SAMPLING AND SEARCHING METHODS FOR PRACTICAL MOTION PLANNING ALGORITHMS

BY
GANNA YERShOVA
Diploma, Kharkiv National University, 1999
M.S., Iowa State University, 2003

DISSERTATION
Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2008

Urbana, Illinois

Doctoral Committee:
Professor Steven M. LaValle, Chair
Professor Seth A. Hutchinson
Associate Professor Jeff G. Erickson
Assistant Professor Stephen Bond
Associate Professor James Kuffner, Carnegie Mellon University
Abstract

In its original formulation, the motion planning problem considers the search of a robot path from an initial to a goal configuration. The study of motion planning has advanced significantly in recent years, in large part due to the development of highly successful sampling and searching techniques. Recent advances have influenced sampling-based motion planning algorithms to be used in disparate areas such as humanoid robotics, automotive manufacturing, architecture, computational geography, computer graphics, and computational biology. Many of these methods work well on a large set of problems, however, they have weaknesses and limitations. This thesis expands the basic motion planning techniques to include critical concerns that are not covered by the motion planning algorithms that are in widespread use now. The technical approach is organized around three main thrusts: 1) the development of efficient nearest neighbor searching techniques for spaces arising in motion planning; 2) the development of uniform sampling techniques on these spaces to allow resolution completeness in sampling-based planning algorithms; and 3) the development of guided sampling techniques for efficient exploration on such spaces. We show that most of the modern motion planners incorporate one or more of these components; therefore, addressing these core issues in motion planning does not only lead to a more fundamental understanding of the problem, but also to more efficient practical algorithms. Our results include algorithms addressing the issues, theoretical analysis of their performance and experimental evaluation on motion planning problems.
To my brother, Mitya, my mom, and my dad
Acknowledgments

First of all I would like to acknowledge and thank my adviser, Steve LaValle, for making the PhD program at UIUC possible for me and helping me throughout the long, difficult, but for most part, enjoyable process. I am grateful to my committee members: Stephen Bond, Jeff G. Erickson, Seth A. Hutchinson, and James Kuffner for giving me constructive feedback and going through the revisions of my work. This work has benefited greatly from the discussions and collaborations with other researchers from various universities: Devin Balkcom, Bruce Donald, Rob Ghrist, Sariel Har-Peled, Jean-Paul Laumond, Julie Mitchell, Mark Overmars, and Nicola Simeon.

I shared my work environment with great lab mates: Leonardo Bobadilla, Peng Cheng, Hamidreza Chitzas, Lars Erickson, Steve Lindemann, Jason O’Kane, Kamilah Taylor, Benjamin Tovar, and Jingjin Yu. It was intellectually stimulating and enjoyable to work together with these smart people.

Outside of my work, I was lucky to have many friends in Illinois and elsewhere: Doina Caragea, Jason Fechter, Darien Hall, Barbara Illie, Dasha Kishenevskaya, Mirko Montanari, Suvda Myagmar, and Mbikayi Nsumuna. My closest friends, Cigdem Sengul, Natalia Stakhanova, Kamilah Taylor, Suvda Myagmar, and Noemi Waight, know and love me. The good times we spent together, their advice, and support are the most memorable and valuable things I earned in Urbana.

Finally, the last months of writing my thesis were especially enduring; these friends shared the long weeks with me, which helped me more than they can imagine: Kamilah Taylor, Noemi Waight, Erin Chambers, and Johnny Wood.

I want to say a special thank you to my brother, Mitya, who was my roommate for the last two years at UIUC. His optimism, energy, and sincere care gave me the necessary confidence to accomplish what I did. My deepest thanks go to Mitya, my mom, and my dad for their endless support and unconditional love.
# Table of Contents

List of Figures .......................................................... vii

Chapter 1  Introduction ......................................................... 1
  1.1 Motivation and Background .............................................. 2
  1.1.1 Collision Detection ............................................... 4
  1.1.2 Sampling Considerations for Motion Planning ....................... 5
  1.1.3 Parameter Tuning in Motion Planning ............................... 6
  1.1.4 Nearest Neighbor Searching ....................................... 7
  1.1.5 Motion Primitives ................................................ 8
  1.1.6 Extensions to Closed Chains and Constrained Geometries .......... 9
  1.2 Summary of Contributions and Thesis Organization .................. 10

Chapter 2  Problem Formulation .............................................. 12
  2.1 Basic Motion Planning Problem ....................................... 12
  2.1.1 Extension to Constrained Geometries and Closed Kinematic Chains 14
  2.1.2 Special Case: Linkages with Closed Chains ......................... 15
  2.2 Metrics, and Measure on Configuration Spaces ....................... 16
  2.2.1 Metric Space .................................................. 16
  2.2.2 Measure Space ................................................ 17
  2.2.3 Common Spaces Arising in Motion Planning ....................... 18
  2.3 Incremental Sampling and Searching (ISS) Framework for Motion Planning 27
  2.3.1 RRT ....................................................... 28
  2.3.2 PRM ....................................................... 28
  2.4 Efficient Implementation of the ISS Framework ...................... 29
  2.4.1 Efficient Nearest Neighbor Searching ............................ 30
  2.4.2 Uniform Deterministic Sampling Methods .......................... 30
  2.4.3 Guided Sampling for Efficient Exploration ....................... 31

Chapter 3  Efficient Nearest Neighbor Searching for Motion Planning ...... 32
  3.1 Introduction ...................................................... 32
  3.2 Problem Formulation ................................................ 34
  3.3 Nearest-Neighbor Searching Overview ................................ 34
  3.4 An Approach Based on Kd-trees ..................................... 35
  3.4.1 A Naive Way to Use Kd-Trees .................................. 35
  3.4.2 Representing the Spaces of Interest ............................. 35
  3.4.3 Kd-trees for the Spaces of Interest ............................. 37
  3.4.4 Construction Phase ......................................... 38
  3.4.5 Query Phase ............................................... 38
  3.4.6 Making Kd-trees Dynamic ..................................... 39
  3.4.7 Analysis .................................................. 40
  3.5 Experimental Results .............................................. 42
List of Figures

1.1 The Piano Mover’s Problem. The goal is to find a path for a piano to be moved out of a house without hitting other furniture. The input is computer-aided design (CAD) models for both the piano and the house. ................................................. 2

1.2 A benchmark from the European automotive manufacturing industry. The motion planning problem here is to find a collision free path dismounting a wiper motor from a car body. (Courtesy of Kineo [35]) ................................................................. 5

1.3 Molecule example. The task is to compute the pathway of a small molecule to the active site located inside the protein model. We thank LAAS-SNRC [110] for providing protein models for our work. ................................................................. 6

1.4 An example of a motion planning problem involving searching a maze. The solution lies in a narrow passage in the configuration space, which usually leads to tens of thousands of configurations being generated by a motion planning algorithm. More complicated mazes may lead to generation of millions, or even billions of nodes. ............................................. 7

1.5 (a) Manipulation problem for a closed chain. (b) Manipulation problem for sliding a washer against a rod. The two seemingly different problems involve searching of the similar thin feasible solution spaces described by kinematic constraints in (a) and geometry of the problem in (b). ................................................................. 10

2.1 Visualization of the hyperspherical coordinates on $SO(3)$. (a) The full range of the hyperspherical coordinate $\psi \in [0, \pi/2]$ is shown while the coordinates $(\theta, \phi)$ form a discretization of size 20 over $S^2$. (b) The half-spheres show the ranges of the hyperspherical coordinates $(\theta, \phi)$, while $\psi$ takes four discrete values over $[0, \pi/2]$ .............. 23

2.2 Visualization of the Hopf coordinates on $SO(3)$. (a) The full range of the Hopf coordinate $\psi \in [0, 2\pi]$ is shown while the coordinates $(\theta, \phi)$ form a discretization of size 12 over $S^2$. (b) The spheres show the ranges of the Hopf coordinates $(\theta, \phi)$, while $\psi$ takes four discrete values over $S^1$. ................................................................. 24

2.3 Some 2D manifolds obtained by identifications of the boundary points of subsets of $\mathbb{R}^2$. Arrows on a pair of opposite edges indicate identification of the opposite points on the edges. If arrows are drawn in opposite directions, then there is a twist in the identification. ................................................................. 27

2.4 The RRT-CONNECT construction algorithm. .................................................. 28

2.5 The basic construction algorithm for sampling-based roadmaps. Note that $i$ is not incremented if $\alpha(i)$ is in collision. This forces $i$ to correctly count the number of vertices in the roadmap. ................................................................. 29

2.6 An architecture of an approach based on ISS framework. ............................... 30

3.1 A kd-tree: a) how a torus is subdivided, b) the corresponding binary tree. ........ 38

3.2 The algorithm for constructing kd-tree in a topological space $T$. ...................... 39

3.3 The algorithm portions for searching a kd-tree on the root level and internal and leaf nodes levels. ................................................................. 40
3.4 Nearest-neighbor computations are shown for 50000 data points generated at random in Euclidean spaces $\mathbb{R}^d$. The time to perform 100 queries is shown for the naive, brute-force algorithm. For other methods the construction time is added to the time required to perform 100 queries.

