoods are region-growing in (a) and (b), "union" hierarchical in (c) and (d), and "best" hierarchical in (e) and (f).

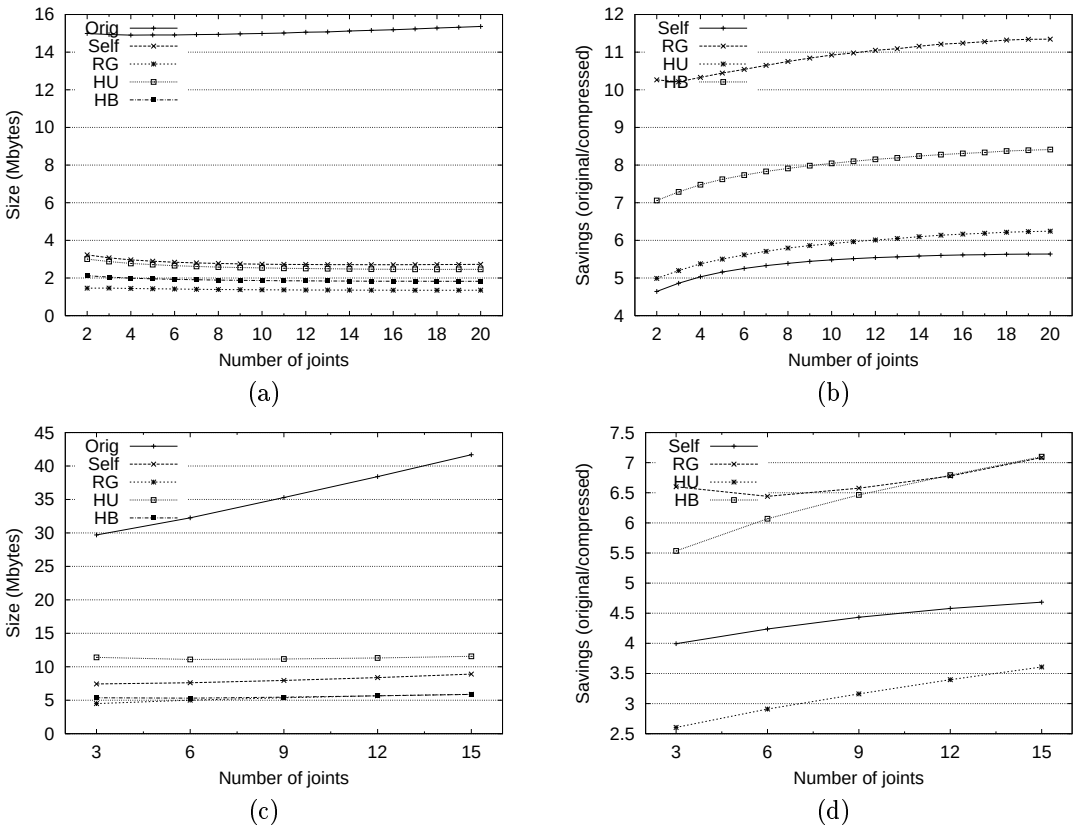


Figure 6.16 File sizes and compression ratios for ϕ_n and a 16 384 node roadmap with efficient label encodings and different neighborhood models. The results for planar robots are in (a) and (b), and for 3D workspace robots in (c) and (d).

in which a reference number must be stored with each workspace cell, and the hierarchical neighborhoods, in which the reference for each cell is implicit in the tree structure.

Shown in Figure 6.17 are the compression results for ϕ_a for planar robots without joint limits. The roadmap for this case has 16 384 nodes and was constructed using the $\mathcal{D}_2^{\mathcal{W}}$ distance function. The compression results here follow the results obtained for the savings in the number of labels using the different neighborhood models. Here, again, the neighborhoods defined by region-growing have the best compression performance, reaching a savings factor of more than 70.

The situation is a bit different for robots with joint limits. First, even for roadmaps constructed using the same distance function, ϕ_a is smaller for robots with joint limits than for those without. This difference can be seen by comparing Figures 6.18(a) and 6.17(a), where the uncompressed form of ϕ_a for robots with joint limits is about one quarter the size of that for robots without joint limits. The savings achieved using the different compression models are also lower, as shown in Figure 6.18(b). The region-growing neighborhoods still perform the best, reaching a compression ratio of about 24.

Table 6.2 Definition of the labels used on figures for the different encoding models.

Label	Description
Orig	Naive encoding of the labels
Self	Efficient encoding of the labels only
RG	Region-growing neighborhood model with efficient label encoding
HU	"Union" hierarchical neighborhood model with efficient label encoding
HB	"Best" hierarchical neighborhood model with efficient label encoding

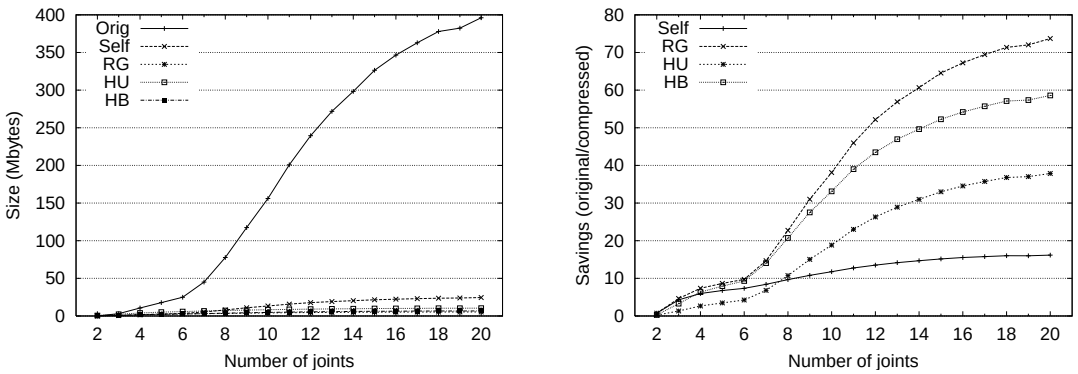


Figure 6.17 File sizes and compression ratios with efficient label encodings and different neighborhood models for ϕ_a for a 16 384 node roadmap for planar robots without joint limits.

This trend continues in Figure 6.18(c) and (d). In this case, the roadmap was constructed with 16 384 nodes using the $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function. Using this distance function, the average swept volume for each arc is lower, resulting in less space required to store ϕ_a using the naive encoding. The corresponding compression ratios are also smaller, though the region-growing neighborhoods still perform the best.

The compression picture is more mixed for the robots with a 3D workspace. An example for this is shown in Figure 6.19, which shows the final file sizes for 16 384 node roadmaps computed using two different distance functions. For these cases, the size of the file for the naive encoding is not available, since it is too large. In the case with the roadmap constructed with the $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function, the best performance was given by two of the neighborhood models: region-growing and "best" hierarchical. On the other hand, for the roadmap constructed with the $\mathcal{D}_{\mathcal{F}}^{\mathcal{W}}$ distance function, the region-growing neighborhoods perform the best.

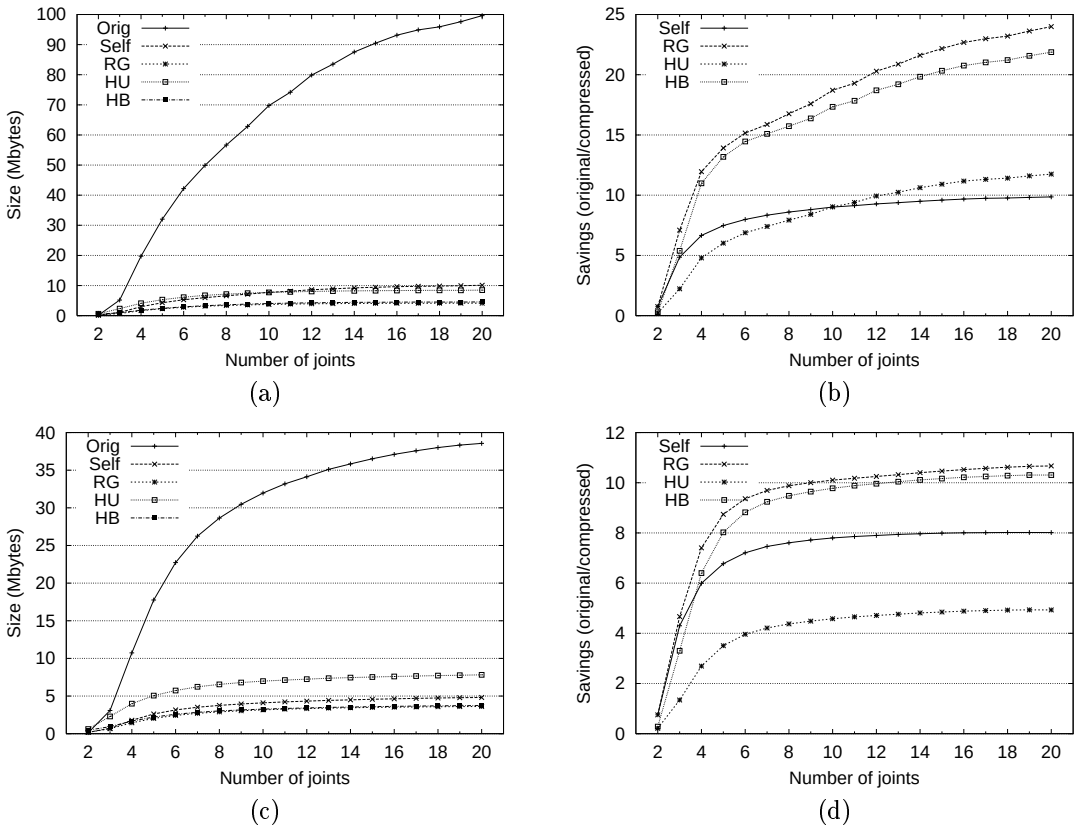


Figure 6.18 File sizes and compression ratios with efficient label encodings and different neighborhood models for ϕ_a for a 16 384 node roadmap for planar robots with joint limits.

6.4 Graph Enhancement

In this section we discuss results from using our enhancement techniques on the roadmaps, including examining the minimum cut set of the roadmap represented as an unweighted graph and the ϵ-robustness property of the roadmaps.

6.4.1 Enhancement based on minimum cut sets

The purpose of using a minimum cut set algorithm is to find clusters of nodes in the roadmap that are not well connected to the rest of the roadmap. For our roadmaps, we define the roadmap to be well connected if the number of arcs in the minimum cut set is equal to the minimum number of arcs connected to any node in the roadmap. Using this definition, it turns out that most of the roadmaps are already well connected, particularly when the $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function was used to construct the roadmap. There are two primary exceptions to this. One exception occurs for robots with two or three joints and tends to be more prevalent for roadmaps with more nodes. An explanation for this is that for

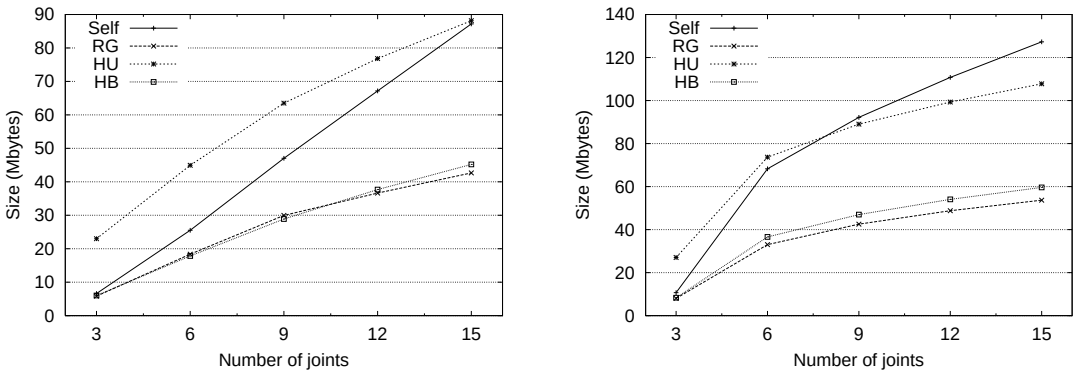


Figure 6.19 Compressed file sizes for ϕ_a and a 16 384 node roadmap for 3D workspace robots. The roadmaps were constructed with the $\mathcal{D}^{\mathcal{W}}_{m2}$ distance function (a) and the $\mathcal{D}^{\mathcal{W}}_{\mathcal{F}}$ distance function (b).

low dimensional configuration spaces, the samples are closer together and therefore more readily form clusters.

The other exception to well-connected roadmaps is when the roadmap construction process fails to produce a roadmap with a single connected component; for this situation, the minimum cut set procedure is a useful tool for ensuring that the roadmap does consist of a single component. We have observed two cases for which this happens. One of these is for planar robots and roadmaps constructed using the $\mathcal{D}^{\mathcal{W}}_{2}$ distance function. This case tends to occur for robots with 14 or more joints, and the cluster tends to consist of the samples drawn uniformly from the range of the first joint, whereas the positions of the other joints are fixed (see Section 6.2.1 for a discussion of these samples). These samples correspond to a dense sampling of a line in the configuration space, and for robots with more joints, these samples tend to be "far" from the other samples taken uniformly at random relative to other samples on the line.

The other case we have observed, in which the roadmap constructed did not consist of a single component, occurred for one of the 3D workspace robots and the roadmap constructed using the $\mathcal{D}^{\mathcal{W}}_{\mathcal{F}}$ distance function. This case was also more prevalent for robots with more joints and for roadmaps with more nodes.

6.4.2 Enhancement based on ϵ-robustness

The purpose of the ϵ-robustness test is to find individual cells and cubical clusters of cells that disconnect the roadmap. The enhancement algorithm then attempts to reconnect the pieces of the roadmap for each of these cells. To find the connected components of the roadmap, we use a non-recursive version of the DFS-based algorithm in [115] for finding strongly connected components. This

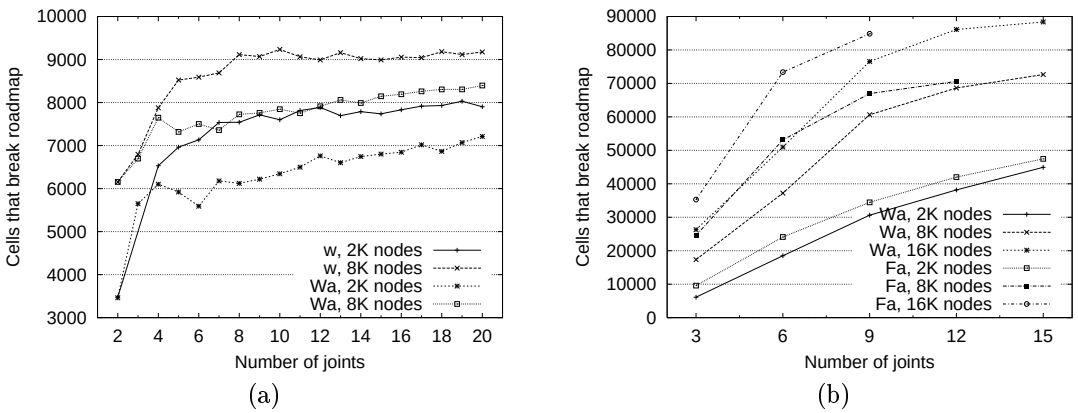


Figure 6.20 Test for ϵ-robustness for roadmaps constructed with different distance functions: (a) planar robots and (b) 3D workspace robots.

particular algorithm allows the ϵ-robustness process to work with directed roadmaps, though all the roadmaps we test are undirected.

We begin our discussion with a test of the conjecture that we stated in Chapter 2, namely that a more robust roadmap will result from the use of a distance function that better takes into account the motion of the robot when following a path connecting two configurations. To evaluate this, we sweep a single-cell obstacle over the workspace of the roadmap, and count the number of cell locations for which the roadmap is disconnected into more than one piece.

Some results for this test are shown in Figure 6.20 for both planar robots and 3D workspace robots. The labels in these graphs for the distance functions follow the notation in Table 6.1. In Figure 6.20(b) we excluded the cases for the $\mathcal{D}_{\mathcal{F}}^{\mathcal{W}}$ distance function for which the initial roadmap was not connected; for these cases, the ϵ-robustness test reports that all cells disconnect the roadmap. For comparison, note that the number of nonempty cells in the workspace is about 15 000 for planar robots, and 300 000 for 3D robots.