3.5 Nearest-neighbor computations are shown for 50000 data points generated at random in spaces $(S^1)^d$. The time to perform 100 queries is shown for the naive, brute-force algorithm. For other methods the construction time is added to the time required to perform 100 queries.

3.6 Nearest-neighbor computations are shown for 50000 data points generated at random in spaces $(\mathbb{R}^3 \times \mathbb{R}P^3)^k$. The time to perform 100 queries is shown for the naive, brute-force algorithm. For other methods the construction time is added to the time required to perform 100 queries.

3.7 This example involves moving the C-shaped object to the end of the maze. There are many narrow corridors in the configuration space, $\mathbb{R}^2 \times S^1$. The problem was solved using a RRTConCon [122].

3.8 This 56-dimensional problem involves exchanging positions of 8 L-shaped objects contained in a rectangular box. It was solved using RRTConCon [122].

3.9 This example is solved using the PRM approach [101]. The goal is to move the 3D rigid object out of the cage.

4.1 An illustration of the notions of dispersion and discrepancy for a set of points on a 2-sphere. (a) The discrepancy searches for the subset $R$ for which the deviation from the measure of $R$ to the number of samples placed inside $R$ is the largest. (b) The dispersion searches for a point $q$ on the sphere which is the farthest from the sample points.

4.2 The comparison of different sampling methods related to the problem of Section 4.1.2. The rows correspond to the desired properties of these methods. The columns represent different methods.

4.3 Different sequences of samples inside unit cubes. The Halton (a) and Hammersley (b) points achieve better discrepancy and dispersion than random (c) samples. Sukharev grid (d) and latices (e) provide optimal dispersion.

4.4 Distribution of points on the sphere $S^2$ generated by a grid (Sukharev [192]) on each spherical face.

4.5 Recursive generation of a new sample from the multiresolution grid sequence.

4.6 Different sampling methods on $S^2$. (a) 200 random samples (b) 192 Sukharev grid samples [213] (c) icosahedron samples (d) 216 HEALPix samples [76]

4.7 Base grid of the HEALPix sequence consists of 12 points. The cylindrical projection of the grid cells from $S^2$ to $(\cos(\theta), \phi)$ coordinates is shown. Each next resolution subdivides each of the spherical squares into 4 squares of equal area. Figures are borrowed from [76].

4.8 The base grid of the proposed $SO(3)$ sequence consists of 72 points. For the Hopf coordinates $(\theta, \phi, \psi)$ the projections of the grid regions on each of the coordinates are shown. Grid regions for $\psi$ are chosen according to the ordinary grid on $S^1$. The grid regions for $(\cos(\theta), \phi)$ are obtained using the HEALPix method.

4.9 Different sets of samples on $SO(3)$ (a) 2000 random samples; (b) 2048 Sukharev grid samples.

4.10 Different sets of samples on $SO(3)$ (a) 1944 icosahedral samples; (b) 1944 HEALPix samples.

4.11 Problems involving: a) moving a robot (black) from the north pole to the south pole. Multiple views of the geometry of the problem are shown (obstacles are drawn in lighter shades); b) moving a robot along the corridor.
4.12 Problems involving: (a) moving a robot from one corner of a 3D grid to the opposite corner; (b) moving an L-shaped object through the holes in the obstacles. The comparisons of the number of nodes generated by different sampling strategies are shown in the table.  

5.1 Trees constructed by an RRT-based planner and the Voronoi regions associated with the nodes of the trees. (a) A bug trap problem in high dimensions can be a challenging problem for this planner. (b) It becomes much more challenging when the region for sampling is enlarged.  

5.2 For a set of points inside a bug trap different sampling domains are shown: (a) regular RRTs sampling domain, (b) visibility Voronoi region, (c) dynamic domain.  

5.3 For a non-boundary point different sampling domains are shown: (a) regular RRTs sampling domain, (b) visible Voronoi region, (c) dynamic domain.  

5.4 For a boundary point different sampling domains are shown: (a) regular RRTs sampling domain, (b) visible Voronoi region, (c) dynamic domain.  

5.5 The dynamic domain RRT construction algorithm.  

5.6 The goal is to move a point robot out of the two-dimensional bug trap. Results for different environment sizes are shown in (1), (2), (3). Each next environment has 50 times the volume of the previous one.  

5.7 Molecule example. The task is to compute the pathway of a small molecule to the active site located inside the protein model.  

5.8 An example from the manufacturing industry. The goal is to find a collision free path dismounting a wiper motor from a car body.  

5.9 The goal in this example is to move a two link articulated body with 4 degrees of freedom from one corner of the labyrinth to another.  

5.10 The RRT-CONNECT construction algorithm.  

5.11 The DD-RRT-CONNECT construction algorithm.  

5.12 The algorithm for constructing a kd-tree for a set of points \( P \).  

5.13 The algorithm for updating the kd-tree with a new point \( p \).  

5.14 (a) An RRT with 800 nodes in two dimensional space, together with the corresponding kd-tree dynamic domain is shown. (b) An RRT with 600 nodes is growing inside the diagonal corridor. The corresponding kd-tree dynamic domains with two different \( r \)-parameters are shown.  

5.15 The algorithm portions for searching a kd-tree on the root level and internal and leaf nodes levels.  

5.16 The goal in this example is to move a closed chain with 12 links and revolute joints from the left to the right part of the environment through a narrow opening.  

5.17 The goal is to use the PUMA robot to move the toroidal object from one rod in the work space to the other.  

5.18 The goal in this example is to unfold a 3D-closed chain with 12 links amidst the cloud of obstacles.  

5.19 This experiment involves two PUMA robots holding an object, which needs to be moved from the back to the front of the environment.
Chapter 1

Introduction

The study of robotic systems involves multiple research areas. Physics, electrical engineering, and mechanical engineering allow modeling sensor fabrication, power sources and dynamics. Computer science and mathematics deal with algorithmic considerations in robotics. Robotics research offers a very active area of study, in which many open problems still need solutions, and many solutions need to be perfected before they can be applied practically in a commercial setting. Recent success of robots such as the Roomba show that robots also start entering our everyday experiences. As such, these are very exciting times for the robotics research community, which further motivates the study of robotics systems.

In general, robotics may be divided into three main areas of study: perception, actuation, and computation. Perception involves the design and study of sensing devices that provide observations of the current state of the robotic system. Actuation considers problems such as the design of control laws and inputs to drive a robotic system. Finally, computation deals with considerations such as the design of algorithms that find the correct control inputs given the state of the world as measured from the sensors. This thesis focuses on the last area, particularly, on the motion planning problem.

In a loose sense, the motion planning problem considers the search for a robot path from an initial to a goal configuration, avoiding any kind of collisions. The traditional view, which is followed in this thesis, is to consider perfect sensing (the state of the robotic system is assumed to be known), and to concentrate on the search for control inputs that drive the robot between the desired configurations.

Motion planning is a fundamental problem in robotics, which requires a conversion from a high-level description of a task to a low-level specification of a solution path. The classical problem is often referred to as the Piano Mover’s Problem. In this problem, the algorithm is required to find a path for a piano to be moved out of a house without hitting other furniture. The input to the problem is usually computer-aided design (CAD) models for both the piano and the house (see Figure 1.1). Traditionally, the problem ignores possible kinematic and dynamic constraints and primarily focuses on finding the required rotations and translations to move the piano. Extensions
Figure 1.1: The Piano Mover’s Problem. The goal is to find a path for a piano to be moved out of a house without hitting other furniture. The input is computer-aided design (CAD) models for both the piano and the house.

to the classical formulation of the motion planning problem include other specifications, such as multiple robots, sensing uncertainties, model uncertainties, differential, kinematic, and dynamics constraints.

Applications of the motion planning problem are abundant, including automotive manufacturing and assembly, video gaming industry, humanoid robotics, automotive navigation of vehicles, drug design, and many more. Due to the significant progress in the area, many difficult problems have been solved by the existing planners. The goal of this thesis is to expand the application of motion planning algorithms to even more challenging problems, and to include issues not considered by the planners that are widespread in use today. To understand the current state of the art, we start with describing how the problem has evolved since the 1970s, when the field of motion planning emerged.

1.1 Motivation and Background

Motion planning became an active area of research after the notion of configuration space (originally introduced by Lagrange in 1800s) was first used in robotics [136, 137, 138]. The study of the configuration space allowed motion planning to be viewed as a search in a high-dimensional space
that implicitly represents both obstacles and the geometry of the robot. The dimension of the space corresponds to the number of degrees of freedom of the robot. This level of abstraction allows many motion planning problems that appear different in terms of geometry and kinematics to be solved by the same planning algorithm.

It also became known in 1970s that the path planning problem is PSPACE-hard [163], and general complete algorithms were introduced by the seminal work of Schwartz and Sharir [171, 172], and Canny [37]. These complete combinatorial planners build an exact discrete representation of the free configuration space from the input geometric models. They guarantee to find a solution if it exists, or to report failure otherwise. Unfortunately, complete methods have exponential complexity in the dimension of the configuration space, and are very difficult to implement and inefficient in practice. Although Canny’s algorithm was close to optimal time complexity, these results made it seem unlikely that practical motion planning algorithms could ever be found.

Over the 1990s an alternative philosophy was developed. Instead of complete combinatorial planners, sampling-based motion planners were introduced. These methods avoid an explicit representation of the free configuration space. Instead they “probe” the configuration space using an efficient collision detector algorithm as a “black box” [20, 62, 63, 156]. Completeness is lost in such approaches to resolution completeness, or probabilistic completeness.

Sampling-based algorithms are usually divided into two general approaches: multiple-query and single-query methods. The primary philosophy behind the multiple-query methods is allow substantial precomputation time so that multiple queries for the same environment can be answered quickly. The Probabilistic Roadmap (PRM) planner [101] is one of the most influential approaches in motion planning in this respect. Numerous extensions of this method exist. For instance, some roadmaps concentrate samples in a nonuniform way, such as along the configuration space boundaries [4, 28], or the medial axis [87, 158, 207]. Other planners were primarily designed for solving problems with narrow corridors [74, 88]. The visibility approach by Simeon et al. [181] leads to a nonuniform way of sampling the configuration space which avoids oversampling. PRMs can also be extended to problems of motion planning for closed chains [51, 80, 208], multiple robots [194], and nonholonomic robots [173].

Multiple-query methods may take considerable precomputation time; thus, different approaches were developed for solving single-query problems [27, 90, 143, 118, 170]. Rapidly-exploring Random Trees (RRTs) were primarily designed for targeting single-query holonomic problems and problems with differential constraints [108, 118, 124]. The performance success of RRTs on many motion
planning problems has led to broad extensions and applications of this approach. For example, problems with complicated geometries are handled with RRT-based planners [45, 64]. Extensions of RRTs can handle manipulation problems and motions of closed articulated chains [50, 97, 98, 183]. Adaptations of RRTs for kinodynamic and nonholonomic planning also exist [29, 42, 71, 103, 112, 124]. Instead of random sampling, deterministic, resolution-complete alternatives of RRTs have been proposed [119, 132].

Next, we describe major components comprising most of the sampling-based motion planners. Some of the components have been already extensively studied, efficient implementations of which have contributed to the success of modern planners. Many of the other components are still the area of active research, and effective solutions will contribute to even further success of the motion planning field. This thesis introduces solutions to several of the components of the sampling-based motion planners.

1.1.1 Collision Detection

For motion planning problems in which the obstacles and the robot are described by thousands, or even millions of geometric primitives, constructing explicit representation of the configuration space is not feasible. Examples of such problems include automated manufacturing, automotive assembly, and drug design (see Figures 1.2 and 1.3).

The success of sampling-based motion planning can be attributed to the idea of avoiding explicit construction of the configuration space, and, instead, searching the configuration space by “probing” it with some sampling scheme. The probing is enabled by a collision detection component [20], which is often considered as a “black box” by motion planning algorithms.

Collision detection is the major bottleneck in efficiency of many modern planners. Therefore, it is important to have a thorough understanding of collision detection methods not only to improve their performance, but also to efficiently use the information they provide in motion planning algorithms [216, 212]. Excellent surveys on collision detection methods are provided in [96, 129, 130, 148]. There are many successful approaches for collision detection in the context of motion planning and other areas. Hierarchical collision detection is covered in [77, 130, 147]. The incremental collision detection mostly used in motion planning was inspired by works such as [52, 128, 147]. Numerous collision detection packages are available for use in motion planning research; one of the most widely used is the Proximity Query Package (PQP) [197].
1.1.2 Sampling Considerations for Motion Planning

As sampling-based planning algorithms were developed, it appeared to many that their efficiency was due to randomization [4, 20, 62, 63, 101, 113, 156]. However, recent developments in the field demonstrate that deterministic methods are at least as good as randomized methods, and furthermore offer a stronger guarantee of success: resolution completeness. Random sampling is almost surely (with probability 1) resolution complete. However, there is no deterministic guarantee on resolution. In some cases though, it might be valuable to verify a system down to some level of resolution, in which case random sampling might leave undesired gaps. Applications in which deterministic guarantees are needed include verification problems in areas of autonomous navigation of land and air vehicles [145, 205].

Deterministic sampling has also led to practical performance improvements in comparison to random sampling [41, 42, 73, 131, 142, 152]. This is explained by the fact that random sampling schemes require more samples to achieve any desired resolution. For recent work on deterministic sampling techniques for motion planning, see [31, 109, 120, 131, 134, 213].

The work on sampling for motion planning is highly influenced by sampling theory developed by the applied mathematics community in the context of integration and optimization problems.
Excellent overviews of the subject include [142, 152]. Deterministic sample sets and sequences were developed to replace the random sequences traditionally used for numerical integration and optimization applications; they received the name quasi-Monte Carlo to indicate the connection. Important references for further reading on sampling theory include [40, 195, 196]. Comprehensive introductions to Monte Carlo algorithms appear in [65, 100, 146].

1.1.3 Parameter Tuning in Motion Planning

As mentioned before, even though the early sampling-based motion planning algorithms obtained remarkable results on many challenging motion planning problems, they often required substantial parameter tuning.

The randomized potential field [19, 21, 113] approach uses random walks to attempt to escape local minima when best-first search becomes stuck. This allowed solving many challenging high-dimensional problems. It was one of the first sampling-based planners that developed specialized techniques beyond classical discrete search, in an attempt to better solve challenging motion planning problems. The heavy amount of parameter tuning caused most people to abandon the method. Many other proposed planners, such as Ariadne’s Clew [143, 144], expansive planner [90], suffered from the
Figure 1.4: An example of a motion planning problem involving searching a maze. The solution lies in a narrow passage in the configuration space, which usually leads to tens of thousands of configurations being generated by a motion planning algorithm. More complicated mazes may lead to generation of millions, or even billions of nodes.

problem of geometry-specific parameter tuning [170], which led to the quest for developing simple, reliable, and fast motion planners, which would be applicable to a large class of motion planning problems.

Two of such methods include Probabilistic Roadmap Methods (PRMs) [101] and Rapidly-exploring Random Trees (RRTs) [118, 123]. Both of these methods are widely used in practice today, and have been adapted to handle many extensions of the motion planning problem. RRTs have been used in several applications, and many variants have been developed [23, 30, 33, 42, 45, 49, 56, 71, 75, 97, 99, 98, 103, 127, 132, 133, 190, 191, 198, 208, 212]. The original PRM, along with its numerous extensions and variants [1, 6, 16, 17, 27, 34, 38, 49, 93, 120, 125, 126, 154, 158, 181, 199, 207, 208, 215], have been applied to problems in robotics, computer animation, and computational biology [105, 157, 186]. An experimental comparison of many of these variants appears in [73]. Some analysis of PRMs appears in [18, 90, 111].

1.1.4 Nearest Neighbor Searching

Sampling-based motion planning methods typically generate a graph in the configuration space, which is later searched for a solution path. The cost of nearest neighbor searching becomes a
bottleneck in efficient implementation of these approaches [15, 160, 214]. Motion planning algorithms sometimes require generating millions of nodes to find a solution (for example, when solving problems that involve searching mazes, see Figure 1.4), and nearest neighbor queries are typically performed at each iteration. Therefore, it is crucial to develop efficient techniques for nearest-neighbor searching in configuration spaces arising in motion planning.

There has been a significant interest in nearest-neighbor and related problems over the last couple of decades. For data sets in Euclidean spaces, kd-tree-based methods [13, 72, 187, 169, 55] proved to be one of the most effective in practice. One of the first appearances of the kd-trees is in [72], and a more modern introduction appears in [55]. Improvements to the data structure and its construction algorithm in the context of nearest-neighbor searching are described in [187]. In [13] it is shown that using kd-trees with minor modifications for finding approximate nearest neighbors allows significant improvement in running time with a very small loss in performance for higher dimensions. Other data structures for nearest-neighbor searching in Euclidean spaces are used for high-dimensional problems [91], and for dynamic data [2].

Several techniques have been developed for nearest-neighbor searching in general metric spaces [48, 86]. Practical implementations [25, 46, 107] were tested on various data sets [25, 47]. Most of these techniques consider the metric as a “black box” function provided to the algorithm. Since these techniques are more general and allow any metric space to be searched, they are usually not as efficient on Euclidean spaces as techniques designed primarily for Euclidean spaces, such as kd-trees [47].

### 1.1.5 Motion Primitives

In addition to satisfying global constraints such as avoiding obstacles, it is also crucial in most applications to satisfy local constraints which are expressed by differential equations. Motion planning problems under differential constraints can be divided into three categories. **Nonholonomic planning** was first introduced by [114], and was extended in [3, 94, 115, 116, 140, 173, 201]. **Kinodynamic planning** was introduced by [61] to refer to the problem in which both velocity and acceleration bounds need to be satisfied. Other works on this topic include [36, 43, 45, 60, 59, 68, 84, 89, 124]. **Trajectory planning** approaches decompose the problem into two parts: first planning a path for the system without considering the constraints; then, transforming the path to satisfy the differential constraints [87, 168, 174, 175, 176, 177, 178]. Planning under differential constraints is generally much harder than basic motion planning; therefore, virtually all approaches are sampling-based.
RRTs were developed for higher-dimensional problems [124]; however, they do not lead to good performance in many settings due to poor metrics that do not reflect the actual distances in the configuration space and motion primitives.