The data in Figure 6.20 show some evidence to support our conjecture: those distance functions for which the average swept volume associated with each arc is higher also result in more cells that cause the roadmap to become disconnected. The average swept volume is larger for both the $\mathcal{D}_2^{\mathcal{W}}$ and $\mathcal{D}_{\mathcal{F}}^{\mathcal{W}}$ distance functions relative to the $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function; therefore, more cells cause the roadmap to become disconnected. The gap between the two distance functions is larger in the planar case shown in Figure 6.20(a) because the difference in swept volume is much larger for that pair of distance functions.

Next, we examine the number of arcs added to the roadmap as a result of enhancing a roadmap to improve its ϵ-robustness properties. For this purpose, we scan a cube obstacle over the workspace, starting with a cube of side length one and increasing the side length by one for each successive pass.

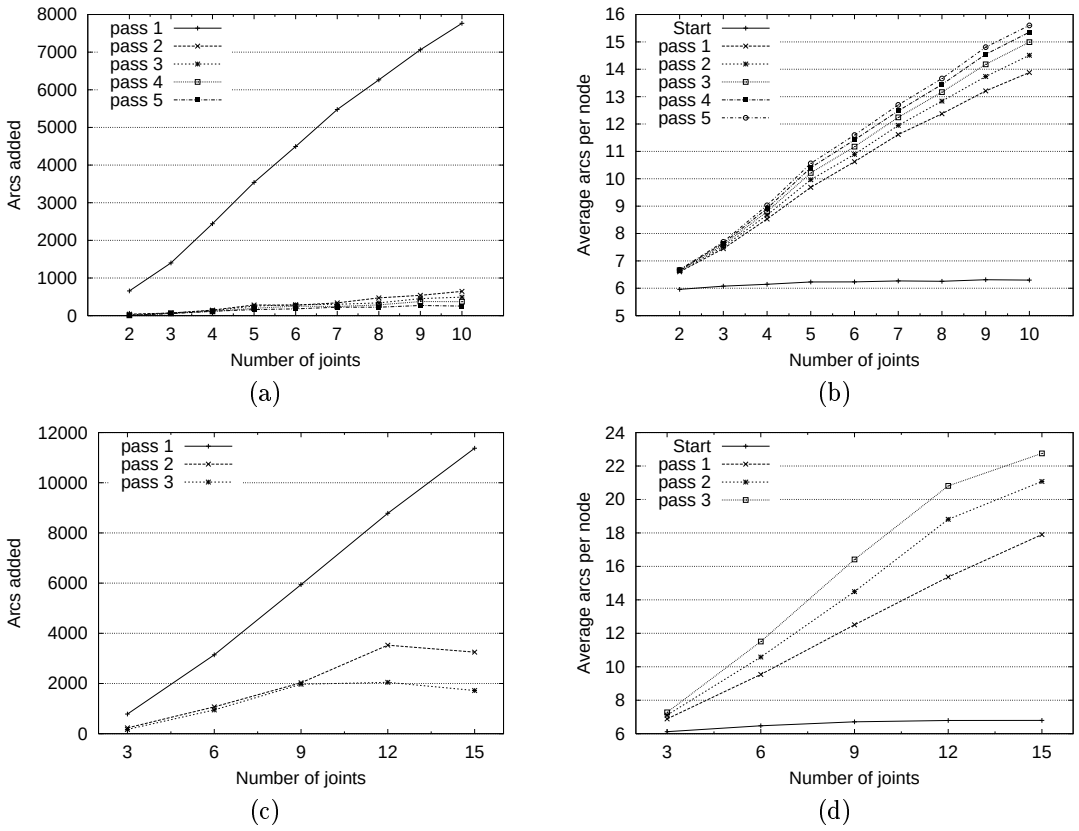


Figure 6.21 Arcs added and resulting connectivity at different steps during processing for ϵ-robustness for 2048-node roadmaps constructed with the $\mathcal{D}_{m2}^{\mathcal{W}}$ distance functions: (a) and (b) planar robots, and (c) and (d) 3D workspace robots.

At each location of the cube, we test the connectivity of the collision-free portion of the roadmap, and attempt to repair any breaks. Each arc added to the roadmap is tested to ensure that the cell that disconnected the roadmap does not interfere with the new arc. We also reject arcs along which the robot collides with itself.

Shown in Figure 6.21 are the number of arcs added for each pass over the workspace and the resulting average connectivity of the roadmap measured in arcs per node. We tested planar robots with 2–10 joints, performing five passes over the workspace, and we tested 3D workspace robots with 3-15 joints, performing three passes. The robot type for the 3D workspace robots is the one shown in Figure 6.1(a).

As can be seen in Figure 6.21, we add most of the arcs in the first pass. An explanation for this is that, in the first pass over the workspace, we find all of the "fragile" pieces of the roadmap that can be easily disconnected. The arcs we add to repair these pieces then reduce the number of additional arcs needed for the larger cube obstacles. The result after processing is a near-linear dependency between the number of arcs per node and the number of joints.

Table 6.3 The effect on ϵ-robustness of increasing the number of arcs per node in the initial roadmap for 3D workspace robots and roadmaps with different numbers of nodes.

	Arcs	Breaks			Arcs	Breaks		
	per	2048	8192	16 384	per	2048	8192	16 384
Joints	Node	Nodes	Nodes	Nodes	Node	Nodes	Nodes	Nodes
3	5	6109	17 339	26 291	8	1583	3200	4703
6	5	18 501	37 211	50 991	11	3386	7460	16 058
9	5	30 618	60 643	76 563	14	5787	16 283	
12	5	38 166	68 641	86 116	17	7847		
15	5	44 943	72 653	88 631	20	9522		

This last observation leads us to examine the effect on ϵ-robustness if we construct the roadmap such that the number of arcs per node increases linearly with the number of joints. Some results for this are shown in Table 6.3. The data in this table are for the robot type shown in Figure 6.1(a) and for roadmaps constructed using the $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function. As shown in the table, there is an improvement in the ϵ-robustness of the roadmap when the number of arcs per node is increased. There is also a pitfall: some elements of the table are missing because the program computing ϕ_a for the roadmap exceeded the available memory of the computer. We will say more later in this section about the effect on ϵ-robustness of roadmaps that are initially created with more arcs.

In the remainder of this section, we focus on robots with a 3D workspace. We also examine the effect of using the manipulability measure to bias the sample distribution in the configuration space. For this purpose, we examine two robot types, one being the robot in Figure 6.1(a), for which we have already presented some results above. The other robot type is the one shown in Figure 6.1(b). The inspiration for this robot comes from problem cell case II in Figure 4.5 and the robot modification shown in Figure 4.7 that overcomes this case of problem cells. By shifting the offset between the second and third links of each triple of links from the "inside" to the "outside," we eliminate this case of problem cells for the robot. This change also eliminates a type of problem cell that was observed in Figure 4.7. In the following, we label the type of robot shown in Figure 6.1(a) as type "A" and the robot shown in Figure 6.1(b) as type "B."

For the remaining results for ϵ-robustness shown in this section, we use the following set of parameters. The evaluation process for each roadmap consists of sweeping a cube of side length one over the workspace of each robot. When enhancing the roadmap, we use three passes, starting with a cube of side length one and finishing with a cube of side length three. For each location of the cube where the roadmap becomes disconnected, we limit the number of connection attempts between pairs of roadmap pieces to 100 (we used larger limits for the earlier examples). The effect of such a low limit on the number of connection attempts is to reduce the number of cells for which we can enhance the robustness of the roadmap.

Table 6.4 Results for ϵ-robustness for a 2048-node roadmap created for the type A robots.

	Low Manipulability			High Manipulability		
	Breaks		Arcs	Breaks		Arcs
Joints	Before	After	Added	Before	After	Added
3	6476	55	1218	6065	61	1113
6	17 073	522	5454	19 406	266	4798
9	29 449	1265	10 298	32 058	1081	9163
12	36 592	3303	14 153	42 585	1753	12 915
15	41 426	5921	15 777	47 908	3685	14 960

Table 6.5 Results for ϵ-robustness for a 2048-node roadmap created for the type B robots.

	Low Manipulability			No Manipulability			High Manipulability		
	Breaks		Arcs	Breaks		Arcs	Breaks		Arcs
Joints	Before	After	Added	Before	After	Added	Before	After	Added
3	5893	0	1022	6233	0	999	5198	4	863
6	18 426	86	5047	18 347	87	4639	18 005	53	4143
9	32 505	315	9156	33 853	443	8478	32 667	213	7392
12	39 663	627	11 777	42 014	547	11 024	44 903	446	10 160
15	45 863	1010	13 774	51 454	880	12 863	54 543	765	12 074

It also reduces the computation time needed for enhancement (we say more about computation time below).

We tested three sets of roadmaps, one set with the samples concentrated in regions of low manipulability, one with the samples concentrated in regions of high manipulability, and one unbiased set. To take into account the joint limits in the manipulability, we declared the manipulability to be zero if any joint of the robot was within a factor of 0.016 from a joint limit. For each roadmap, we take 2048 samples and use the $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function to select the arcs. Except where stated otherwise, each node is connected to at least its five nearest neighbors.

Shown in Tables 6.4 and 6.5 are the results for the robot types A and B, respectively. Shown in these tables are the number of cells in the workspace that disconnect the roadmap before and after enhancement, and the number of arcs added by enhancement.

There are two interesting patterns that can be observed in these tables. First, before enhancement, the roadmaps for the type A robots are disconnected less often than the roadmaps for the type B robots. On the other hand, after enhancement, the roadmaps for the type B robots are disconnected less often, and fewer arcs are needed to repair the connectivity. We expected that the roadmaps for the type B robots would be more robust because these robots have fewer problem cells in the workspace that disconnect the configuration space of the robot. This expectation holds after the enhancement step.

Table 6.6 Number of arcs in 2048-roadmap for which the robot collides with itself while following the path corresponding to the arc.

	Robot type A		Robot type B		
Joints	Low	High	Low	None	High
3	0	0	0	0	0
6	1062	154	566	346	104
9	2726	806	1896	1138	576
12	4592	1634	2730	2266	1086
15	5938	2794	3832	3082	1650

The second interesting pattern that can be observed in these tables is the effect of using the manipulability as a criterion for sampling the configuration space of the robot. For both robot types, the roadmaps created with nodes biased toward high manipulability were more robust after enhancement, and the enhancement required fewer arcs to produce this result. On the other hand, before enhancement, the roadmaps created with nodes biased toward low manipulability tended to be more robust, particularly for robots with more joints.

An exception to both of the above patterns is the robot with three joints: the roadmaps for the type B robots are more robust than those for the type A robots, both before and after enhancement. Indeed, the roadmaps for the type B can be enhanced to the point that no single cell in the workspace disconnects the roadmap, but this is not true for the type A robots. One reason for this is that the type B robots do not have the case II problem cells shown in Figure 4.5; these cells account for 27 of the cells that disconnect the roadmap. The design of the type B robots also eliminates some other pathological cells; an example of the type of configuration obstacles associated with these pathological cells was shown in Figure 4.7(b).

For the three-joint robots, there is also a different response to the manipulability bias on the sampling of the configuration space: before enhancement, the high-manipulability roadmap is more robust, but after enhancement, the low-manipulability roadmap is more robust. Enhancement adds fewer arcs to the high-manipulability roadmap, however. We have observed a similar result for planar robots with two joints, namely, that by biasing the sampling of configuration space to regions of low manipulability, we are able to create a roadmap that is robust to single-cell obstacles after enhancement.

To gain some insight into the ϵ-robustness results discussed above, we examine an issue that we have to this point largely ignored, namely the presence of arcs in the roadmaps where the robot collides with itself while following the path corresponding to the arc. Shown in Table 6.6 is a count of the number of arcs in the roadmap where this is true; these were computed by testing for self-collision each path corresponding to an arc in the roadmap. There are no such paths for the three-joint robots because there are no regions of their configuration spaces where they are in self-collision.

Table 6.7 Arcs rejected due to self-collision during ϵ-robustness enhancement.

	Robot type A		Robot type B		
Joints	Low	High	Low	None	High
3	0	0	0	0	0
6	54 244	15 603	3530	2576	612
9	135 329	53 709	22 946	17 112	5357
12	294 809	81 450	40 771	34 229	17 655
15	383 661	138 762	71 824	53 200	20 476

As shown in Table 6.6, the number of arcs with self-collisions is larger for sampling distributions biased toward low manipulability. An explanation for this is that the regions of the configuration space where self-collision occurs also tend to be regions of low manipulability. We saw this earlier in Figure 2.3, in which the peak of probability distribution of the manipulability of samples of the configuration space shifts to higher values when samples in which the robot collides with itself are excluded from the probability distribution. Having more samples in regions of low manipulability, then, also means that there is a greater potential for the arcs connecting these samples to have self-collisions. In addition, the number of these arcs increases with the number of joints because the fraction of the configuration space in which self-collision occurs also increases with the number of joints.

Another measure of the problem with self-collisions is shown in Table 6.7, which shows the number of arcs rejected due to self-collision during the ϵ-robustness enhancement process. Recall that the ϵ-robustness enhancement process rejects arcs with self-collisions, whereas the roadmap construction process does not. Here we see a clear difference between the two classes of robots and the different sample distributions: the enhancement process has greater difficulty in finding self-collision-free arcs for the type A robots and for roadmaps whose nodes are concentrated in regions of low manipulability.

The presence of arcs with self-collisions in the roadmap and the difficulty in finding self-collision-free arcs during the enhancement process suggest an explanation for the results we observed in Tables 6.4 and 6.5. In particular, the presence of arcs in the roadmap with self-collisions allows the initial roadmap to appear more robust for the roadmaps for the type A robots and the roadmaps with nodes concentrated in regions of low manipulability. These arcs with self-collisions provide a means for some of the nodes of the roadmap to remain connected to the rest of the roadmap, when, without these arcs, they might not be connected for some cells in the workspace.

In addition, with fewer arcs rejected due to self-collision during enhancement, the roadmaps for the type B robots and the roadmaps with nodes concentrated in regions of high manipulability can be made more robust by enhancement. An implication of fewer arcs being rejected is that the nodes in the roadmap are farther from the self-collision region of the configuration space, and this reduces the constraints on the set of arcs that may be added. This reduction in the constraints on the set of arcs

Table 6.8 The results for creating roadmaps with the number of arcs per node proportional to the number of joints in the robot.

Joints	Arcs per Node	Breaks		Arcs Added
		Before	After	
3	8	1127	0	426
6	11	3292	83	2356
9	14	5744	241	4498
12	17	7291	526	5759
15	20	8299	792	6577

has two effects, the first being that more of the fragile portions of the roadmaps can be repaired, and the second being that fewer arcs will be needed to effect these repairs.

Shown Table 6.8 is a continuation of the examination of the effect of creating the roadmap with the number of arcs per node proportional to the number of joints. The robot for this example is the type B robot shown in Figure 6.1(b), and the 2048 nodes of the roadmap were concentrated in regions of low manipulability. As with the earlier case, there is an improvement in the ϵ-robustness of the roadmap when the number of arcs per node is increased. Comparing these data with those in Table 6.5 for the low-manipulability case, we see that there are fewer cells in the workspace that disconnect the initial roadmap, fewer arcs are needed in the enhancement phase, and fewer cells disconnect the roadmap after enhancement. Unfortunately, the degree of improvement, particularly after ϵ-robustness enhancement, does not seem to justify the added cost in creating the initial roadmap and computing ϕ for that roadmap. This suggests that the arcs added as a result of enhancement are not the same as those added when the number of nearest neighbors is increased for the initial roadmap.

A drawback to processing a roadmap to increase its ϵ-robustness is the computation time required for this process. The time required for the test procedure, which finds the number of cells that result in the roadmap becoming disconnected, is proportional to the product of the number of nonempty cells in the workspace and the size of the roadmap. The procedure that repairs the roadmap for each of the cells that disconnect the roadmap adds the cost of evaluating the arcs needed to reconnect the pieces of the roadmap. This last step dominates the computation time for enhancing the ϵ-robustness of a roadmap.

For example, the test procedure, which finds the number of cells that result in the roadmap becoming disconnected, took 35 min for a 2048-node roadmap for the 3D workspace robots. The times for the 8192 and 16 384 node roadmaps were 3.5 and 8.3 h, respectively. The test times were shorter for the planar robots, since there are fewer cells to test (around 15 000 instead of 300 000). The program that improves the robustness of the roadmap can easily take several days or more, particularly for larger roadmaps and robots with more joints.

6.5 Some Planning Examples

The implementation of our planner tests for the existence of a path between two selected nodes in the roadmap modified to take into account obstacle constraints; it does not connect arbitrary configurations to the roadmap. In this section, we discuss the two different versions of the planner. Both of these planners use a search procedure based on A* search, and the user is allowed to select the distance function used as the heuristic to guide the search. In addition, the user may select a best-first search, which is implemented by setting to zero the estimates for the cost to reach the goal. First we discuss the planner for fixed-based serial-link manipulators, and then we discuss the planner for mobile robots.

6.5.1 Fixed-base robots

The process for finding a path for fixed-base robots consists of two steps: modifying the roadmap to take into account the obstacles and searching the modified roadmap for a path. The search process itself is fast, taking on the order of 10's of milliseconds to search the roadmaps that we have tested. This is true for both robots with 2D workspaces and robots with 3D workspaces. The path smoothing that we apply to the output of the planner takes a variable amount of time, spending more time when the robot is closer to the obstacles along the path and taking more time for robots with more joints.

Modifying the roadmap to take into account the obstacles is slower, and the time required depends on the neighborhood model used to encode ϕ. The fastest roadmap update times were for ϕ encoded with no neighborhood model, the "best" hierarchical neighborhoods had the second fastest update times, and the region-growing neigborhoods had the third fastest times. Nevertheless, the update times for all three of these were close in magnitude. This was not the case for the "union" hierarchical neighborhoods, whose performance was around an order of magnitude worse. This performance gap may be due to our implementation of the decoding procedure for the hierarchical neighborhoods, which decodes ϕ for a given cell from the bottom of the hierarchy to the top. It may be more efficient for this case to process the cells from the top of the hierarchy down.

Shown in Figure 6.22 is an example path generated by our planner for a 20-joint planar robot. The roadmap used for this plan had 8192 nodes. The time required to process the obstacles shown in Figure 6.22 was on the order of a few seconds for 2048- and 8192-node roadmaps and ϕ encoded using no neighborhood model, using the region-growing neighborhood model, and using the "best" hierarchical neighborhood model. On the otherhand, using the "union" hierarchical neighborhood model, processing the obstacles required on average 41 s for the 2048-node roadmap and 167 s for the 8192-node roadmap. These times did not vary much for robots with different numbers of joints. For the example path shown in Figure 6.22, path smoothing took around 12 s to perform.

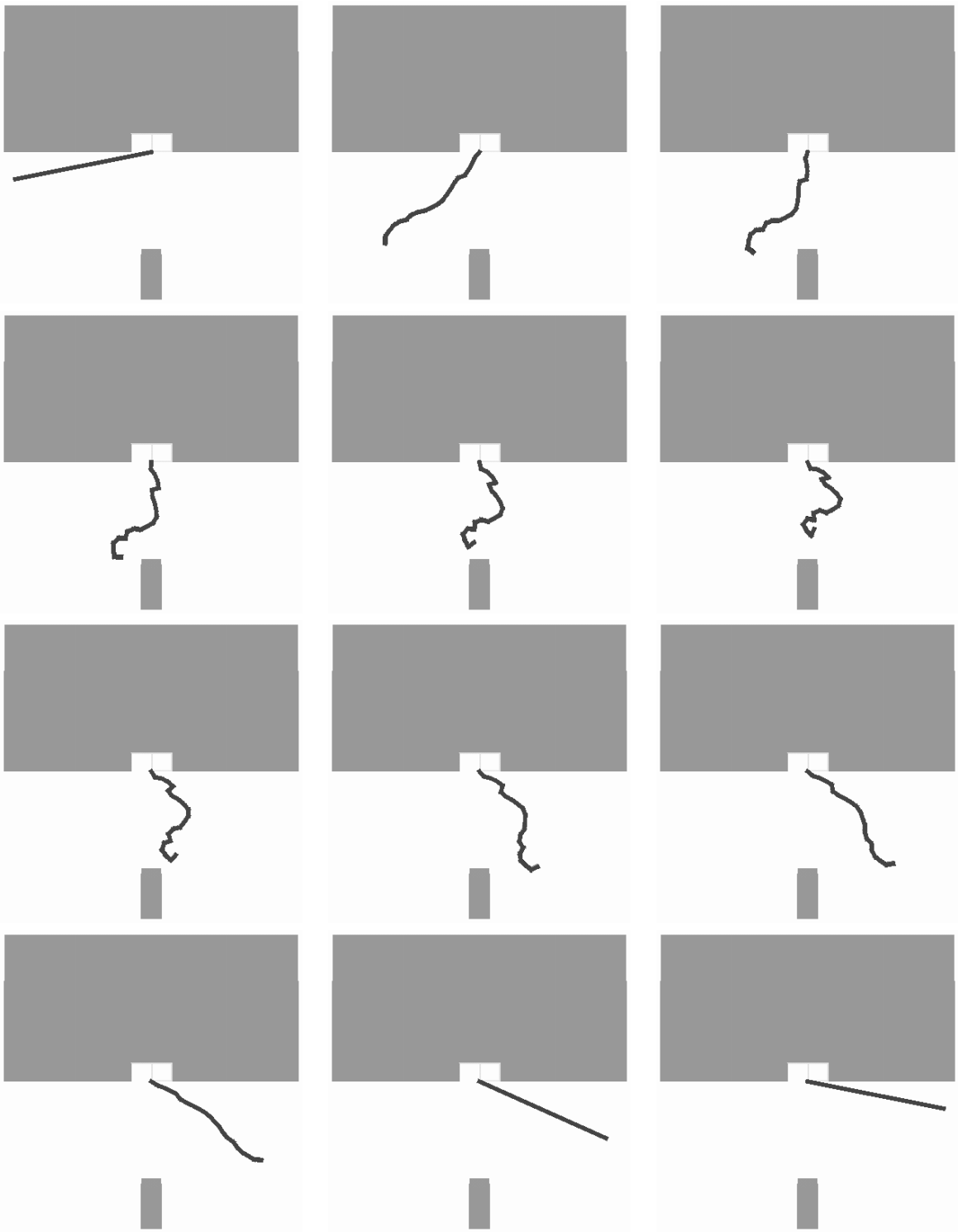


Figure 6.22 An example plan for a 20-joint planar robot with joint limits. The sequence of steps is from left to right, top to bottom.

Shown in Figure 6.23 is an example path generated by our planner for a six-joint robot with a 3D workspace and a 2048-node roadmap. We observed a pattern similar to the planar robot case for the processing time required for the different neighborhood models when processing the obstacles in the workspace. For the obstacles shown in Figure 6.23, the processing time was less than one second for the neighborhood models of none, region-growing, and "best" hierarchical, with exceptions for the 12 and 15 joints robots, where the time increased up to 2.8 s for the 8192-node roadmaps. The times for the "union" hierarchical neighborhood models were much slower, requiring around 10.5 s for the 2048-node roadmaps and around 43 s for the 8192-node roadmaps. In addition, the times tended to increase with the number of joints, which correlates with the increase in the size of the representation of ϕ with the number of joints.

6.5.2 Mobile robots

We have tested a restricted case for this planner in which we limit the region in which the base of the robot may move to a square 101 units on a side. Inside this region, we maintain all of the bitmaps used for obstacle avoidance simultaneously for each of the possible positions of the robot. We have imposed this restriction for three reasons. First, this bounds the region over which the planner operates, putting an upper limit on the path search time. Second, this allows us to examine the efficiency of maintaining the bitmaps in this manner. Third, this allows us to examine the time required for searching the state space of the robot independently of the time required to evaluate ϕ for each of the obstacles.

Shown in Figure 6.24 is an example path generated by our planner. Processing the obstacles in this environment required 76 s to update the bitmaps. Unlike our planner for fixed-base robots, for which the choice of the distance function used to guide the search has little effect on the search time, for the mobile robots, the search time depends strongly on the distance function. For example, for the path shown in Figure 6.24, the path search took 2 s when using the distance function defined on the configuration space. On the other hand, the search required 48 s when using the distance function defined in the workspace. In general, we have observed that using the distance function defined on the configuration space results in shorter computation times.

We have also observed that the search time depends on the difficulty in reaching the goal. For example, the goal configuration for the path shown in Figure 6.24 is relatively open compared to the initial configuration. If we switch the initial and goal configurations, the planning time increases to 315 s for the search based on the configuration-space distance function and 239 s for the workspace distance function. This increase in search time is due to the planner exploring more of its search space, examining more paths that lead to dead ends.

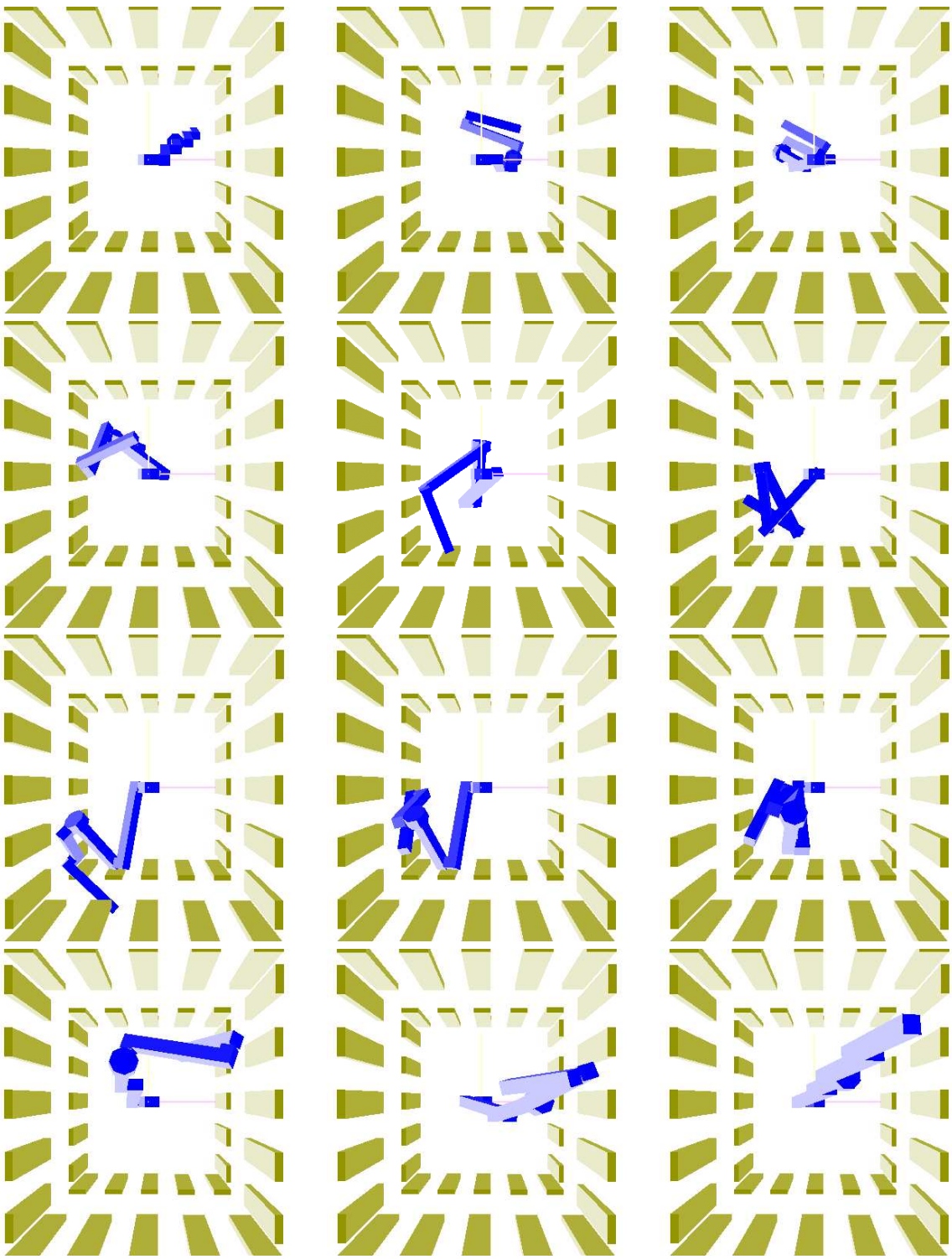


Figure 6.23 An example plan for a six-joint robot with a 3D workspace. The sequence of steps is from left to right, top to bottom.

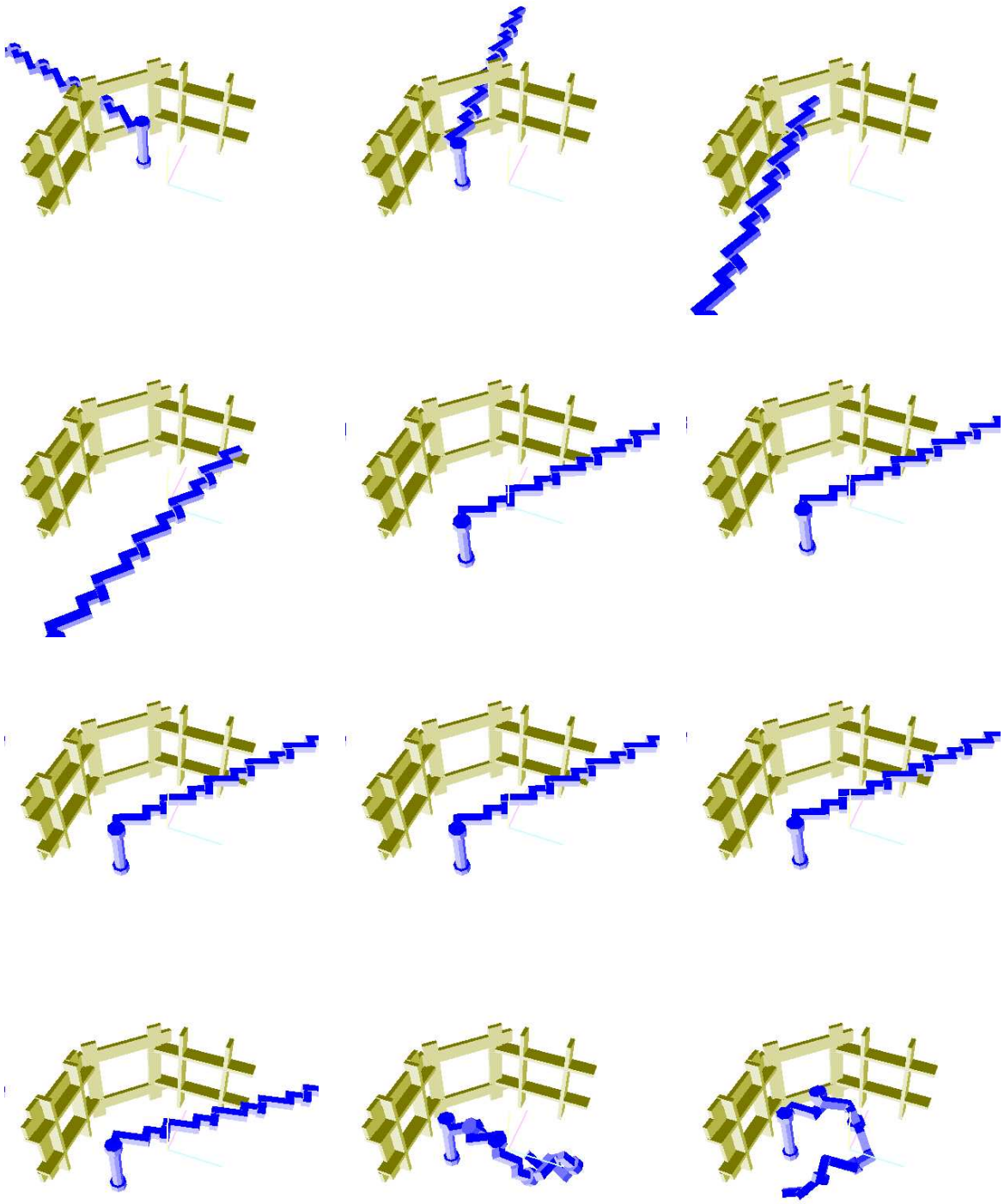


Figure 6.24 An example plan for a 15-joint mobile robot. The sequence of steps is from left to right, top to bottom.