The study of motion primitives is an emerging area of research interest in motion planning under differential constraints [32]. It has been observed in several experimental efforts that computational performance is improved by orders of magnitude by having a good collection of motion primitives for a system [69, 75, 159]. These primitives usually respect the differential constraints but ignore the obstacles. However, there is no good notion of “goodness” for selecting motion primitives. Some of the approaches appear in computer graphics, in which the selection criteria is determined by the quality of the resulting animation. In [7] the motion primitives for human walking are proposed from the observations in the motion capture room. The main difficulty here is that currently there are no criteria designed for choosing the motion primitives for a system. Even though these works demonstrated the effectiveness of using motion primitives, the primitives were designed heuristically for each specific system.

1.1.6 Extensions to Closed Chains and Constrained Geometries

In many motion planning problems, the feasible subspace becomes “thin” in some directions. This is often due to kinematic closure constraints, which restrict the feasible configurations to a lower-dimensional manifold or variety. An arrangement of obstacles may also result in thin feasible configuration spaces, similar to the ones obtained from kinematic closure constraints. A motivating example for such a problem with two different scenarios is shown in Figure 1.5.

Planning for closed chains is traditionally considered as a separate class of motion planning problems, since the kinematic constraints are given a priori. Analytical approaches construct explicit geometrical and topological representation of the closure set [37, 135, 81], but are usually inefficient in practice. Practical sampling-based methods [208, 22] usually project the closure set on the subset of parameters, on which the planning is performed [80, 50]. An inverse kinematics solver is used in these approaches as a black box to get the solution back on the configuration space.

Separate techniques have been also developed for motion planning problems with constrained geometries. Several approaches were proposed recently [64, 212] for solving such problems. However, they can only handle a small set of motion planning problems in low dimensional configuration spaces.

An important direction of current research is to find efficient planners that handle such sce-
Figure 1.5: (a) Manipulation problem for a closed chain. (b) Manipulation problem for sliding a washer against a rod. The two seemingly different problems involve searching of the similar thin feasible solution spaces described by kinematic constraints in (a) and geometry of the problem in (b).

narios in a unified way, whether the constraints implicitly or explicitly define the set of feasible configurations.

1.2 Summary of Contributions and Thesis Organization

This thesis analyzes the basic motion planning problem to address critical concerns that are not covered by the motion planning algorithms in use today. We characterize and study the following components of motion planning: nearest neighbor searching, uniform deterministic sampling, and guided sampling for efficient exploration. We show that most of the modern motion planners incorporate one or more of these components. Therefore, efficient implementations of the components will improve the performance of the motion planners. The rest of the thesis is organized to present our contributions.

• Chapter 2: Problem Formulation. In this chapter we provide background notations and definition for both general motion planning, and particular problems that are solved in later chapter. The notions of configuration space, metric, and measure are defined there. The problem of motion planning in configuration spaces is also defined. We present an architecture for solving motion planning problems, called the incremental sampling and searching framework, which provides a background for issues arising in efficient implementations. Each of these
issues is covered in the later chapters.

- **Chapter 3: Efficient Nearest Neighbor Searching for Motion Planning.** The problem of efficiently answering nearest neighbor queries in configuration spaces arising in motion planning is formally defined in this chapter. We overview existing literature on the subject, and present a method based on kd-trees, which is adapted to handle the topologies and metrics of configuration spaces. We also present experimental results on motion planning problems.

- **Chapter 4: Deterministic Sampling Methods for Configuration Spaces.** Sampling issues are fundamental not only in the area of motion planning but also in related fields. In this chapter we present the general requirements on sampling in the context of motion planning. We overview the existing relevant techniques, and present our approach for sampling on configuration spaces arising in motion planning. Handling metrics and measure becomes particularly challenging for this problem. We illustrate the use of the proposed sampling methods on motion planning problems.

- **Chapter 5: Motion Planning for Highly Constrained Spaces.** In this chapter we propose a simple new planner that treats some of the pathological cases on which current motion planning methods perform poorly. Results from both Chapter 3 and 4, together with new considerations for kinematic constraints and complex geometries are used in the implementation of the method. We present our experimental results on several benchmarks, as well as many other motion planning problems.

Addressing the core issues in motion planning described above is crucial for more complete understanding of the motion planning problem. Motion planning algorithms are already widely used throughout many applications and industries. It remains only to demonstrate that a broader class of problems can be efficiently solved, which is the focus of this thesis. The improved motion planning algorithms will tackle many problems of active interest in applications. Efficiency is always an important issue in applications which involve online planning, such as mobile robotics, autonomous vehicles, and spacecraft mission planning. The DARPA Grand Challenge [39] is a good example of one such application. The problem of handling complex kinematic constraints is highly relevant to applications in humanoid robotics, and computational biology. The application areas of our work are intentionally diverse to help insure that we develop general purpose algorithms, that could have applications beyond the scope considered here. Therefore, we expect this work to have a long-term, widespread impact across many disciplines.
Chapter 2

Problem Formulation

This chapter outlines all of the problems we address in the thesis. We first define the basic motion planning problem in Section 2.1, and the spaces over which it is formulated in Section 2.2. In Section 2.3 we present the incremental sampling and searching (ISS) framework which provides a unified description of sampling-based approaches solving these problems. Finally, we define several of the components of this framework, which are major bottlenecks in many implementations. Later chapters present our solutions to each of the problems we identified as bottlenecks in this chapter.

2.1 Basic Motion Planning Problem

The basic motion planning problem requires computing a collision-free trajectory in the configuration space of one or more movable objects. Before giving a formal definition of the problem, we introduce some relevant notations. Suppose that the world, \( \mathcal{W} = \mathbb{R}^2 \) or \( \mathcal{W} = \mathbb{R}^3 \), contains an obstacle region, \( \mathcal{O} \subset \mathcal{W} \), which is a closed semialgebraic set and a subset of \( \mathcal{W} \). Assume that a robot, \( \mathcal{A} \), is also described as a closed semialgebraic set, and a subset of \( \mathbb{R}^2 \), or \( \mathbb{R}^3 \), matching the dimension of \( \mathcal{W} \).

Let \( q : \mathcal{A} \rightarrow \mathcal{W} \) be a function that defines a transformation of the robot. The image of \( q \) is

\[
q(\mathcal{A}) = \{ q(\mathcal{a}) \in \mathcal{W} \mid \mathcal{a} \in \mathcal{A} \}.
\]

(2.1)

\( q(\mathcal{A}) \) indicates all of the points in \( \mathcal{W} \) occupied by the transformed robot. If the robot consists of one body, the transformations that we consider are called rigid-body transformations, that is, the transformations that preserve distances between points, and orientation. In case of multiple bodies, or chains of bodies, we consider rigid-body transformations applied to each of the links.

The configuration space \( \mathcal{C} \) is the set of all possible transformations that could be applied to the robot. The dimension of the configuration space is the number of degrees of freedom of the robot. The configuration space which commonly arises in motion planning is either a non-Euclidean manifold or a collection of manifolds. A 2D rigid body freely translating and rotating in the plane
has the configuration space \( C = \mathbb{R}^2 \times S^1 \), in which the circle \( S^1 \) represents the 2D rotations. Three-dimensional rigid body rotations result in the configuration space which is the real projective space, \( \mathbb{RP}^3 \). Toroidal manifolds arise as the configuration spaces of unlimited revolute joints of a manipulator. In the case of multiple bodies the resulting configuration space is a Cartesian product of copies of the configuration spaces of the individual bodies. If several of the joints of a manipulator form closed loops, then the configuration space is an algebraic variety. We define some of these manifolds in detail in Section 2.2.

We now define the **obstacle region** in the configuration space, \( C_{\text{obst}} \subseteq C \), which is induced by the obstacles \( O \) as

\[
C_{\text{obst}} = \{ q \in C \mid q(A) \cap O \neq \emptyset \}. \tag{2.2}
\]

The **free configuration space** \( C_{\text{free}} \) is the space of all collision-free configurations, defined as

\[
C_{\text{free}} = C \setminus C_{\text{obst}}. \tag{2.3}
\]

If the robot and obstacles are closed sets, then \( C_{\text{obst}} \) is also closed. Since \( C \) is a topological space it follows that \( C_{\text{free}} \) is an open set [113, 119].

The motion planning problem for the robot \( A \) in the world \( W \) amidst obstacles \( O \) is referred to as the **piano mover’s problem**, or **basic motion planning problem**. We now give the formal definition.

**Definition 2.1 (Basic Motion Planning Problem)**

Given:

1. A world \( W = \mathbb{R}^2 \) (or \( \mathbb{R}^3 \)) and obstacles \( O \subseteq W \).

2. A robot \( A \subseteq \mathbb{R}^2 \) (or \( \mathbb{R}^3 \), matching the dimension of \( W \)). It may consist of one body \( A \), or a collection of \( m \) links, \( A_1, A_1, \ldots, A_m \).