CHAPTER 7

CONCLUSION

We have presented a new method for generating collision-free paths for robots operating in changing environments. This approach is closely related to recent probabilistic roadmap approaches, which use preprocessing and query stages and are aimed at planning many times in the same environment. In contrast, our preprocessing stage creates a representation of the configuration space that can be easily modified in real time to account for changes in the environment. As with previous approaches, we begin by constructing a roadmap in the configuration space of the robot, but for an obstacle-free environment. We then construct a mapping from the workspace to the roadmap in configuration space, and use this mapping to make modifications to the roadmap appropriate to the obstacles in the environment. Plans are then generated by searching the modified roadmap.

We began our presentation with a discussion of the construction of the roadmap used for path planning. We found that many of the existing sampling techniques used to generate the nodes of the roadmap were not applicable for our approach because these sampling techniques depend explicitly on the obstacles in the workspace. We then chose two sampling techniques to use in generating our roadmap. The first of these is based on uniform random sampling and has been used by some of the previous PRM approaches. The second sampling technique uses the manipulability measure to bias a uniform random sampler.

To define the arcs in the roadmap, we followed the approach used in many traditional PRM planning approaches. In particular, we used a local planner that generates straight-line paths in the configuration space, and we chose to define the arcs by connecting each node to a set of its nearest neighbors. We also discussed other local planners used by some PRM approaches, and some of the distance functions used by others to define the nearest neighbors. We then discussed the distance functions we have used to define the nearest neighbors, and methods for finding the nearest neighbors for each node.

We next discussed the core feature of our planning approach, the mapping from the workspace to the roadmap in the configuration space of a robot. Given the positions of obstacles in the environment, this mapping allows us to quickly determine the set of nodes and arcs of our roadmap that are no longer

valid due to the obstacles in the environment. We discussed both efficient techniques for computing this mapping, and efficient means for storing a representation of the mapping.

We then examined some techniques that we could use to enhance our roadmap. We examined the techniques others have used to enhance their roadmaps and noticed that these methods were not applicable to our approach due to their dependence on the obstacles in the environment. We defined two evaluation methods that are applicable to our approach: one based on the the minimum cut set of a graph and the other based on an analysis of the mapping from the workspace to the roadmap. For both of these analysis methods, we defined techniques for adding arcs to the roadmap to improve the roadmap.

We continued with an extension of our approach to mobile robots. This extension is based on the observation that the mapping from the workspace to the roadmap is invariant under translations of the robot base. Based on this invariance, we then defined a planner that partitions the path planning problem into two parts: one for the motion of the robot base, the other for internal motion (such as the motion of an arm attached to the base). We described how the state space of this robot may be searched, and how to take obstacles into account in this search.

We then presented some experimental results for a planner based on our framework. As part of our experiments, we examined serial-link manipulators for both 2D and 3D workspaces. For roadmap construction, we examined the effect of the geometry of the robot on the sampling of its configuration space, and we evaluated several techniques for computing nearest neighbors. For the mapping from the workspace to the roadmap, we examined the size of the mapping, the computation time required to generate the mapping, and the improvement in the size of the mapping gained by using a more efficient encoding. For roadmap enhancement, we examined the amount of improvement we can expect using either of the two techniques for a given robot and roadmap. We concluded our experimental results with some planning examples.

APPENDIX A

PLANAR ROBOTS WITHOUT JOINT LIMITS

For a real-time motion planner, we are interested in three parameters: graph update time, planning time, and data structure size. We are also interested in the preprocessing time, though this is less important. In this appendix, we evaluate a preliminary version of the planner, studying the case of serial-link manipulators operating in a 2D workspace. Note that the implementation is not limited to revolute joints, prismatic joints can be used as well. We limit the following discussion to revolute joints to simplify the analysis.

A.1 Experimental Setup

The following experimental setup is used to evaluate the approach. The robots tested are all serial-link manipulators with two to 20 revolute joints in a 2D workspace. The workspace cells and the robot are given 3D polyhedral descriptions to allow the use of standard collision checking packages for collision testing. Each cell of the workspace is modeled as a unit cube, and each link of the robot is modeled as a rectanguloid with a width of 2.1 units and a height of 1 unit. The total length of the robot was fixed at 70 units, with each link length being scaled appropriately. Each joint of the robot was given the full revolute range of motion, and the joints were allowed to wrap around. In addition, the robot was not tested for self-collision, except when the samples of configuration space were generated. To evaluate how the data structures and computation times grow with the graphs, we tested graphs with 2048, 8192, and 16 384 nodes.

A.2 Results for Planar Robots without Joint Limits

In this section we present some results from using the above setup. We start with an analysis of the data structures that are computed to implement our planning approach. Each of the four data structures—graph nodes, graph arcs, ϕ_n, and ϕ_a—was computed and stored separately to simplify analysis. We follow the analysis with some compression results for ϕ_n and ϕ_a, and we conclude with some planning examples.

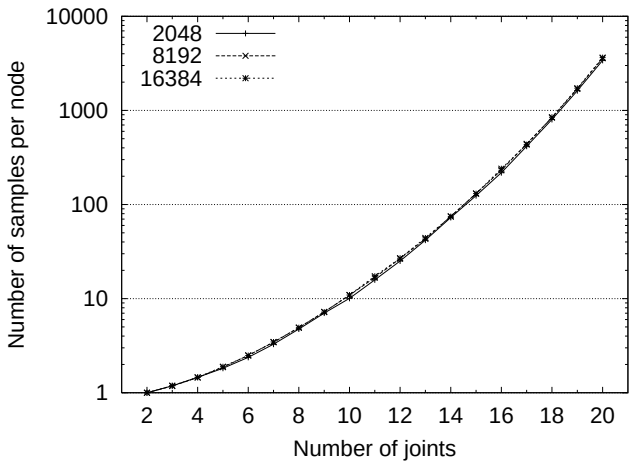


Figure A.1 Number of samples of configuration space generated per node accepted.

A.2.1 Configuration space sampling

We use two different sampling distributions for sampling $\mathcal{C}$ for each of the robots. The first 129 samples are distributed uniformly along a line parallel to the axis of the first joint in $\mathcal{C}$. The positions of the other joints are held fixed at a position that maximizes the length of the robot (see Figure A.8 for an example of what these samples look like). The remaining samples come from a uniform distribution over all of $\mathcal{C}$. Of this second set of samples, those in which the robot collides with itself are rejected.

It is interesting to note that, when sampling $\mathcal{C}$ uniformly, it becomes exponentially more difficult as the number of joints increase to find configurations in which the robot did not collide with itself. This is a consequence of a robot consisting of revolute joints operating in the plane: for each set of three links, there is a region of $\mathcal{C}$ in which the third link is in collision with the first. For each additional joint beyond three, there is a combinatorial effect with combinations of links in collision, as well as an interaction among these links that tends to make the collision regions larger.

The exponential sampling effect can be seen in Figure A.1, which shows the number of samples of configuration space generated per node accepted. Notice that the y-axis scale of this graph is logarithmic. As expected, the number of samples generated per node accepted does not change as the desired number of nodes increases.

A.2.2 Graph construction

As described above, arcs in the graph are constructed for the k nearest neighbors of each node. For our evaluation, we tested the case of $k = 5$ nearest neighbors. We computed the nearest neighbors using the simple $O(n^2)$ method of calculating the distance between all pairs of nodes and keeping the closest

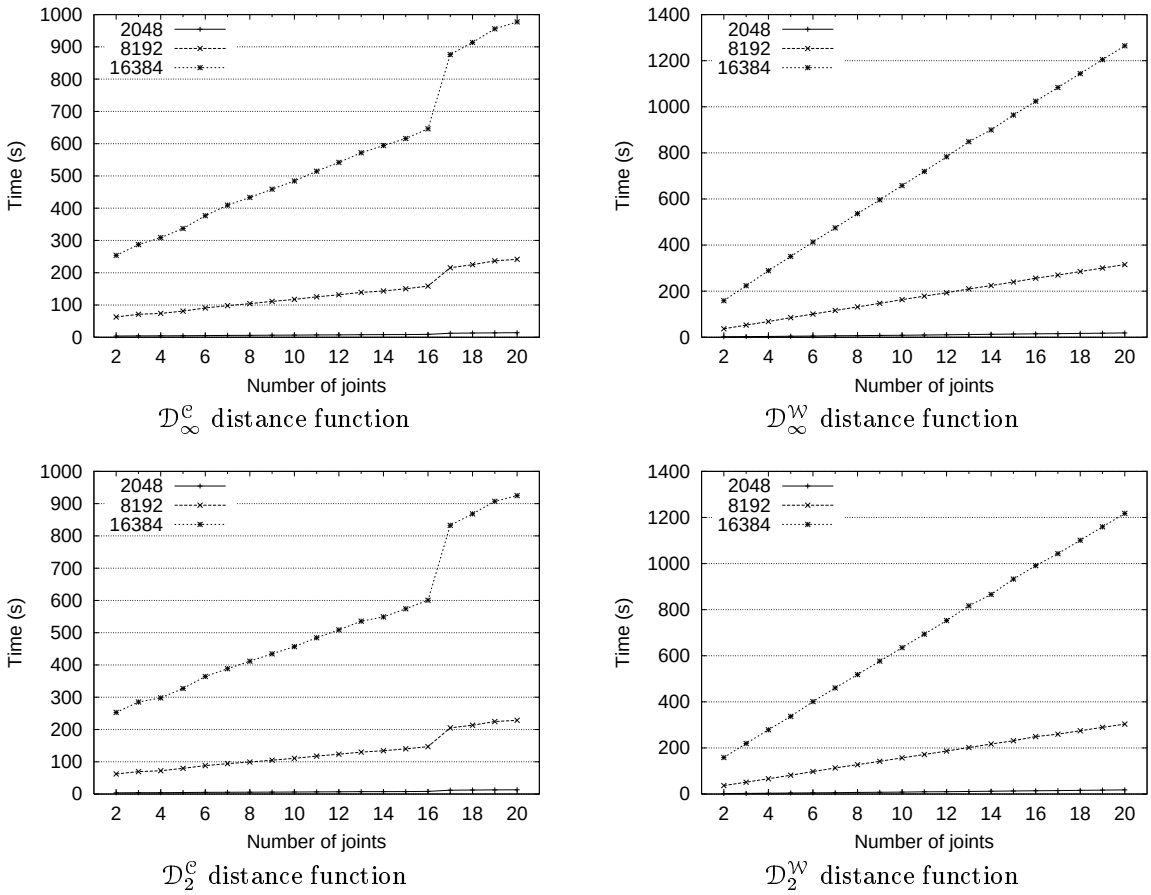


Figure A.2 Time in seconds to compute the k nearest neighbors for the different distance functions.

five for each node. The paths connecting these neighbors are not checked for feasibility, that is, whether the robot avoids self-collision while following the paths.

The computation times for computing the graph are shown in Figure A.2. The increase in computation time with the number of joints reflects the increase in the cost of computing the distance function. The sharp rise in the computation time after 16 joints seen in the graph for the $\mathcal{D}_\infty^{\mathcal{C}}$ and $\mathcal{D}_2^{\mathcal{C}}$ distance functions is unexpected; this may be due to a memory effect in the implementation, where the amount of memory used for the calculation crosses some system threshold.

What also can be seen in Figure A.2 is that there is not much difference in computation time between the 2-norm and the ∞-norm, except that the computations involving the ∞-norm tend to take somewhat more more time to perform (this is likely an artifact of the processor architecture, in which the cost of a branch is greater than the cost of a floating-point addition). A greater difference can be seen between the distance function in configuration space and the distance function in the workspace, where the latter

appears to have a more linear increase in computation time as the number of joints increases. Overall, for robots with more than five joints, the workspace distance function takes more time to compute.

The size of the graph varies linearly with the number of nodes, with an average size of 163 kbytes for 2048 nodes, 654 kbytes for 8192 nodes, and 1309 kbytes for 16 384 nodes. The variance in size from the average was less than 15%.

A.2.3 Computing ϕ_n

The overall size of ϕ_n is the product of the number of nodes and the average number of cells the robot covers in the workspace. For a robot with a fixed maximum length, this means that the size of ϕ_n is largely independent of the number of joints. For our experiments, the size of ϕ_n per node in $\mathcal{G}_n$ averages around 1000 bytes.

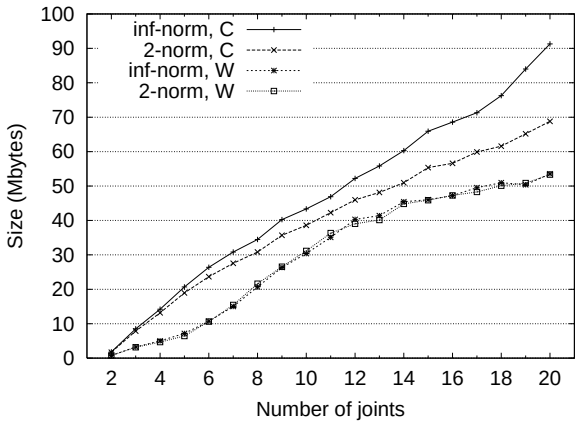
As the number of nodes increases, there is a sublinear increase in the size of ϕ_n that is a result of the overhead for maintaining the data structure being amortized over more nodes. The overhead for the data structure depends on the number of cells in the workspace that intersect with the robot at any node in the graph. This overhead does not depend on the number of nodes in the graph, as long as the graph contains nodes that span the reach of the robot in the workspace (that is, the graph contains nodes that touch all the cells in the workspace that the robot can reach).

The time required per node to compute the ϕ_n increases linearly as the number of joints increases. This is expected because the size of the geometric description of the robot used for collision testing increases with the number of joints.

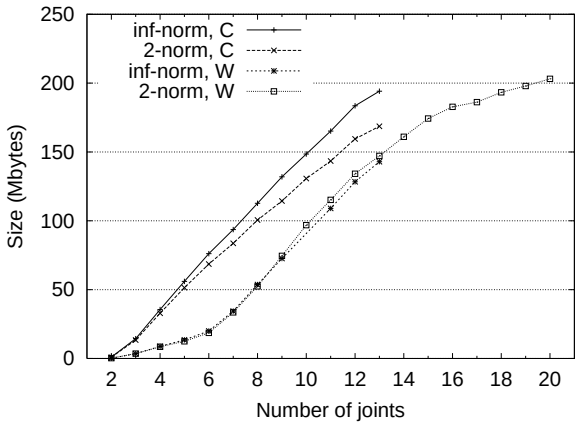
A.2.4 Computing ϕ_a

The size of ϕ_a for different numbers of nodes is shown in Figure A.3. As can be seen in the figure, ϕ_a can become quite large. This is the reason that the graphs stop at 13 joints for the 8192 node case, and 8 joints for the 16 384 case for roadmaps constructed with three of the distance functions. The limit in this case was the process size of the program computing ϕ_a; in order to keep the computation times comparable, the memory size of the program computing ϕ_a was limited to the physical memory available in the computer. More examples were computed for the $\mathcal{D}_2^{\mathcal{W}}$ distance function for the 8192 and 16 384 nodes cases in order to evaluate the compressibility of the resulting data files (see Section A.2.5).

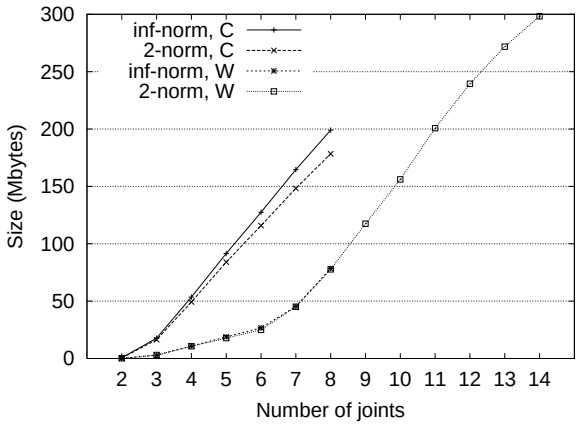
It is also apparent that the $\mathcal{D}_\infty^{\mathcal{C}}$ performs the worst in terms of the size of ϕ_a. The $\mathcal{D}_2^{\mathcal{C}}$ is the next worst, and the $\mathcal{D}_\infty^{\mathcal{W}}$ and $\mathcal{D}_2^{\mathcal{W}}$ produce the best results. It can be expected that the two distance functions defined on the workspace would produce better results than the distance functions defined on the configuration space, because the workspace distance functions penalize the motion in the workspace more than do the configuration space distance functions. There is no apparent advantage to $\mathcal{D}_2^{\mathcal{W}}$ over $\mathcal{D}_\infty^{\mathcal{W}}$.



2048 nodes

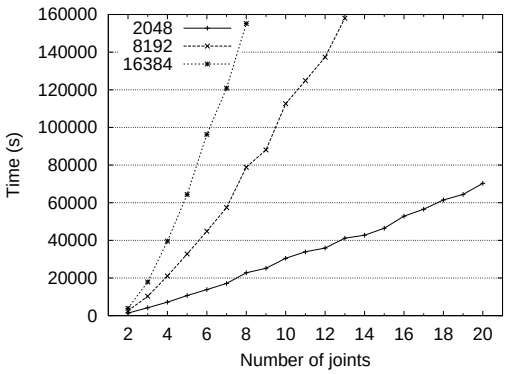


8192 nodes

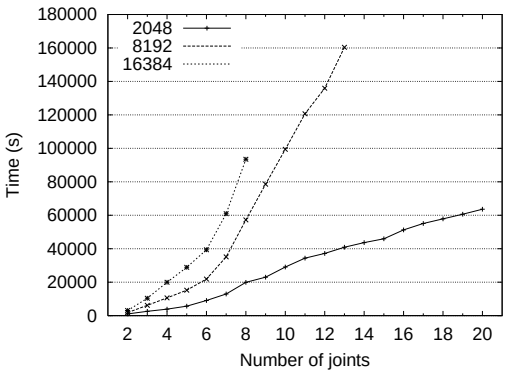


16 384 nodes

Figure A.3 Size of ϕ_a.

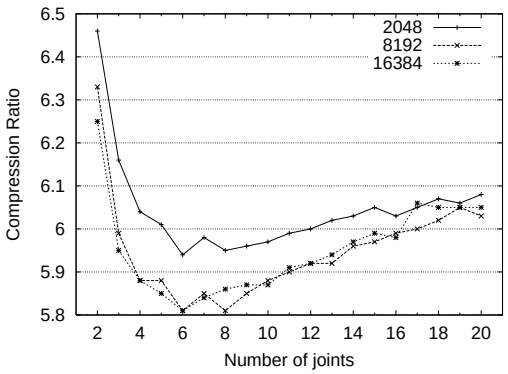


$\mathcal{D}_2^{\mathcal{C}}$ distance function

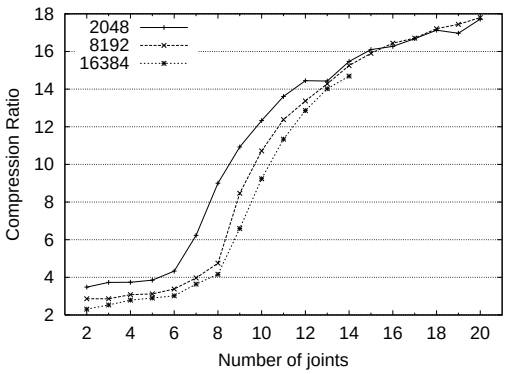


$\mathcal{D}_2^{\mathcal{W}}$ distance function

Figure A.4 Computation time for ϕ_a for two distance functions.



(b) ϕ_n



(b) ϕ_a

Figure A.5 Compression ratios.

Shown in Figure A.4 is the time it takes to compute ϕ_a for the $\mathcal{D}_2^{\mathcal{C}}$ and $\mathcal{D}_2^{\mathcal{W}}$ distance functions (the $\mathcal{D}_\infty^{\mathcal{C}}$ and the $\mathcal{D}_\infty^{\mathcal{W}}$ produce similar results to the $\mathcal{D}_2^{\mathcal{C}}$ and $\mathcal{D}_2^{\mathcal{W}}$ distance functions, respectively). These times correlate well with the increase in the size of the respective data structures.

A.2.5 Some compression results

Shown in Figure A.5 are the compression ratios for ϕ_n and ϕ_a that were achieved by differential encoding of the cells of the workspace and by using a more efficient encoding of the labels used to represent the nodes and arcs in the graph (roughly a factor of two of the compression ratio comes from this change in representation). The compression ratios are relatively flat for ϕ_n as the number of joints changes.

To gain some insight into the behavior of the compression ratios achieved for ϕ_n, a set of images showing the distribution of $|\phi_n(w)|$ over the workspace was generated. Figure A.6 shows four such images, showing the case of the 2048-node graphs with 2, 4, 10, and 20 joint robots. In the figure, the grid lines demark 11x11 collections of workspace cells, and the darkness of a cell is proportional to the logarithm of $|\phi_n(w)|$. Cells that are white have $|\phi_n(w)| = 0$, the lightest gray corresponds to $|\phi_n(w)| = 1$, and black corresponds to $|\phi_n(w)| = 2048$ (the eight neighbor cells of the origin are always obstacles to the robot). In the images for the two- and four-joint robots, the reach of the first link of the robot is readily apparent as a darker disk in the center of the image. This disk is still present in the images for the 10- and 20-joint robots, but is much less visible. No other joint produces similarly visible disk. Some variation in darkness can be seen in the lighter regions.

As the number of joints in the robot increases, the ring corresponding to the first link becomes smaller, and the links themselves have a greater tendency to cluster near the origin due to the uniform random sampling. This clustering also means that clearly visible rings corresponding to links of the robot other than the first do not appear. The clustering also has an effect on the compressibility of the ϕ_n. At two joints, the distribution of $|\phi_n(w)|$ is relatively uniform, compared to robots with more joints. A greater uniformity of coverage of the workspace implies greater spatial coherence between neighboring cells, and that allows for greater compression. At the 20 joints, the distribution tends to be highly clustered near the origin, allowing for higher compression as the cells outside the cluster near the origin are grouped together in larger groups. In between the two extremes, the compression is lower.

On the other hand, the compression ratios for ϕ_a increase with the number of joints, as shown in Figure A.5(b). This is expected because the average length of the paths corresponding to arcs in the graph tends to increase with the number of joints, and that in turn leads to an increase in the volume of the workspace swept by the robot as it traverses the paths. As more cells are affected by the longer paths, the spatial coherence of the cells in the workspace increases. Notice that the inflection point seen around eight joints in this graph is similar to that seen in Figure A.3. The same effect that minimizes the size of ϕ_a also reduces spatial coherence and, therefore, the achievable compression ratio.

Shown in Figure A.7 is the distribution of $|\phi_a(w)|$ per cell for the two-link robot and the 2048 and 16 384 node graphs. In both of these graphs, the darkness is proportional to the size of $|\phi_w(w)|$, with black corresponding to $|\phi_a(w)| = 39$. Both of these graphs also have the same grid spacing as Figure A.6. The sparseness of the distribution of the arcs in this case is the reason for the reduction in the compression ratio available using the differential encoding. Also shown in the figure is the effect of increasing the size of the graph from 2048 to 16 384 nodes. As noted earlier, the average distance between nodes decreases, decreasing the average swept volume of the paths corresponding to the arcs in the graph and, in this case, dramatically increasing the sparseness of ϕ_a.

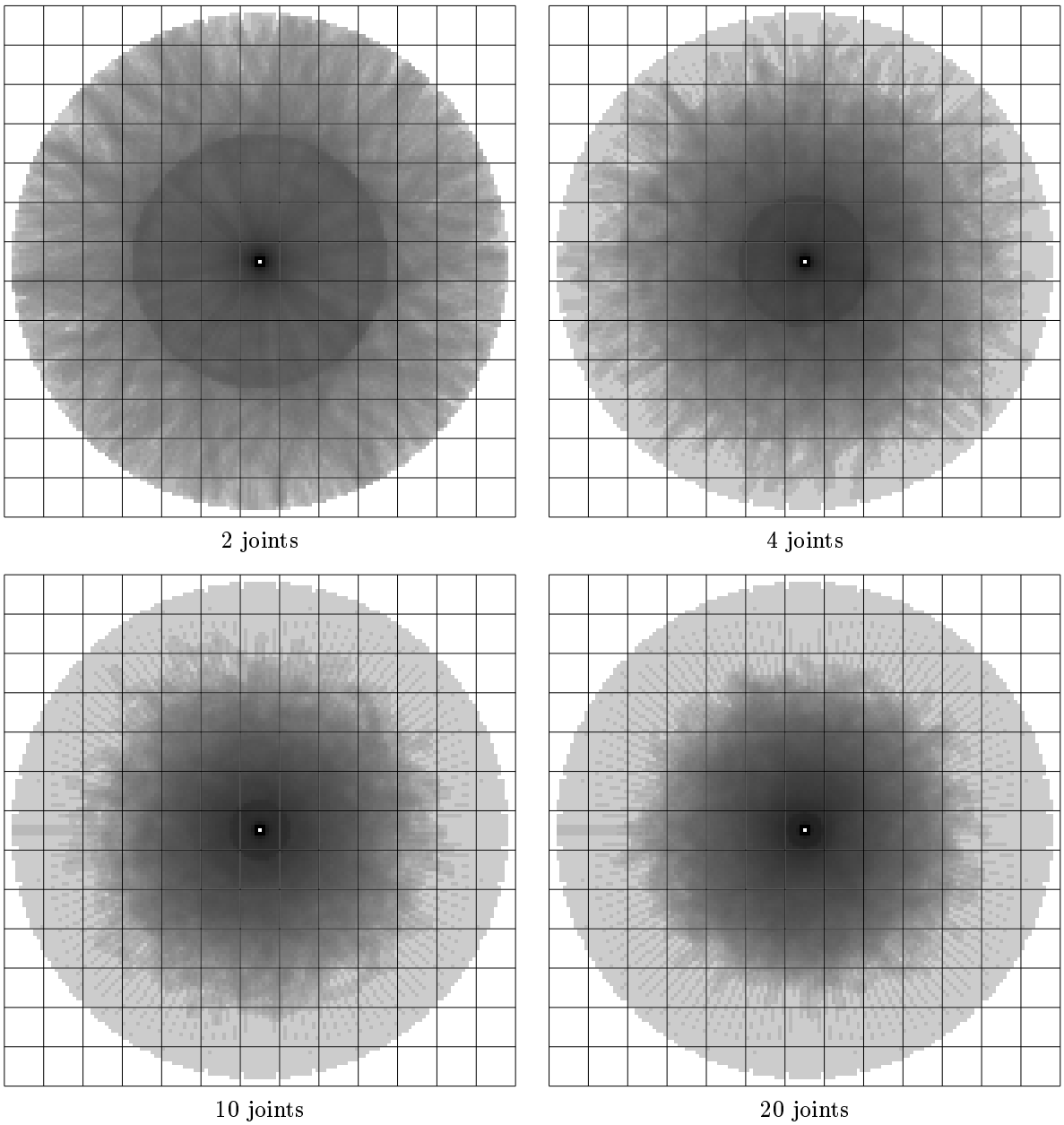
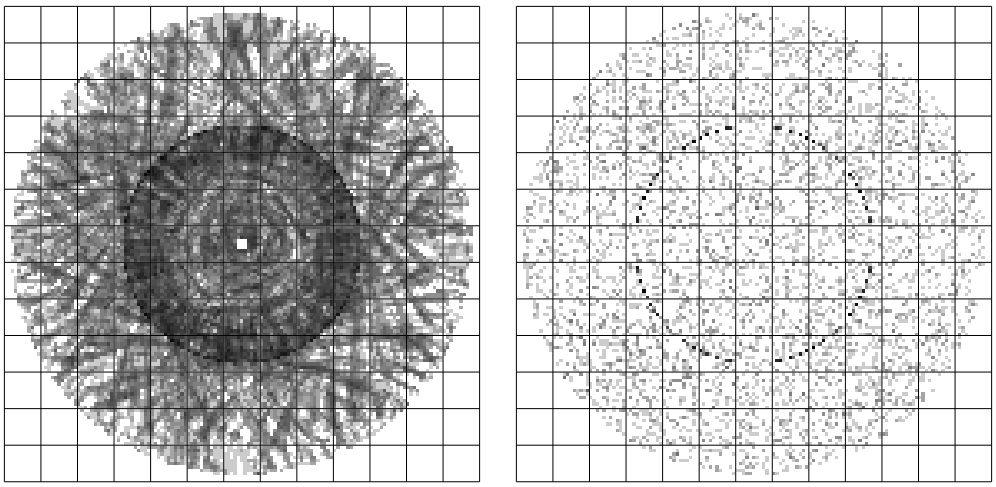


Figure A.6 $|\phi_n(w)|$ per cell for different robots and a graph with 2048 nodes.



2048 node graph 16 384 node graph

Figure A.7 $|\phi_a(w)|$ per cell for a two-link robot.

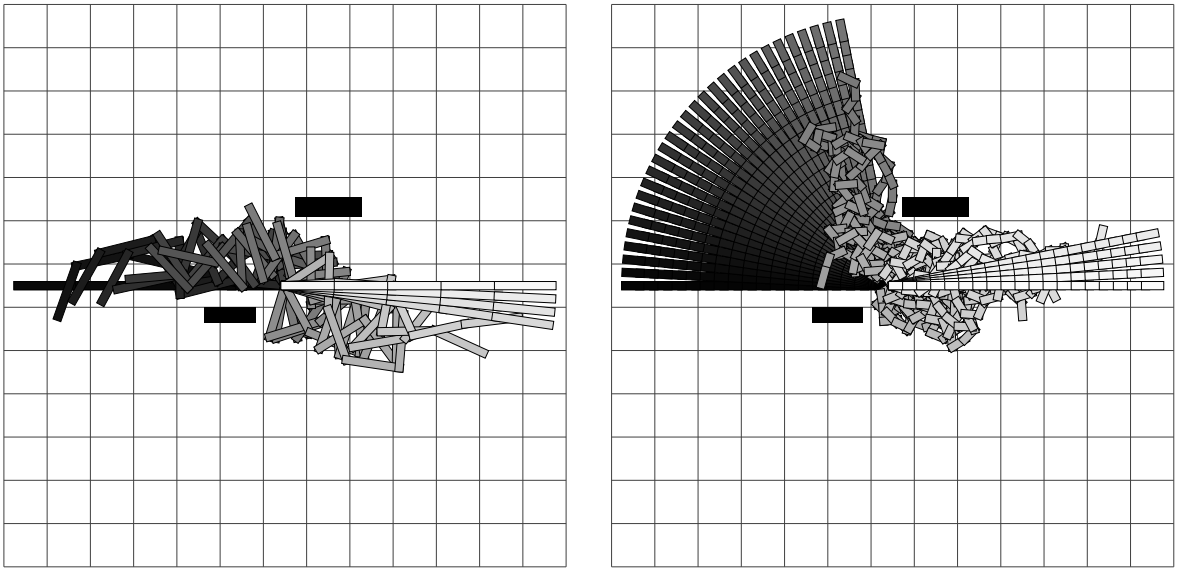
A.2.6 Some planning examples

This preliminary implementation of our path planner tests the connectivity of the graph between two selected nodes in the modified graph; it does not connect arbitrary configurations to the graph. In all of the planning runs tested, it takes less than 1 s to update $\mathcal{G}_n$ and $\mathcal{G}_a$ using the uncompressed representations of ϕ_n and ϕ_a and to find a path in the modified graph. Some preliminary experiments using the compressed representation of ϕ_a and ϕ_n have also been performed; in these, the time to update the roadmap increases up to 15 s when large regions of the workspace are occupied by obstacles.

Shown in Figure A.8 are plans for two robots negotiating between two obstacles (black objects in the figure). Both plans were generated using a best-first search of the 2048-node roadmap, and the steps shown in the figure correspond to the nodes in the graph along the plan. No path smoothing was performed for either of these two cases. For this environment and 2048 nodes, only about one-third of the 76 roadmaps (19 different robots, 4 distance functions for each robot) had a connection that could solve this problem (note that, for robots with two to four joints, the length of the first link precludes the possibility of a plan). When the number of nodes in the roadmap was increased to 8192, most of the roadmaps has a connection that could solve the problem.

A.3 Problems

As with any motion planning system, there are certain pathological cases that cause problems with our approach. The first of these is a mismatch between the local planner and the distance function. One of the worst-case examples of this is shown in Figure A.9, where the $\mathcal{D}_2^{\mathcal{W}}$ distance function is used



(a) 5 joints (b) 19 joints

Figure A.8 Plans for two robots passing through a (relatively) narrow corridor. The dark blocks are the obstacles.

to construct the roadmap, the local planner is a straight line in configuration space, and the robot has 10 joints, with the last nine joints having limited range. As computed by the distance function, the two configurations shaded in light gray are close together, but the path generated by the local planner sweeps out a large region of the workspace. The problem with such a connection in the graph is that, with the addition of any obstacle in the workspace, it is highly likely that this path would be blocked.