3. A configuration space \( C \), partitioned into \( C_{\text{obst}} \) and \( C_{\text{free}} \) according to (2.2), and (2.3).

4. An initial state \( q_{\text{init}} \in C_{\text{free}} \) and a goal state \( q_{\text{goal}} \in C_{\text{free}} \).

A complete algorithm in finite time must return a continuous path \( \tau : [0, 1] \rightarrow C_{\text{free}} \) such that \( \tau(0) = q_{\text{init}}, \tau(1) = q_{\text{goal}}, \) or correctly report that no such path exists.

Sometimes it is useful to consider weaker notions of completeness. An algorithm is called **resolution complete** if it guarantees to find a solution in finite time, if one exists; however, if the solution does not exist, the algorithm may run forever. An algorithm is called **probabilistically complete** if the
probability of finding a solution if one exists goes to 1 as time goes to infinity. The probabilistically complete algorithms may not report that there is no solution in finite time.

The main difficulty in solving the classical motion planning problem is that it is neither straightforward nor efficient to compute an explicit representation of either $C_{\text{free}}$ or $C_{\text{obst}}$. It was shown by Reif [163] that this problem is PSPACE-hard, which implies NP-hard. The main problem is that the dimension of the configuration space is unbounded.

### 2.1.1 Extension to Constrained Geometries and Closed Kinematic Chains

Many important applications require motion planning as described in the Section 2.1 while maintaining additional constraints. These constraints may be a result of kinematic loops, or just the way obstacles are defined. We call them closure constraints. Formally, closure constraints have the form $\{|f_i(q)| \leq \epsilon_i, \forall i \in [0, \ldots m]\}$, in which $m$ is the number of constraints. The values for all $\epsilon_i$ are specified in advance. The solution to the motion planning problem from Section 2.1 must then also satisfy the closure constraints. Planning in this context requires that solution trajectories remain close to a lower dimensional subspace, defined by equations $\{|f_i(q)| = 0, \forall i \in [0, \ldots m]\}$, for which explicit parametrization is not available. In this section we formulate the motion planning problem to handle closure constraints. We also show that closure constraints arise from both kinematic loops and complex geometries. This allows us to handle these two seemingly different situations in a unified way.

Consider the configuration space $C$. For given values of $\epsilon_i \in [0, \infty), i \in [0, \ldots m]$, define the constrained space as the set of all of the configurations that satisfy the closure constraints:

$$
C_{\text{con}} = \{q \| f_i(q) \leq \epsilon_i, \forall i \in [0, \ldots m]\}. \tag{2.4}
$$

The free space, $C_{\text{free}}$, is defined as the set of all configurations $q \in C$, that satisfy the collision constraints (2.3). The valid space is the closure of the free space, $C_{\text{val}} = C_{\text{free}} \cup \partial C_{\text{free}}$. The feasible space is defined as $C_{\text{feas}} = C_{\text{con}} \cap C_{\text{val}}$; it contains the configurations that satisfy the constraints and avoid penetration into obstacles. The motion planning problem is then defined on $C_{\text{feas}}$ as follows:

**Definition 2.2 (Motion Planning Problem with Closure Constraints)**

*Given:*

1. A world $W$, obstacles $O \subset W$, and a robot $A$ from Definition 2.1.
2. Configuration space \( \mathcal{C} \), together with feasible configuration space \( \mathcal{C}_{\text{feas}} \).

3. An initial state \( q_{\text{init}} \in \mathcal{C}_{\text{feas}} \) and a goal state \( q_{\text{goal}} \in \mathcal{C}_{\text{feas}} \).

A complete algorithm in finite time must return a continuous path \( \tau : [0, 1] \to \mathcal{C}_{\text{feas}} \) such that \( \tau(0) = q_{\text{init}}, \tau(1) = q_{\text{goal}} \), or correctly report that no such path exists.

The notions of resolution completeness, and probabilistic completeness can be defined similarly to Section 2.1. It is important to note that if the closure constraints are not explicitly defined (that is, if \( \mathcal{C}_{\text{feas}} = \mathcal{C}_{\text{free}} \)) the free configuration space may still have similar topological properties as a feasible space for which the closure constraints are provided. It can happen if the obstacles define the region in the configuration space similar to the region defined by the closure constraints, \( |f_i(q)| \leq \epsilon_i \). For example, is planning for a closed chain different from planning a sliding motion for a washer against a rod? An illustration of two such instances of the motion planning problem is provided on Figure 1.5. Both of these seemingly different problems satisfy the Definition 2.2.

2.1.2 Special Case: Linkages with Closed Chains

One instance of the motion planning problem with closure constraints can be obtained when planning for linkages with closed kinematic chains. Consider a chain of \( n \) links, such that each link \( L_i \) is a 3D rigid body. If two links \( L_i \) and \( L_k \) are attached to each other, the place at which they attach is called the joint \( J_{i,k} \). Call \( L \) the collection of all of the links in the chain, and \( J \) the collection of all of the joints. The underlying graph, \( G(J,L) \), in which the vertices correspond to all of the joints, and the edges are the corresponding links, represents the topology of the linkage. The underlying graph has cycles if and only if the linkage contains closed loops.

Each joint \( J_{i,k} \) carries information about the type of the attachment (revolute, spherical, etc.), which is often expressed as the homogeneous transformation matrix from the coordinate frame of one link to the frame of another [113, 119]. The variables in the matrix express the freedom of movement of the link around the joint with another link. This leads to a parametrization of the linkage (for example, the Denavit-Hartenburg representation [83]).

Setting each of the parameters to a fixed value results in a vector of real values \( q \) that represents a fixed configuration of the linkage. If \( G(J,L) \) contains cycles, then not all of the configurations \( q \in \mathcal{C} \) yield a valid position and orientation of each of the links in the chain. Only configurations \( q \in \mathcal{C} \) that satisfy the closure constraints of the form \( |f_i(q)| = 0 \) result in valid configurations of the linkage. The closure constraints can be obtained by writing down two homogeneous transformation matrices.
for a coordinate frame of a link in each loop of a closed chain. Each of the matrices corresponds to
the two different paths to the link alone the loop. The closure constraint can then be obtained by
forcing the frame of the link to be the same, regardless of the path that was chosen.

Since the configurations that satisfy the closure constraints $|f_i(q)| = 0$ are defined implicitly, they
often can not be expressed in closed form. It is also natural to assume that some numerical error, $\epsilon$, is
allowed for the configurations on the closure set. The value for $\epsilon$ is usually defined based on the
particular application. In this work we consider, therefore, the relaxation of the closure constraints
for closed chains $|f_i(q)| = 0$ to $|f_i(q)| \leq \epsilon$.

2.2 Metrics, and Measure on Configuration Spaces

Once the motion planning problem has been defined, the next step is to carefully define the properties
of the spaces over which planning occurs. Virtually all sampling-based planning algorithms require
a function that measures distances between points on the configuration space, which often results in
a metric space [119]. In some sampling-based algorithms, volumes of subsets of $C$ need to be defined,
which results in a measure space. Next we briefly review definitions of metric and measure spaces.
We then outline the most common configuration spaces arising in motion planning, and metrics and
measures commonly used over these spaces.

2.2.1 Metric Space

To define a metric space the notion of neighborhoods, or open sets in a topological space need to be
defined first [8].

A set $X$ is called a topological space if there is a collection of subsets of $X$ called open sets for
which the following axioms hold:

1. The union of a countable number of open sets is an open set.

2. The intersection of a finite number of open sets is an open set.

3. Both $X$ and $\emptyset$ are open sets.

The following definition and axioms are used to create a function that converts a topological
space into a metric space. A metric space $(X, \rho)$ is a topological space $X$ equipped with a function
$\rho: X \times X \to \mathbb{R}$ such that for any $a, b, c \in X$:

1. Nonnegativity: $\rho(a, b) \geq 0$. 

16
2. **Reflexivity**: \( \rho(a, b) = 0 \) if and only if \( a = b \).

3. **Symmetry**: \( \rho(a, b) = \rho(b, a) \).

4. **Triangle inequality**: \( \rho(a, b) + \rho(b, c) \geq \rho(a, c) \).

The function \( \rho \) defines distances between points in the metric space, and each of the four conditions on \( \rho \) agrees with our intuitions about distance. The final condition implies that \( \rho \) is optimal in the sense that the distance from \( a \) to \( c \) will always be less than or equal to the total distance obtained by traveling through an intermediate point \( b \) on the way from \( a \) to \( c \).

### 2.2.2 Measure Space

Now we briefly describe how to define volume in a metric space. Measure can be considered as a function that produces real values for subsets of a metric space, \((X, \rho)\).

**Definition 2.3** A collection \( \mathcal{B} \) of subsets of \( X \) is called a \( \sigma \)-algebra if the following axioms are satisfied:

1. The empty set \( \emptyset \) is in \( \mathcal{B} \).
2. If \( B \in \mathcal{B} \), then \( X \setminus B \in \mathcal{B} \).
3. For any collection of a countable number of sets in \( \mathcal{B} \), their union must also be in \( \mathcal{B} \).