Since we are using two different sampling distributions to generate the nodes of the roadmap, another problem is that these two distributions are not well connected, particularly as the number of joints of the robot increases. An example of this can be seen in Figure A.8 for a 19-joint robot and a 2048-node graph. Indeed, most of the configurations in the "fan" are not connected to any of the configurations generated at random, which means that it will be easy to break the connection between the two parts of the roadmap. In addition, some of the roadmaps were generated with two connected components because of this problem. This demonstrates the need for an enhancement phase to the roadmap construction, such as the one described in Chapter 4.

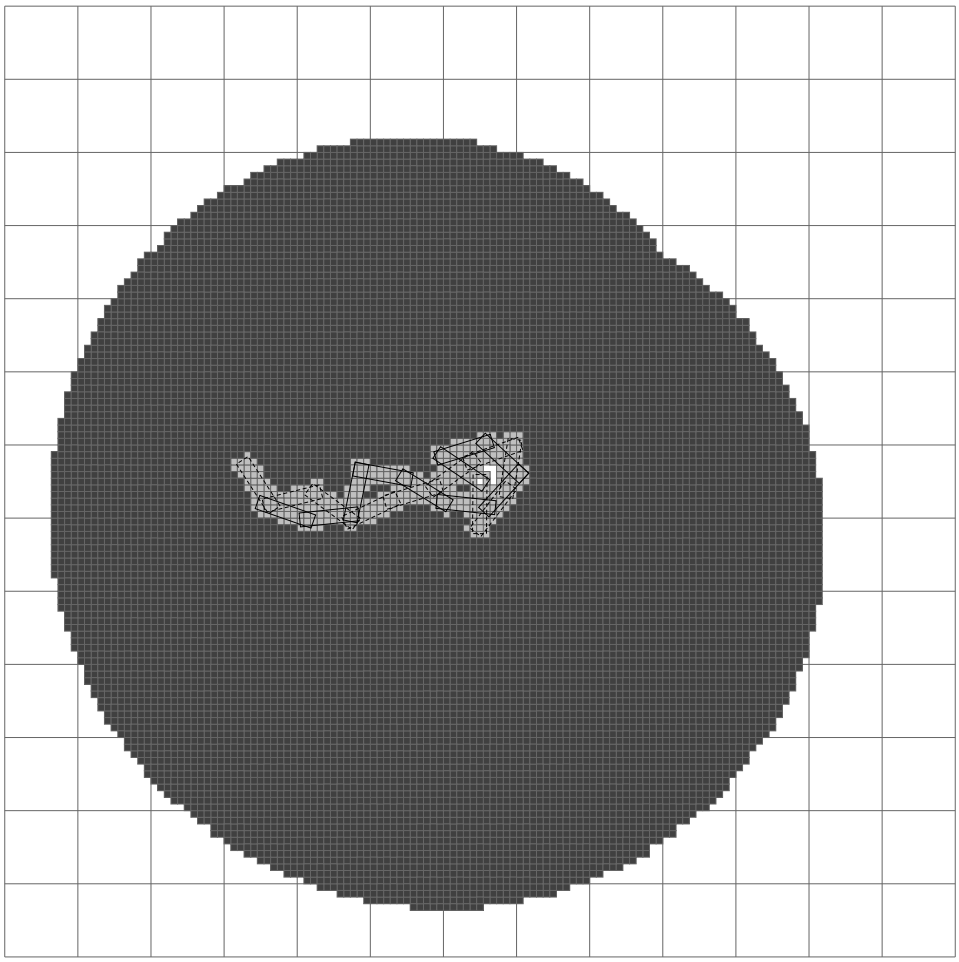


Figure A.9 A problem with the straight-line in configuration space local planner combined with the $\mathcal{D}_2^{\mathcal{W}}$ or the $\mathcal{D}_\infty^{\mathcal{W}}$ distance functions.

APPENDIX B

MANIPULABILITY OF ROBOTS

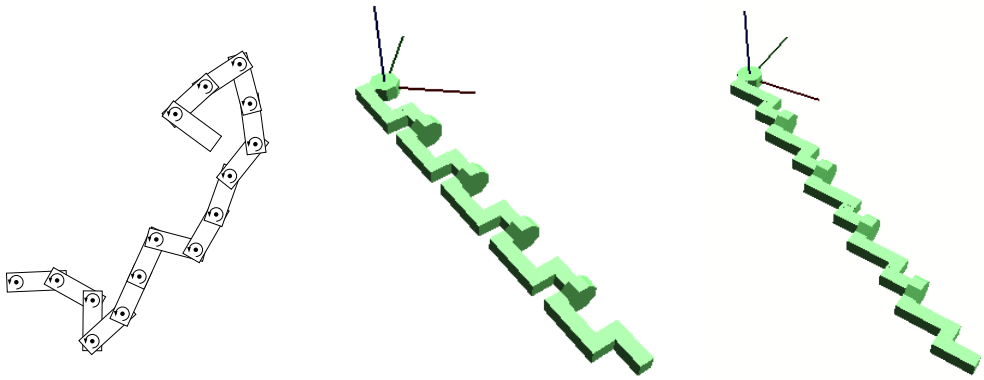
In this appendix we show the probability distribution of the manipulability for several of our robots. For all of the robots, we define the manipulability measure to evaluate the ability of the robot to change the position of the end-effector. For planar robots, this corresponds to the x and y position of the end-effector. For the 3D workspace robots, this corresponds to the x, y, and z position of the end-effector.

Shown in Figure B.1 are three robots for which we have computed the manipulability distribution functions. The first example is a set of 19 planar robots, with the number of joints ranging from two to 20. Each link of these robots is modeled as a rectanguloid with a width of 2.1 units and a height of 1 unit. The total length of these robots is fixed at 70 units, with each link length being scaled appropriately. The first joint of these robots is given full revolute range of motion, and limits are placed on the other joints.

The second and third robot examples shown in Figure B.1 are sets of five 3D workspace robots, with 3, 6, 9, 12, and 15 joints. As with the planar robots, the total length of these robots is fixed at 42 units. The cross section of the links is 1.5 units by 1.5 units. The purpose for showing the manipulability results for both of these is to examine the effect of the different offset between the second and third link of each triple of links. This offset also has an effect on ϵ-robustness, as was shown in Chapter 6.

For all of the robots, we create two distributions, which we label "filtered" and "not filtered" on the figures. The difference between these two distributions is that for the "filtered" distribution, we discard samples for which the robot collides with itself; for the "not filtered" distribution, we do not discard these samples. For the "not filtered" distribution and for all robots, we compute the probability distributions based on evaluating the manipulability of 10^7 configurations chosen uniformly at random from the configuration space of the robot.

For the "filtered" distributions, we evaluate the manipulability at the following number of self-collision-free samples of the configuration space. For the 3D workspace robots and for planar robots with up to 13 joints, we evaluated the manipulability at 10^7 samples. For planar robots with 14, 15, and 16 joints, we used 10^6 samples. For planar robots with 17 to 20 joints, we evaluated 50 000 configura-



(a) (b) (c)

Figure B.1 Three example robots with 15 joints: (a) planar robot, (b) 3D workspace robot class A, (c) 3D workspace robot class B.

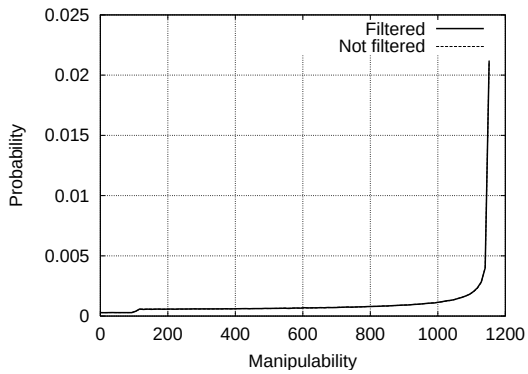


Figure B.2 Special case for planar robots: two joints.

tions. The reason for using fewer samples for the planar robots with more than 13 joints is the difficulty of generating self-collision free samples of the configurations space for these robots (see Figure 6.3(a) for the number of samples needed to find one that is self-collision-free).

We used the following bucket sizes for computing the distributions. For the 3D workspace robots, we use a bucket size of five. For the planar robots with fewer than 17 joints, we use a bucket size of one. For the planar robots with more than 16 joints, we use a bucket size of 25. For all the plots we show based on these distributions, we use the "csplines" function of gnuplot to smooth appearance of the curves.

We begin with the results for the planar robots. Shown in Figure B.2 is a special case for the planar robots. This is the only case for which we have observed the manipulability distribution having a peak at the maximum manipulability. A reason for this can be found in the formula for the manipulability of this robot: $l_1 l_2 |\sin\theta_2|$. The peaks of this function are fairly broad, whereas the valleys are narrow, with the result that the manipulability of the uniform random samples tends to cluster near the peak.

The distribution functions for the remaining planar robots are shown in Figures B.3, B.4, and B.5. A trend that can be observed in these figures is that the peaks move toward low values of manipulability as the number of joints increases, particularly for the "not filtered" case. The result of filtering the samples for self-collision is to shift the peak to higher values of manipulability and to broaden the shape of the curve. The shape of these curves, particularly for robots with six or more joints, tends to follow the gamma distribution, with the filtered case having an appearance that is more Gaussian.

Shown in Figures B.6 and B.7 are the manipulability distributions of two different robots with 3D workspaces. As with the planar robots, these robots also have a special case, though for the three-joint robots instead of the two-joint robots. For the three-joint robots, there is a peak in the distribution at small values of manipulability, and from there the curve trails off to a relatively constant level up to the maximum manipulability. There is also no difference between the filtered and unfiltered curves because, essentially, there are no configurations in which the robot collides with itself.

For robots with more than three-joints, two differences become apparent between the class A robots and the class B robots. First, there are fewer configurations in which the robots of class B are in self-collision; we saw this difference when sampling the configuration spaces of these robots when constructing roadmaps (see Figure 6.3(c) and (d)). The effect of this is that the distribution of the manipulability shifts less when the self-collisions are filtered out. The second difference between the class A and class B robots is that the manipulability value for the class B robots reaches a higher peak. This difference is due to the different offsets between the second and third link of every triple of links.

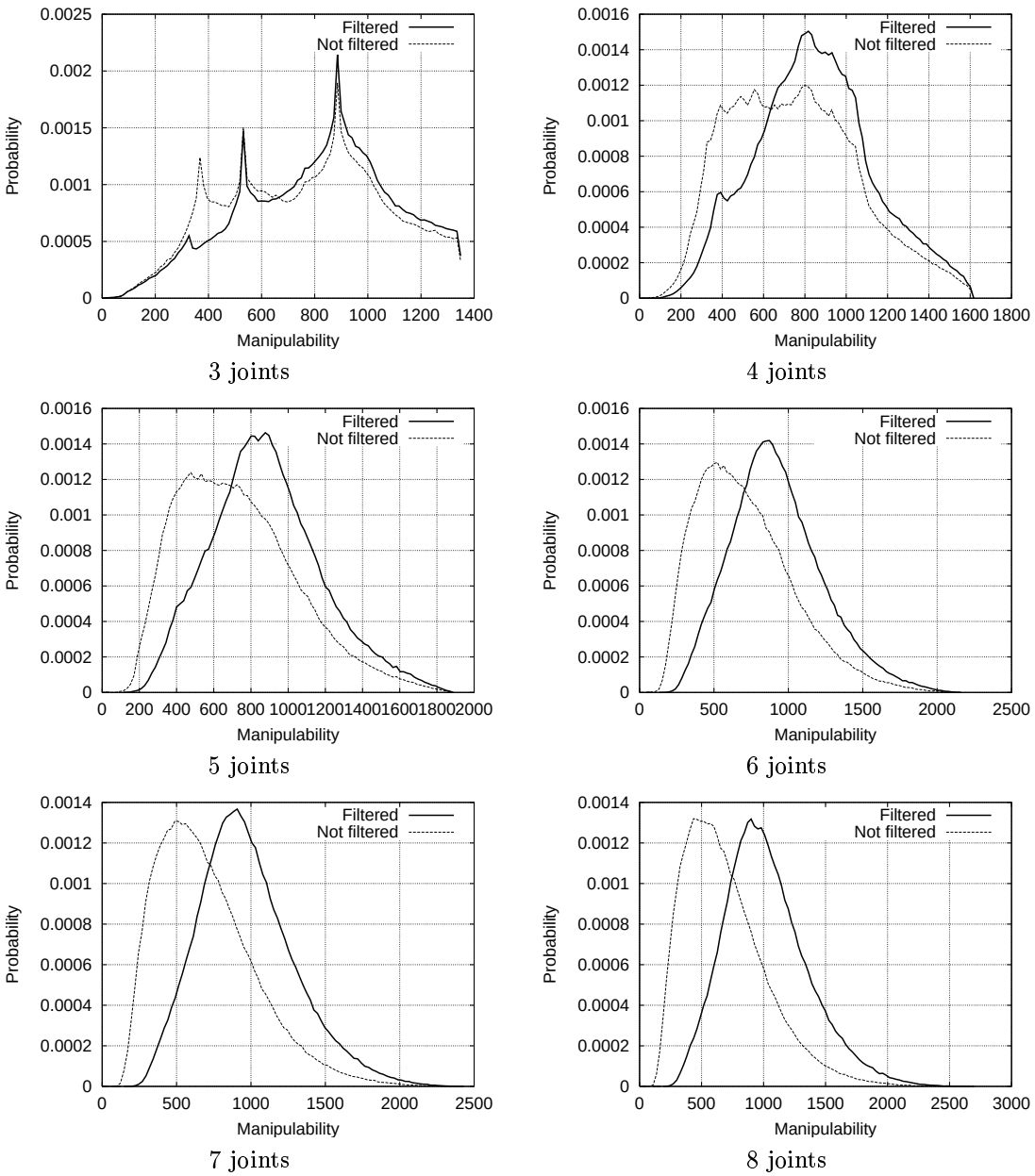


Figure B.3 Planar robot, 3–8 joints.

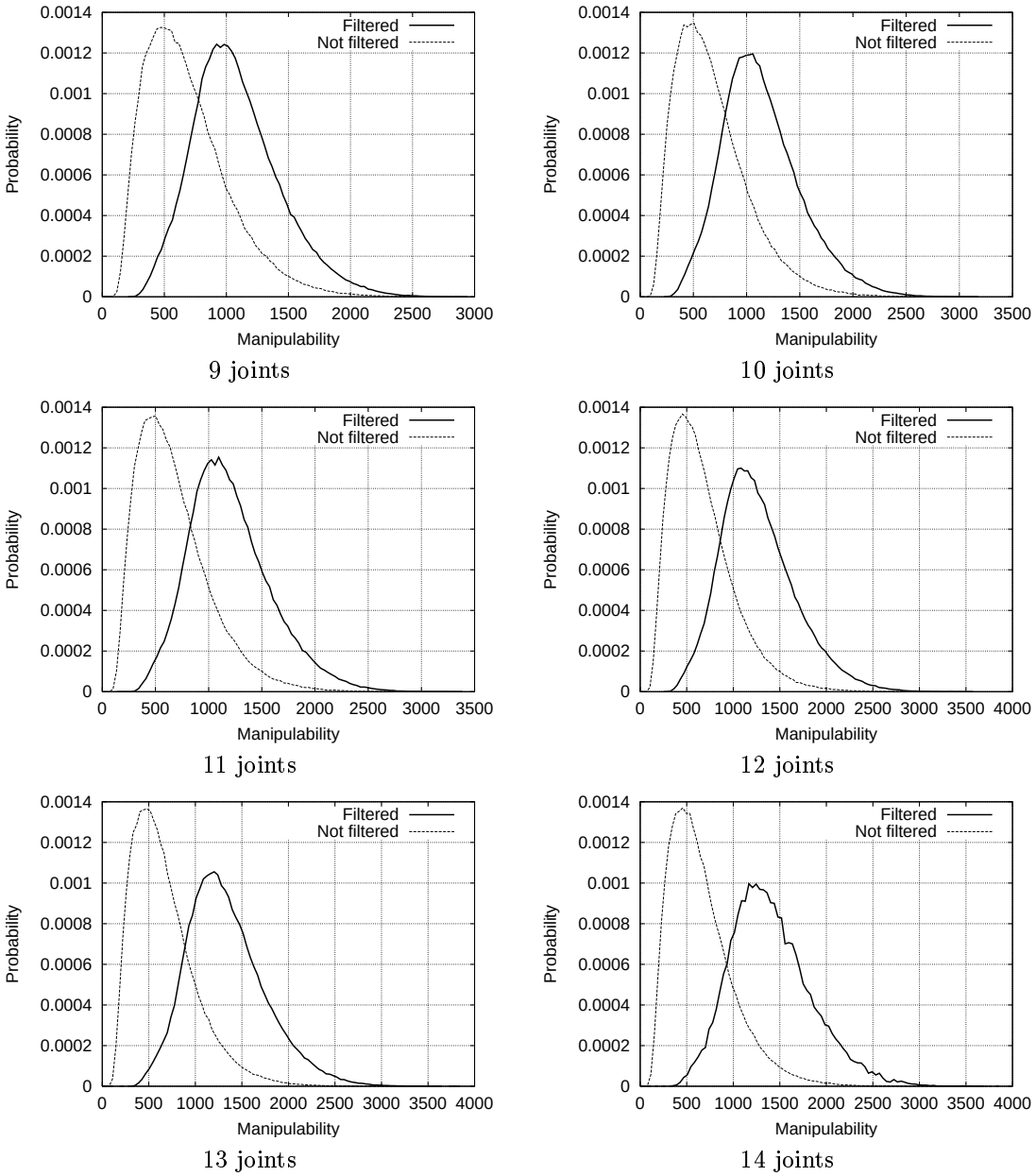


Figure B.4 Planar robot, 9–14 joints.