Note that the last two conditions together imply that the intersection of a countable number of sets in \( \mathcal{B} \) is also in \( \mathcal{B} \). The sets in \( \mathcal{B} \) are called the **measurable sets**.

**Definition 2.4** Using \( \mathcal{B} \), a measure \( \mu \) is now defined as a function \( \mu : \mathcal{B} \to [0, \infty] \) such that the measure axioms are satisfied:

1. For the empty set, \( \mu(\emptyset) = 0 \).
2. For any countable collection, \( E_1, E_2, E_3, \ldots \), of pairwise disjoint, measurable sets, let \( E \) denote their union. The measure \( \mu \) must satisfy

\[
\mu(E) = \sum_i \mu(E_i),
\]  

(2.5) in which \( i \) counts over the whole collection.
For a metric space $X$ consider an open ball of radius $r \in \mathbb{R}$ with center at $y \in X$

$$B(y, r) = \{x \in X \mid \rho(x, y) < r\}.$$  \hspace{1cm} (2.6)

The measure on $X$ can then be defined using the set of all open balls $B(y, r)$, for all $y \in X, r \in \mathbb{R}$, to form a $\sigma$-algebra, called the Borel sets, or Borel $\sigma$-algebra [67].

### 2.2.3 Common Spaces Arising in Motion Planning

Now we describe the most common spaces arising in motion planning [119]. For each of the spaces we describe topology, metric, measure, and useful representations that are most commonly used in the context of motion planning.

#### Euclidean Space

The *Euclidean space*, $\mathbb{R}^d$ [166], arises as a configuration space for translating rigid bodies. The distance between two points $p, q \in \mathbb{R}^d$ is denoted as

$$\rho_{\mathbb{R}^d}(q, p) = \|q - p\|,$$

in which $\|\cdot\|$ is the Euclidean $L_2$ norm in $\mathbb{R}^d$. The measure used on $\mathbb{R}^d$ is the *Lebesgue measure*, which is the Borel measure with respect to $L_2$ norm [67]. Lebesgue measure becomes the standard notion of length in $\mathbb{R}$, area in $\mathbb{R}^2$, and volume in $\mathbb{R}^d$ for $d \geq 3$.

Euclidean space is the simplest and most common space arising in motion planning. In practice, Euclidean space is often used as a local approximation of the configuration space, because most configuration spaces are manifolds, that is, locally homeomorphic to $\mathbb{R}^d$. The structure of Euclidean space is so simple that many notions, such as measure, metric, and uniformity are intuitive. This makes the algorithms for Euclidean spaces considerably simpler than for other more complex configuration spaces. However, careful consideration of the structure of the configuration space leads to more efficient and accurate methods in motion planning.

#### Spheres

*Spheres* are also common spaces arising in motion planning. We denote as $S^d$ a $d$-dimensional sphere embedded in $\mathbb{R}^{d+1}$. There are many different coordinate systems that can be used to represent
spheres. Here, we consider two most common ones: \textit{Cartesian coordinates}, and \textit{hyperspherical coordinates}.

- \textbf{Cartesian Coordinates for }$S^d$. The sphere can be represented using Cartesian coordinates as $S^d = \{ x \in \mathbb{R}^{d+1} \mid \|x\| = 1 \}$. The metric on $S^d$ is then defined as the length of the geodesic arc between the points on the surface of the sphere:

$$
\rho_{S^d}(q, p) = \cos^{-1}(q \cdot p),
$$

in which $(q \cdot p)$ denotes the inner product for vectors in $\mathbb{R}^{d+1}$. Intuitively, the inner product corresponds to the cosine of the length of the shortest arc of the great circle going through the two points.

Another common metric defines the distance between two points as the length of the connecting segment in the ambient space $\mathbb{R}^{d+1}$:

$$
\rho_{S^d}(q, p) = \|q - p\|.
$$

The metric computation is very convenient, and computationally efficient using the Cartesian coordinates. However, the surface measure is better expressed using hyperspherical coordinates, which we define next.

- \textbf{Hyperspherical Coordinates for }$S^d$. Spheres can also be represented using hyperspherical coordinates.

The angular coordinates $(\phi_1, \phi_2, ..., \phi_d)$ relate to the above Cartesian coordinates in the following way:

$$
x_1 = \cos(\phi_1),
$$
$$
x_2 = \sin(\phi_1) \cos(\phi_2),
$$
$$
x_3 = \sin(\phi_1) \sin(\phi_2) \cos(\phi_3),
$$
$$
\ldots
$$
$$
x_d = \sin(\phi_1) \cdots \sin(\phi_{d-1}) \cos(\phi_d),
$$
$$
x_{d+1} = \sin(\phi_1) \cdots \sin(\phi_{d-1}) \sin(\phi_d),
$$

in which the last angle $\phi_d$ has a range of $2\pi$, while the other angles have a range of $\pi$. 

19
The volume element of the $d$-sphere defines the surface measure for $S^d$, and has the following expression [66]:

$$d_{S^d}V = \sin^{d-1}(\phi_1) \sin^{d-2}(\phi_2) \cdots \sin(\phi_{d-1}) \, d\phi_1 \, d\phi_2 \cdots d\phi_d.$$  \hfill (2.11)

**Special Orthogonal Group $SO(2)$**

$SO(2)$ arises from rigid rotations in the plane. Consider the set of all rotations of a 2D rigid body around the origin in $\mathbb{R}^2$. Such rotations are linear transformations that preserve orientation and the lengths of vectors in $\mathbb{R}^2$. The set of all such rotations is called $SO(2)$. It is well known that $SO(2)$ is a Lie group, which means it is both a group and a manifold. The group operation is composition. (Similarly, $SO(3)$ is defined for rigid rotations around origin in $\mathbb{R}^3$, which we discuss in detail in the next subsection.)

The group $SO(2)$ is diffeomorphic to the circle $S^1$, and, therefore, to a unit interval with identified endpoints:

$$SO(2) \cong [0, 1]/(0 \sim 1).$$  \hfill (2.12)

The metric for two points $p, q \in SO(2)$ is defined as

$$\rho_{SO(2)}(p, q) = \min(|q - p|, 1 - |q - p|).$$  \hfill (2.13)

If $SO(2)$ is represented by unit complex numbers, then it is a subspace of $\mathbb{R}^2$ given by \{(a, b) \in $\mathbb{R}^2 \mid a^2 + b^2 = 1\}. Next we define several metrics using this representation. For any pair of points $(a_1, b_1)$ and $(a_2, b_2)$ let

$$\rho_{SO(2)}(a_1, b_1, a_2, b_2) = \sqrt{(a_1 - a_2)^2 + (b_1 - b_2)^2}.$$  \hfill (2.14)

This metric does not give the distance traveling along the circle; it instead takes a shortcut by computing the length of the line segment in $\mathbb{R}^2$ that connects the two points. An alternative metric is obtained by

$$\rho_{SO(2)}(a_1, b_1, a_2, b_2) = \cos^{-1}(a_1 a_2 + b_1 b_2),$$  \hfill (2.15)

for two points $(a_1, b_1)$ and $(a_2, b_2)$. 

20
Special Orthogonal Group $SO(3)$

The special orthogonal group $SO(3)$ arises from rotations around the origin in $\mathbb{R}^3$. $SO(3)$ is a Lie group, with the group action being the composition of rotations.

- **Topology of $SO(3)$**. The special orthogonal group $SO(3)$ is diffeomorphic to the real projective space $\mathbb{RP}^3$, which is hard to visualize, because it can not be embedded in $\mathbb{R}^3$. Fortunately, it can be represented as the 3-sphere, $S^3$ embedded in $\mathbb{R}^4$ with antipodal points identified:

\[
\mathbb{RP}^3 \cong S^3/(x \sim -x).
\]  

(2.16)

It is said that the 3-sphere is a double covering of $\mathbb{RP}^3$, because one point of the projective space has two corresponding points on the 3-sphere.

- **Haar Measure on $SO(3)$**. Up to a scalar multiple, there exists a unique measure on $SO(3)$ that is invariant with respect to the group action, which is called the Haar measure. That is, the Haar measure of a set is equal to the Haar measure of all of the rotations of the set. In our particular situation, we can think of the Haar measure as being invariant under all orthogonal coordinate changes [44].

It is important to note that the Haar measure is an intrinsic property of $SO(3)$ which comes from the group structure, and is independent of its topological structure.

We have not used any coordinate system or parametrization of $SO(3)$ yet. One has to use extreme caution when expressing the measure in terms of any of the coordinate systems we describe next. Not all of these naturally preserve the Haar measure.

- **Orthogonal Matrices**. The elements of $SO(3)$ are defined as $3 \times 3$ orthogonal matrices with determinant $+1$. The group operation is multiplication of matrices. Because rotation matrices are less efficient and less numerically stable than quaternions, they are generally used less often than quaternions, which we describe next.