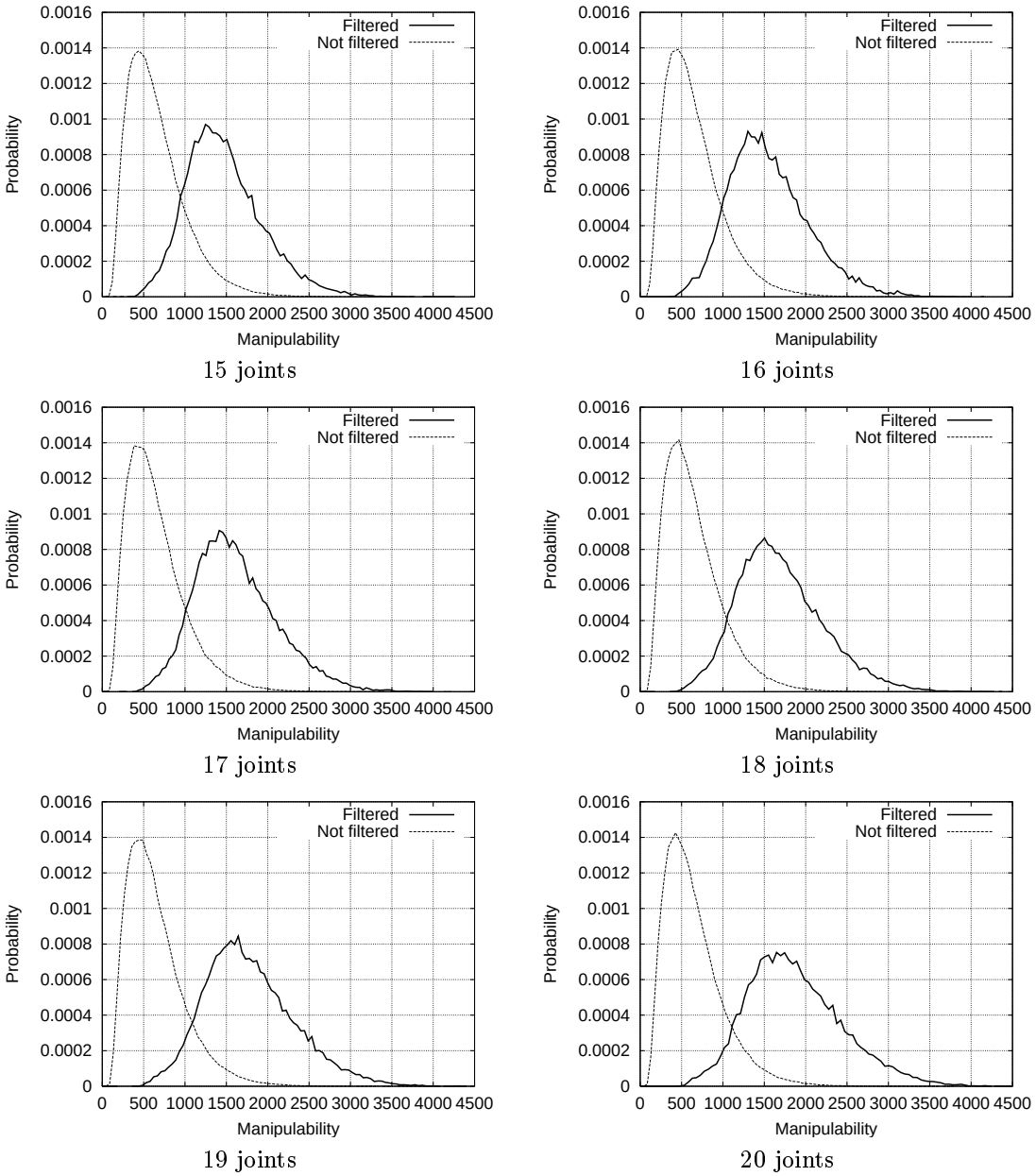


Figure B.5 Planar robot, 15–20 joints.

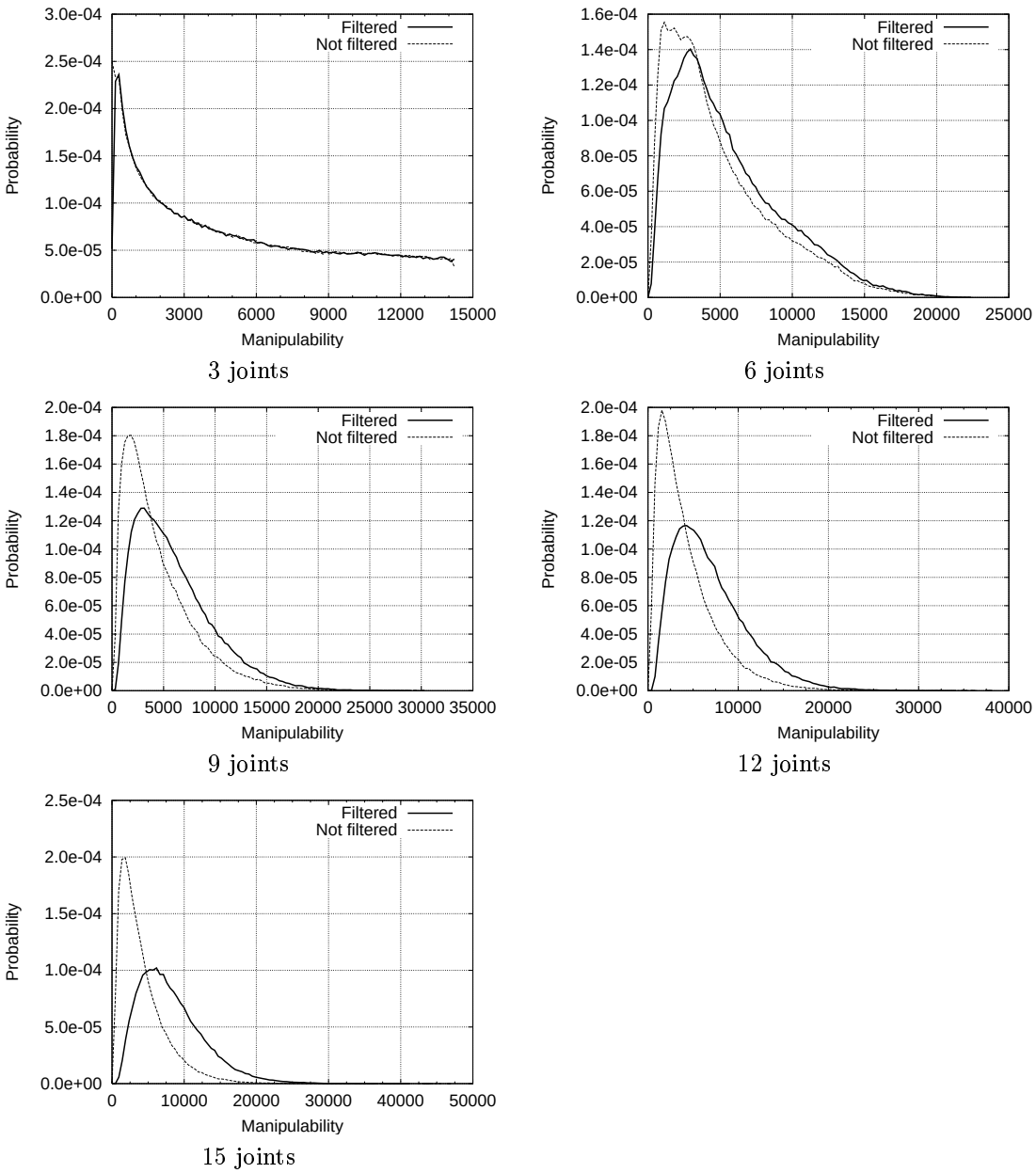


Figure B.6 Robot class A with 3D workspace.

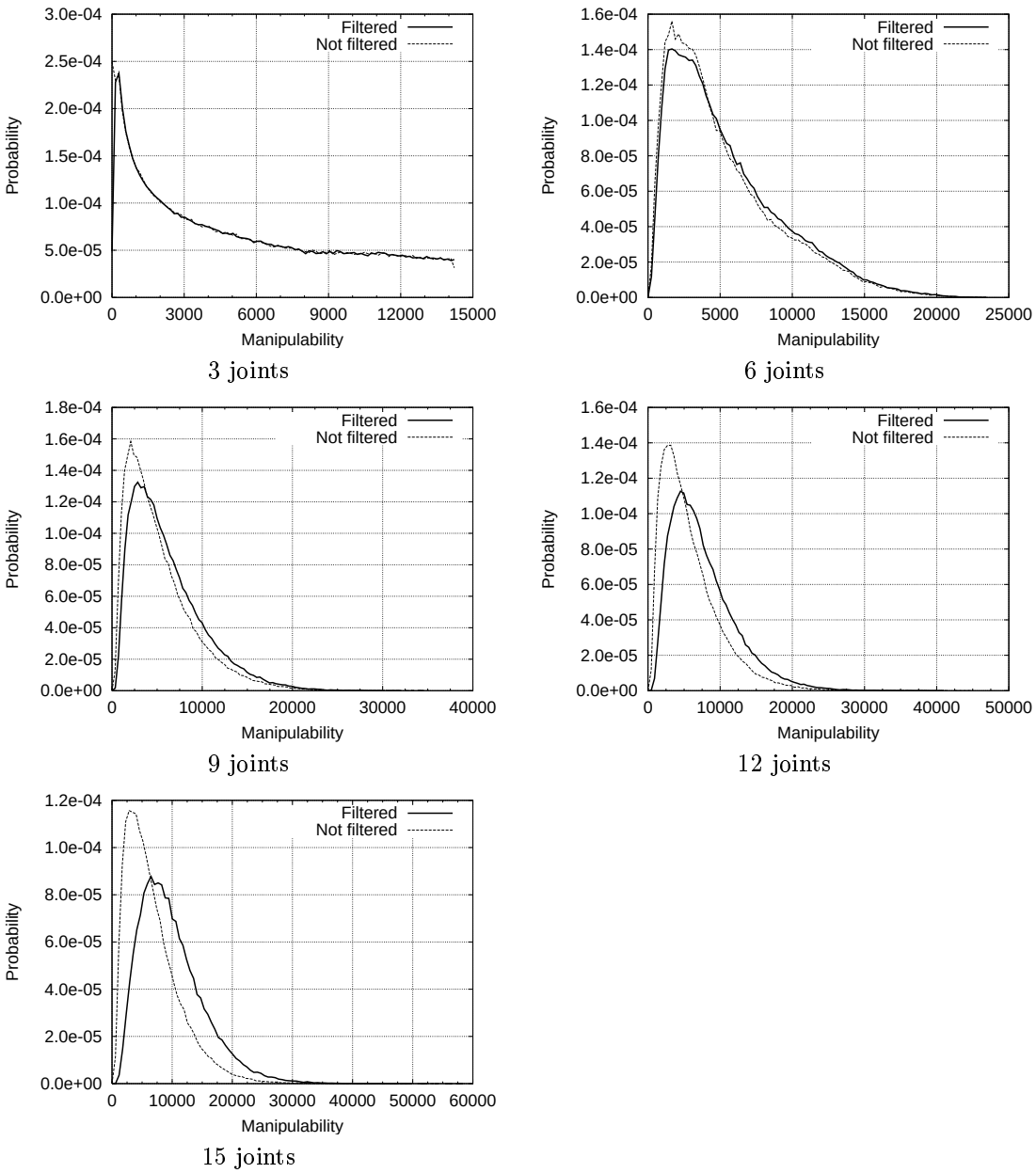


Figure B.7 Robot class B with 3D workspace.

REFERENCES

[1] S. Abrams and P. K. Allen. Swept volumes and their use in viewpoint computation in robot work-cells. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 188–193, 1995.

[2] S. Abrams, P. K. Allen, and K. A. Tarabanis. Dynamic sensor planning. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 605–610, 1993.

[3] J. M. Ahuactzin and K. Gupta. The kinematic roadmap: A motion planning based global approach for inverse kinematics of redundant robots. *IEEE Transactions on Robotics and Automation*, 15(4):653–669, Aug. 1999.

[4] J. M. Ahuactzin, K. Gupta, and E. Mazer. Manipulation planning for redundant robots: A practical approach. *International Journal of Robotics Research*, 17(7):731–747, July 1998.

[5] J.-M. Ahuactzin, E. Mazer, and P. Bessière. Fondements mathématiques d'algorithme "fil d'Ariane". *Revue d'Intelligence Artificielle*, 9(1):7–34, 1995.

[6] J.-M. Ahuactzin, E.-G. Talbi, P. Bessière, and E. Mazer. Using genetic algorithms for robot motion planning. In *European Conference on Artificial Intelligence*, pages 671–5, 1992.

[7] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Proceedings of Workshop on Algorithmic Foundations of Robotics*, pages 155–168, 1998.

[8] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. *IEEE Transactions on Robotics and Automation*, 16(4):442–447, Aug. 2000.

[9] N. M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *Proceedings of IEEE Conference on Robotics and Automation*, volume 1, pages 113–120, 1996.

[10] E. Anshelevich, S. Owens, F. Lamiraux, and L. E. Kavraki. Deformable volumes in path planning applications. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 2290–2295, 2000.

[11] S. Arya and D. M. Mount. Algorithms for fast vector quantization. In *Proceedings of Data Compression Conference*, pages 381–390, 1993.

[12] S. Arya and D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 271–280, 1993.

[13] S. Arya, D. M. Mount, and O. Narayan. Accounting for boundary effects in nearest neighbor searching. *Discrete and Computational Geometry*, 16(2):155–176, Sept. 1996.

[14] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM*, 45(6):891–923, Nov. 1998.

[15] P. J. Ausbeck, Jr. The piecewise-constant image model. *Proceedings of the IEEE*, 88(11):1779–1789, Nov. 2000.

[16] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4), Dec. 1996.

[17] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *International Journal of Robotics Research*, 10(6):628–649, Dec. 1991.

[18] O. B. Bayazit, G. Song, and N. M. Amato. Providing haptic 'hints' to automatic motion planners. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 529–536, 2000.

[19] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text Compression.* Prentice Hall, Englewood Cliffs, NJ, 1990.

[20] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, Sept. 1975.

[21] J. L. Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, Apr. 1980.

[22] J. L. Bentley. k-d trees for semidynamic point sets. In *Proceedings of the ACM Symposium on Computational Geometry*, pages 187–197, 1990.

[23] J. L. Bentley, B. W. Weide, and A. C. Yao. Optimal expected-time algorithms for closest point problems. *ACM Transations on Mathematical Software*, 6(4):563–580, Dec. 1980.