- **Quaternions**. One of the most useful representations of the $SO(3)$ is the set of quaternions. Let $x = (x_1, x_2, x_3, x_4) \in \mathbb{R}^4$ be a unit ($\|x\| = 1$) quaternion $x_1 + x_2i + x_3j + x_4k$ representing a 3D rotation. Because of the topological relationship between the projective space and the 3-sphere, once the identifications of the opposite points on the 3-sphere are taken into account, metrics similar to those defined for the 3-sphere can be used for the projective space. Moreover, such metrics will respect the Haar measure on $SO(3)$. 

21
The most natural way to define a metric for any two quaternions \( x, y \in SO(3) \) is as the length of the shortest arc between \( x \) and \( y \) on the 3-sphere:

\[
\rho_{SO(3)}(x, y) = \cos^{-1}|(x \cdot y)|, \tag{2.17}
\]

in which \((x \cdot y)\) denotes the dot product for vectors in \( \mathbb{R}^4 \), and the absolute value, \(|.|\), guarantees that the shortest arc is chosen among the identifications of the two quaternions (for a more detailed explanation, see [109]).

Similarly to metrics (2.8) and (2.9) on spheres, another metric can be defined:

\[
\rho_{SO(3)}(x, y) = \min \{ \|x - y\|, \|x + y\| \}, \tag{2.18}
\]

in which the two arguments of the \( \min \) correspond to the distances from \( x \) to \( y \) and \( -y \), respectively. The \( x + y \) appears because \( y \) was negated to yield its antipodal point, \( -y \). This metric measures the length of line segments that cut through the interior of \( S^3 \), as opposed to traveling along the surface.

Quaternion representation is also very useful for calculating the composition of rotations, which is expressed as the multiplication of quaternions. Any rotation invariant surface measure on \( S^3 \) naturally preserves the Haar measure for \( SO(3) \) and can be used for quaternions. However, the surface measure is not straightforwardly expressed using quaternions. Next we describe hyperspherical and Hopf coordinates, which are more convenient for measuring the volume of surface regions.

- **Hyperspherical Coordinates for \( SO(3) \).** Because of the relationship between the 3-sphere and \( \mathbb{RP}^3 \), hyperspherical coordinates can be used for \( SO(3) \). Consider a rotation \((\theta, \phi, \psi)\), in which \( \psi \) has a range of \( \pi/2 \) (to compensate for identifications on the 3-sphere), \( \theta \) has a range of \( \pi \), and \( \phi \) has a range of \( 2\pi \). For each \( \psi \), the ranges of \( \theta \) and \( \psi \) define a 2-sphere of radius \( \sin(\psi) \). The quaternion \( x = (x_1, x_2, x_3, x_4) \) corresponding to the rotation \((\theta, \phi, \psi)\) can be obtained using the formula:
The full range of the hyperspherical coordinate \( \psi \in [0, \pi/2] \) is shown while the coordinates \((\theta, \phi)\) form a discretization of size 20 over \( S^2 \). (b) The half-spheres show the ranges of the hyperspherical coordinates \((\theta, \phi)\), while \( \psi \) takes four discrete values over \([0, \pi/2]\).

\[
\begin{align*}
x_1 &= \cos(\psi), \\
x_2 &= \sin(\psi) \cos(\theta), \\
x_3 &= \sin(\psi) \sin(\theta) \cos(\phi), \\
x_4 &= \sin(\psi) \sin(\theta) \sin(\phi).
\end{align*}
\]

(2.19)

The volume element of \( SO(3) \) that defines the Haar measure has the following expression in hyperspherical coordinates:

\[
dV = \sin^2(\psi) \sin(\theta) d\theta d\phi d\psi.
\]

(2.20)

This representation is not convenient for integration though, because of the complicated expression for the Jacobian. Hyperspherical coordinates are also cumbersome for computing composition of rotations.

- **Hopf Coordinates for \( SO(3) \).** As opposed to spherical coordinates for hyperspheres, the *Hopf coordinates* are unique for the 3-sphere (and some other \( d \)-spheres for curtain \( d \)), and thus for \( \mathbb{RP}^3 \). They naturally describe the intrinsic structure of both the 3-sphere and \( \mathbb{RP}^3 \) and provide a natural tool for obtaining uniform distributions on these spaces.

The *Hopf fibration* describes \( \mathbb{RP}^3 \) in terms of a circle \( S^1 \) and an ordinary 2-sphere \( S^2 \). Intu-
Figure 2.2: Visualization of the Hopf coordinates on $SO(3)$. (a) The full range of the Hopf coordinate $\psi \in [0, 2\pi]$ is shown while the coordinates $(\theta, \phi)$ form a discretization of size 12 over $S^2$. (b) The spheres show the ranges of the Hopf coordinates $(\theta, \phi)$, while $\psi$ takes four discrete values over $S^1$.

Intuitively, $\mathbb{RP}^3$ is composed of non-intersecting fibers, such that each fiber is a circle corresponding to the 2-sphere. This fiber bundle structure is denoted as $\mathbb{RP}^3 \cong S^1 \otimes S^2$. The Hopf fibration has the important property of locally being a product space. The projective space, $\mathbb{RP}^3$, however, is not (globally) a product of $S^2$ and $S^1$. Intuitively, $\mathbb{RP}^3$ is the product of $S^2$ and $S^1$ similarly to the way the Möbius band is locally the product of an interval and a circle $S^1$. That is, locally a sequence of coordinates from each subspace results in a global parametrization of the space, whereas the global embedding into the Euclidean space does not have the Cartesian product structure. The Hopf coordinates can also be used for the 3-sphere, because of the topological relationship between the 3-sphere and $\mathbb{RP}^3$.

Each rotation in Hopf coordinates can be written as $(\theta, \phi, \psi)$, in which $\psi$ parametrizes a circle $S^1$ and has a range of $2\pi$. The ranges of $\theta$ and $\phi$ are $\pi$ and $2\pi$ respectively, and they represent spherical coordinates for $S^2$. The transformation to a quaternion $x = (x_1, x_2, x_3, x_4)$ can be expressed using the formula:

$$
\begin{align*}
    x_1 & = \cos(\theta/2) \cos(\psi/2), \\
    x_2 & = \cos(\theta/2) \sin(\psi/2), \\
    x_3 & = \sin(\theta/2) \cos(\phi + \psi/2), \\
    x_4 & = \sin(\theta/2) \sin(\phi + \psi/2).
\end{align*}
$$

(2.21)
The volume element on $\mathbb{R}P^3$, which respects the Haar measure, is then defined as the surface volume on $S^3$:

$$dV = \sin \theta \, d\theta \, d\phi \, d\psi.$$  \hfill (2.22)

Note that $\sin \theta \, d\theta \, d\phi$ represents the surface area on the 2-sphere, and $d\psi$ is the length element on the circle. The formula (2.22) additionally demonstrates that the volumes from the two subspaces, $S^2$ and $S^1$, are simply multiplied to obtain the volume on $SO(3)$. The Hopf coordinates, though, are not convenient for expressing compositions of rotations.

• **Axis-Angle Representation for $SO(3)$**. One of the most intuitive ways to represent rotations is by using Euler’s theorem [44], which states that every 3D rotation is a rotation by some angle $\theta$ around a unit axis $n = (n_1, n_2, n_3), \|n\| = 1$. The transformation from angle and axis representation to quaternions is achieved by using this formula:

$$
\begin{align*}
    x_1 &= \cos(\theta/2), \\
    x_2 &= \sin(\theta/2)n_1, \\
    x_3 &= \sin(\theta/2)n_2, \\
    x_4 &= \sin(\theta/2)n_3.
\end{align*}$$  \hfill (2.23)

The angle and axis representation is useful for visualizing the projective space in 3D. Each rotation is drawn as a vector with direction $n$ and a magnitude corresponding to $\theta$ (a multiple or a function of $\theta$ can be used; see Chapter 4.3.4 and [44, 150, 210]). Figures 2.1 and 2.2 show the visualization of the spherical and Hopf coordinates on $SO(3)$ using the angle and axis representation. From this visualization one can immediately notice the singularities produced by the spherical coordinates. It is also possible to see the advantage of using Hopf coordinates from this visualization, which does not introduce singularities. The circles represented by the range of the variable $\psi$ are non-intersecting; they uniformly cover the $SO(3)$. The fiber structure formed by these circles is also seen on Figure 2.2.

• **Representation with Euler Angles**. Sometimes Euler angles are used for representing 3D rigid rotations instead of quaternions. In this case, each rotation is represented as a vector

$$\mathbf{x} = (x_1, x_2, x_3), x_i \in [-\pi, \pi]/-\pi \sim \pi.$$  \hfill (2.24)

Since the topology of the space in this case is $S^1 \times S^1 \times S^1$, Euler angles do not correctly
capture the structure of the set of rotations. There are many detrimental consequences of this. Special tricks (see [109]) are needed to implement metric and measure that preserve Haar measure. Moreover, Euler angles are harder to compose, compared to quaternions, and also present the problem of singularities and the gimbal lock [179]. In the rest of the thesis we avoid using Euler angles, and use either quaternions, or Hopf coordinates for representing rotations.