[24] S. Berchtold, D. A. Keim, C. Böhm, and H.-P. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *Proceedings of ACM Symposium on Priciples of Database Systems*, pages 78–86, 1997.

[25] M. Bern. Approximate closest-point queries in high dimensions. *Information Processing Letters*, 45(2):95–99, 26 Feb. 1993.

[26] P. Bessiére, J.-M. Ahuactzin, E.-G. Talbi, and E. Mazer. The "Ariadne's clew" algorithm: Global planning with local methods. In *Proceedings of Workshop on Algorithmic Foundations of Robotics*, pages 39–47, 1994.

[27] D. Blackmore and M. C. Leu. A differential equation approach to swept volumes. In *Proceedings of Rensselaer's Second International Conference on Computer Integrated Manufacturing*, pages 143–149, 1990.

[28] R. Bohlin and L. E. Kavraki. Path planning using lazy PRM. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 521–528, 2000.

[29] V. Boor, M. H. Overmars, and A. F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 1018–1023, 1999.

[30] S. Boussac and A. Crosnier. Swept volumes generated from deformable objects application to nc verification. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 1813–1818, 1996.

[31] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *Proceedings of ACM Conference on Management of Data*, pages 357–368, 1997.

[32] T. Bozkaya and M. Ozsoyoglu. Indexing large metric spaces for similarity search queries. *ACM Transactions on Database Systems*, 24(3):361–404, Sept. 1999.

[33] S. Brin. Near neighbor search in large metric spaces. In *Proceedings of IEEE VLDB Conference*, pages 574–584, 1995.

[34] A. J. Broder. Strategies for efficient incremental nearest neighbor search. *Pattern Recognition*, 23(1/2):171–178, 1990.

[35] R. Brooks and T. Lozano-Pérez. A subdivision algorithm in configuration space for findpath with rotation. In *International Joint Conference on Artificial Intelligence*, pages 799–806, 1983.

[36] J. F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.

[37] B. Carpentieri, M. J. Weinberger, and G. Seroussi. Lossless compression of continuous-tone images. *Proceedings of the IEEE*, 88(11):1797–1809, Nov. 2000.

[38] T. M. Chan. Approximate nearest neighbor queries revisited. In *Proceedings of the ACM Symposium on Computational Geometry*, pages 352–358, 1997.

[39] P. C. Chen and Y. K. Hwang. SANDROS: A motion planner with performance proportional to task difficulty. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 2346–2353, 1992.

[40] K. L. Clarkson. Fast algorithms for the all nearest neighbors problem. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 226–232, 1983.

[41] K. L. Clarkson. An algorithm for approximate closest-point queries. In *Proceedings of the ACM Symposium on Computational Geometry*, pages 160–164, 1994.

[42] J. G. Cleary. Analysis of an algorithm for finding nearest neighbors in euclidean space. *ACM Transations on Mathematical Software*, 5(2):183–192, June 1979.

[43] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, second edition, 2000.

[44] D. Dolev, Y. Harari, N. Linial, N. Nisan, and M. Parnas. Neighborhood preserving hashing and approximate queries. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 251–259, 1994.

[45] C. Eastman and S. Weiss. Tree structures for high dimensionality nearest neighbor searching. *Information Systems*, 7(2):115–122, 1982.

[46] M. Effros. PPM performance with BWT complexity: A fast and effective data compression algorithm. *Proceedings of the IEEE*, 88(11):1703–1712, Nov. 2000.

[47] P. Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, IT-21(2), Mar. 1975.

[48] C. D. Feustel and L. G. Shapiro. The nearest neighbor problem in an abstract metric space. *Pattern Recognition Letters*, 1(2):125–128, Dec. 1982.

[49] R. A. Finkel and J. L. Bentley. Quad trees — a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974.

[50] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transations on Mathematical Software*, 3(3):209–226, Sept. 1977.

[51] K. Fukunaga and P. M. Narendra. A branch and bound algorithm for computing k-nearest neighbors. *IEEE Transactions on Computers*, C-24(7):750–753, July 1975.

[52] B. Furht, J. Greenberg, and R. Westwater. *Motion Estimation Algorithms for Video Compression*. Kluwer Academic Publishers, Boston, 1997.

[53] S. W. Golumb. Run-length encodings. *IEEE Transactions on Information Theory*, IT-12(3):399–401, July 1966.

[54] D. Greene, M. Parnas, and F. Yao. Multi-index hashing for information retrieval. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 722–731, 1994.

[55] L. J. Guibas, C. Holleman, and L. E. Kavraki. A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach. In *Proceedings of IEEE/RSJ Conference on Intelligent Robots and Systems*, 1999.

[56] L. Han and N. M. Amato. A kinematics-based probabilistic roadmap method for closed chain systems. In *Proceedings of Workshop on Algorithmic Foundations of Robotics*, 2000.

[57] D. Hankerson, G. A. Harris, and P. D. Johnson, Jr. *Introduction to Information Theory and Data Compression*. Discrete Mathematics and its Applications. CRC Press, New York, 1998.

[58] C. Holleman and L. E. Kavraki. A framework for using the workspace medial axis in PRM planners. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 1408–1413, 2000.

[59] C. Holleman, L. E. Kavraki, and J. Warren. Planning paths for a flexible surface patch. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 21–26, 1998.

[60] T. Horsch, F. Schwarz, and H. Tolle. Motion planning with many degrees of freedom — random reflections at c-space obstacles. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 3318–3323, 1994.

[61] D. Hsu, L. E. Kavraki, J.-C. Latombe, and R. Motwani. Capturing the connectivity of high-dimensional geometric spaces by parallelizable random sampling techniques. In P. M. Pardalos and S. Rajasekaran, editors, *Advances in Randomized Parallel Computing*, pages 159–182. Kluwer Academic Publishers, 1999.

[62] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In *Proceedings of Workshop on Algorithmic Foundations of Robotics*, pages 141–154, 1998.

[63] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *International Journal of Computational Geometry and Applications*, 9(4 & 5):495–512, 1999.

[64] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of IRE*, 40(9):1098–1101, Sept. 1952.

[65] G. M. Hunter and K. Steiglitz. Operations on images using quad trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):145–153, Apr. 1979.

[66] Y. K. Hwang and N. Ahuja. Path planning using a potential field representation. Technical Report UILU-ENG-8-2251, University of Illinois, Oct. 1988.

[67] P. Indyk. On approximate nearest neighbors in non-euclidean spaces. In *Proc. 39th Symposium on Foundations of Computer Science*, pages 148–155, 1998.

[68] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 604–613, 1998.

[69] S. Kambhampati and L. S. Davis. Multiresolution path planning for mobile robots. *IEEE Journal of Robotics and Automation*, 2(3):135–145, Sept. 1986.

[70] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest-neighbor queries. In *Proceedings of ACM Conference on Management of Data*, pages 369–380, 1997.

[71] A. Kaufman and E. Shimony. 3D scan-conversion algorithms for voxel-based graphics. In *Proceedings of 1986 Workshop on Interactive 3D Graphics*, pages 45–76, New York, 1986. ACM.

[72] L. E. Kavraki, M. N. Kolountzakis, and J.-C. Latombe. Analysis of probabilistic roadmaps for path planning. In *Proceedings of IEEE Conference on Robotics and Automation*, volume 4, pages 3020–3025, 1996.

[73] L. E. Kavraki, F. Lamiraux, and C. Holleman. Towards planning for elastic objects. In *Proceedings of Workshop on Algorithmic Foundations of Robotics*, 1998.

[74] L. E. Kavraki and J.-C. Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proceedings of IEEE Conference on Robotics and Automation*, volume 3, pages 2138–2145, 1994.

[75] L. E. Kavraki and J.-C. Latombe. Probabilistic roadmaps for robot path planning. In K. Gupta and P. del Pobil, editors, *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, pages 33–53. John Wiley & Sons LTD, 1998.

[76] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, Aug. 1996.

[77] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986.

[78] B. S. Kim and S. B. Park. A fast nearest neighbor finding algorithm based on the ordered partition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):761–766, Nov. 1986.

[79] R. Kindel, D. Hsu, J.-C. Latombe, and S. Rock. Kinodynamic motion planning amidst moving obstacles. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 537–543, 2000.

[80] J. M. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 599–608, 1997.

[81] D. Koditschek. Robot planning and control via potential functions. In *The Robotics Review 1*, pages 349–367. MIT Press, 1989.

[82] J. J. Kuffner, Jr. and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 995–1001, 2000.

[83] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 614–623, 1998.

[84] N. J. Larsson and A. Moffat. Off-line dictionary-based compression. *Proceedings of the IEEE*, 88(11):1722–1732, Nov. 2000.

[85] S. M. LaValle and J. J. Kuffner, Jr. Randomized kinodynamic planning. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 473–479, 1999.

[86] S. M. LaValle and J. J. Kuffner, Jr. Rapidly-exploring random trees: Progress and prospects. In *Proceedings of Workshop on Algorithmic Foundations of Robotics*, 2000.

[87] S. M. LaValle, J. H. Yakey, and L. E. Kavraki. A probabilistic roadmap approach for systems with closed kinematic chains. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 1671–1676, 1999.

[88] P. Leven and S. Hutchinson. Toward real-time path planning in changing environments. In *Proceedings of Workshop on Algorithmic Foundations of Robotics*, 2000.

[89] P. Leven, S. Hutchinson, D. Burschka, and G. Färber. Perception-based motion planning for indoor exploration. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 695–701, 1999.

[90] K.-I. Lin, H. V. Jagadish, and C. Faloutsos. The TV-Tree: An index structure for high-dimensional data. *VLDB Journal*, 3(4):517–542, Oct. 1994.

[91] M. Lin and D. Manocha. Efficient contact determination in dynamic environments. *International Journal of Computational Geometry and Applications*, 7(1):123–151, 1997.

[92] M. Lizhuang, J. Zhongding, and C. K. Y. Tony. Interpolating and approximating moving frames using b-splines. In *Proceedings of the Eight Pacific Conference on Computer Graphics and Applications*, pages 154–164, 2000.

[93] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, Feb. 1983.

[94] R. W. Lucky. *Silicon Dreams: Information, Man, and Machine.* St. Martin's Press, New York, 1989.

[95] D. W. Matula. A linear time $2 + \epsilon$ approximation algorithm for edge connectivity. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 500–504, 1993.

[96] E. Mazer, J. M. Ahuactzin, and P. Bessière. The Ariadne's clew algorithm. *Journal of Artificial Intelligence Research*, 9:295–316, 1998.

[97] A. McLean and I. Mazon. Incremental roadmaps and global path planning in evolving industrial environments. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 101–107, 1996.

[98] B. Meyer and P. Tischer. TMW— a new method for lossless image compression. In *Proceedings of International Picture Coding Symposium*, 1997.

[99] B. Mirtich. V-Clip: Fast and robust polyhedral collision detection. Technical Report TR97-05, Mitsubishi Electric Research Laboratory, 201 Broadway, Cambridge, MA 02139, June 1997.

[100] J. L. Mitchell, W. B. Pennebaker, C. E. Fogg, and D. J. LeGall, editors. *MPEG Video Compression Standard.* Chapman and Hall, New York, 1997.

[101] G. Motta, J. A. Storer, and B. Carpentieri. Lossless image coding via adaptive linear prediction and classification. *Proceedings of the IEEE*, 88(11):1790–1796, Nov. 2000.

[102] N. Nilsson. A mobile automaton: An application of artificial intelligence techniques. In *Proceedings of the First International Joint Conference on Artifical Intelligence*, pages 509–520, Washington, D.C., 1969.

[103] M. H. Overmars and P. Švestka. A probabilistic learning approach to motion planning. In *Proceedings of Workshop on Algorithmic Foundations of Robotics*, pages 19–37, 1994.

[104] M. H. Overmars and P. Švestka. A paradigm for probabilistic path planning. Technical Report UU-CS-1995-22, Utrecht University, Mar. 1995.

[105] B.-U. Pagel, F. Korn, and C. Faloutsos. Deflating the dimensionality curse using multiple fractal dimensions. In *Proceedings of IEEE International Conference on Data Engineering*, pages 589–598, 2000.

[106] V. Ramasubramanian and K. K. Paliwal. An efficient approximation-elimination algorithm for fast nearest-neighbor search based on a spherical distance coordinate formulation. *Pattern Recognition Letters*, 13(7):471–480, July 1992.

[107] R. L. Rivest. On the optimality of Elias's algorithm for performing best-match searches. In *Proceedings of the IFIP Conference on Information Processing*, pages 678–681, 1974.

[108] D. F. Rogers. *Procedural Elements for Computer Graphics.* McGraw-Hill, Boston, MA, second edition, 1998.

[109] A. Rosenfeld and A. C. Kak. *Digital Picture Processing.* Academic Press, New York, second edition, 1982.

[110] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of ACM Conference on Management of Data*, pages 71–79, 1995.

[111] H. Samet. The quadtree and related hierarchical data structures. *Computing Surveys*, 16(2):187–260, June 1984.

[112] J. T. Schwartz, M. Sharir, and J. Hopcroft, editors. *Planning, Geometry, and Complexity of Robot Motion.* Ablex, Norwood, NJ, 1987.

[113] R. F. Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6(4):579–589, 1991.

[114] P. Švestka and M. H. Overmars. Coordinated motion planning for multiple car-like robots using probabilistic roadmaps. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 1631–1636, 1995.

[115] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2), June 1972.

[116] J. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, 25 Nov. 1991.

[117] P. M. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete and Computational Geometry*, 4(2):101–115, 1989.

[118] D. Vallejo, C. Jones, and N. M. Amato. An adaptive framework for 'single shot' motion planning. Technical Report TR99-024, Department of Computer Science, Texas A&M University, College Station, TX, Oct. 1999.

[119] G. van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4):1–14, 1997.

[120] G. van den Bergen. A fast and robust GJK implementation for collision detection of convex objects. *Journal of Graphics Tools*, 4(2):7–25, 1999.

[121] D. A. White and R. Jain. Similarity indexing with the SS-tree. In *Proceedings of IEEE International Conference on Data Engineering*, pages 516–523, 1996.

[122] S. A. Wilmarth, N. M. Amato, and P. F. Stiller. Motion planning for a rigid body using random networks on the medial axis of the free space. In *Proceedings of ACM Symposium on Computational Geometry*, pages 173–180, 1999.

[123] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, Inc., San Francisco, California, second edition, 1999.

[124] P. G. Xavier. Fast swept-volume distance for robust collision detection. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 1162–1169, 1997.

[125] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321, 1993.

[126] P. N. Yianilos. Locally lifting the curse of dimensionality for nearest neighbor search. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 361–370, 2000.

[127] T. Yoshikawa. Manipulability of robotic mechanisms. *International Journal of Robotics Research*, 4(2):3–9, Apr. 1985.

[128] Y. Yu and K. Gupta. Sensor-based roadmaps for motion planning for articulated robots in unknown environments: Some experiments with an eye-in-hand system. In *Proceedings of IEEE/RSJ Conference on Intelligent Robots and Systems*, 1999.

[129] G. Yuval. Finding near neighbors in k-dimensional space. *Information Processing Letters*, 3(4):113–114, Mar. 1975.

VITA

Peter J. Leven received B.S. degrees *summa cum laude* in electrical engineering and in computer science in 1993 from Michigan Technological University and an M.S. degree in electrical engineering in 1997 from the University of Illinois at Urbana-Champaign.

He is a member of Tau Beta Pi, Phi Kappa Phi, and the Institute of Electrical and Electronics Engineers. His published articles include "Perception-based motion planning for indoor environments," by Peter Leven, Seth Hutchinson, Darius Burschka and Georg Färber in *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*; "Toward real-time path planning in changing environments," by Peter Leven and Seth Hutchinson in *Proceedings of the 2000 Workshop on the Algorithmic Foundations of Robotics*; and "Real-time path planning in changing environments: Some preliminary results," by Peter Leven and Seth Hutchinson in *Proceedings of the 2000 International Symposium on Robotics*.

While at the University of Illinois at Urbana-Champaign, he taught the Digital Systems Laboratory (ECE 249) for three semesters as a teaching assistant. He also taught the Introduction to Electrical and Computer Engineering (ECE 110), first for two semesters as a laboratory teaching assistant, and then for ten semesters as head laboratory teaching assistant.

From September 1997 to May 1998, he was a visiting scholar at the Technische Universität München.