**Cartesian Products**

Configuration spaces are often constructed from Cartesian products of the spaces we have described above, especially in the case of multiple bodies. Let \((X, \rho_X)\) and \((Y, \rho_Y)\) be two metric spaces. A metric space \((Z, \rho_Z)\) can be constructed for the Cartesian product \(Z = X \times Y\) by defining the metric \(\rho_Z\) as

\[
\rho_Z(z, z') = \rho_Z(x, y, x', y') = c_1 \rho_X(x, x') + c_2 \rho_Y(y, y'),
\]

in which \(c_1 > 0\) and \(c_2 > 0\) are arbitrary positive real constants, and \(x, x' \in X\) and \(y, y' \in Y\). Each \(z \in Z\) is represented as \(z = (x, y)\).

**Polygonal Schemas**

Many of the configuration spaces that we have defined in this Section have additional topological structure. These \(d\)-dimensional configuration spaces can be represented by defining a subset of \(\mathbb{R}^d\) and identifying appropriate pairs of boundary points to obtain the desired topology (such as \(S^1 \cong [0, 1]/(0 \sim 1)\)). Such configuration spaces are called *polygonal schemas*. Examples of several two-dimensional manifolds obtained by identifying points on the unit square or unit circle in the plane are shown in Figure 2.3. Cylinders (Figure 2.3 (a)) arise from from rigid rotations and translations in the plane. Toroidal configuration spaces (Figure 2.3 (b)) arise from two-dimensional kinematic chains. The metric and measure definitions for all of these spaces were presented earlier in this Section.
2.3 Incremental Sampling and Searching (ISS) Framework for Motion Planning

Most of the sampling-based motion planning algorithms follow the same template. In this section we present this template, which is called the incremental sampling and searching (ISS) framework for motion planning problems [119]. In Section 2.4 we characterize the crucial components that need to be considered for efficient implementation of the framework.

1. **Initialization:** Let $G(V, E)$ represent an undirected search graph, for which the vertex set $V$ contains a vertex for $q_{\text{init}}$ and possibly other states in $C_{\text{free}}$ (or $C_{\text{feas}}$, if closure constraints are present), and the edge set $E$ is empty.

2. **Selection Method:** Choose a configuration $q_r$ according to the selection method. Choose a vertex $q_{\text{cur}} \in V$ for expansion in the direction of $q_r$; $q_r$ may not be a vertex of $G$.

3. **Local Planning Method:** Generate an interpolated path, $s(t) : [0, T] \rightarrow C_{\text{free}}(C_{\text{feas}})$ between the configurations $q_{\text{cur}}$ with $q_r$, such that $s(0) = q_{\text{cur}}$, and $s(T) = q_r$. Using the collision detection algorithm the trajectory must be verified to be collision free. If this step fails go to Step 2.

4. **Insert an Edge in the Graph:** Insert $s(t)$ into $E$. If $q_r$ is not already in $V$, it is added. If $q_{\text{cur}}$ lies in the interior of an edge trajectory for some $s \in E$, then $s$ is split by the introduction of a new vertex at $q_{\text{cur}}$.

5. **Check for a Solution:** Determine whether $G$ encodes a solution path. In some applications, a small gap in the state trajectory may be tolerated.
BUILD_RRT(q_{init})
1 \ T_{init}(q_{init});
2 for k = 1 to K do
3 \quad q_{rand} \leftarrow \text{RANDOM_CONFIG();}
4 \quad q_{near} \leftarrow \text{NEAREST_NEIGHBOR(q_{rand}, \ T);}
5 \quad \text{if CONNECT(} T, q_{rand}, q_{near}, q_{new} \text{) then}
6 \quad \quad \ T.add\_vertex(q_{new});
7 \quad \quad \ T.add\_edge(q_{near}, q_{new});
8 \quad \text{Return } T;

Figure 2.4: The RRT-CONNECT construction algorithm.

6. **Return to Step 2:** Iterate unless a solution has been found or some termination condition is satisfied. In the latter case, the algorithm reports failure.

A sample architecture of an algorithm which follows the ISS framework is shown on Figure 2.6. Many sampling-based motion planning methods follow the ISS framework [21, 90, 93, 101, 123, 143, 154, 170, 181, 199]. We describe the most practical implementations of the framework, RRT and PRM methods, in more detail next.

### 2.3.1 RRT

RRTs were originally introduced in [118, 121]. Starting at a given initial configuration, RRTs incrementally search the configuration space for a path connecting the initial and the goal configurations. At each iteration a new configuration is sampled and the extension from the nearest node in the tree toward this sample is attempted. If the extension succeeds, a new node in the tree is created.

There are several planners that exploit the exploration properties of the basic RRTs, such as the RRT-CONNECT planner (the pseudocode is shown on Figure 5.10). Bidirectional versions of RRTs exist (bi-RRTs), which alternate execution of the basic algorithm for two trees growing from the initial and the goal configurations, and put some additional bounds on the sizes of each of the trees (bidirectional balanced RRTs).

RRT exploration is determined by the Voronoi diagram of the nodes in the tree. The probability that a node will be chosen for an extension is proportional to the volume of its Voronoi region. Therefore, the RRT tends to rapidly grow in the unexplored regions of the configuration space.

### 2.3.2 PRM

Previously, it was assumed that a single initial-goal pair was given to the planning algorithm. Suppose now that numerous initial-goal queries will be given to the algorithm, while keeping the
BUILD_ROADMAP
1 \( G.\text{init}() \); \( i \leftarrow 0 \);
2 \textbf{while} \( i < N \)
3 \quad \textbf{if} \( \alpha(i) \in C_{\text{free}} \) \textbf{then}
4 \quad \quad \( G.\text{add\_vertex}(\alpha(i)); \ i \leftarrow i + 1 \);
5 \quad \textbf{for each} \( q \in \text{NEIGHBORHOOD}(\alpha(i), G) \)
6 \quad \quad \textbf{if} \ ((\text{not } G.\text{same\_component}(G(i), q)) \textbf{ and } \text{CONNECT}(\alpha(i), q)) \textbf{then}
7 \quad \quad \quad G.\text{add\_edge}(\alpha(i), q);\)

Figure 2.5: The basic construction algorithm for sampling-based roadmaps. Note that \( i \) is not incremented if \( \alpha(i) \) is in collision. This forces \( i \) to correctly count the number of vertices in the roadmap.

robot model and obstacles fixed. This leads to a \textit{multiple-query} version of the motion planning problem. In this case, it makes sense to invest substantial time to preprocess the models so that future queries can be answered efficiently. The goal is to construct a topological graph called a \textit{roadmap}, which efficiently solves multiple initial-goal queries. Intuitively, the paths on the roadmap should be easy to reach from each of \( q_{\text{init}} \) and \( q_{\text{goal}} \), and the graph can be quickly searched for a solution. The general framework presented here was mainly introduced in [101] under the name \textit{probabilistic roadmaps (PRMs)}. The probabilistic aspect, however, is not important to the method. Therefore, we call this family of methods \textit{sampling-based roadmaps}.

Once again, let \( G(V, E) \) represent a topological graph in which \( V \) is a set of vertices and \( E \) is the set of paths that map into \( C_{\text{free}} \). Under the multiple-query philosophy, motion planning is divided into two phases of computation:

\textbf{Preprocessing Phase:} During the preprocessing phase, substantial effort is invested to build \( G \) in a way that is useful for quickly answering future queries. For this reason, it is called a \textit{roadmap}, which in some sense should be accessible from every part of \( C_{\text{free}} \).

\textbf{Query Phase:} During the query phase, a pair, \( q_{\text{init}} \) and \( q_{\text{goal}} \), is given. Each configuration must be connected easily to \( G \) using a local planner. Following this, a discrete search is performed to obtain a sequence of edges that forms a path from \( q_{\text{init}} \) to \( q_{\text{goal}} \).

2.4 Efficient Implementation of the ISS Framework

Next we outline the components of the framework that are crucial for implementing it efficiently, and that we address in this thesis. Later chapters cover each of the components in detail.
2.4.1 Efficient Nearest Neighbor Searching

Several of the methods in the ISS framework perform a search for the nearest state as part of the extension criteria in Step 2. Efficient implementation of the nearest-neighbor search procedure can dramatically improve the performance of such algorithms. Instead of a brute force linear time algorithm many data structures were developed to speed up the computations [13, 48, 86, 91]. These techniques, however, are developed exclusively for Euclidean spaces and cannot be applied directly to path planning algorithms because of the topologies of configuration spaces. The particular challenge associated with motion planning problems is that the configuration space, $C$, is usually a non-Euclidean manifold or a collection of manifolds. An appropriate metric needs to be defined, and the search for nearest neighbors must be performed with respect to the metric and topology of the space. We address this problem in Chapter 3.

2.4.2 Uniform Deterministic Sampling Methods

At each iteration of the sampling-based methods (Step 2) a configuration is selected according to some sampling requirement. Many of the planning algorithms at this step require uniform sampling over non-Euclidean manifolds, such as spheres, and rotation group $SO(3)$. Although most existing motion planning methods currently use random sampling [4, 20, 62, 63, 101, 121, 113, 156], they are limited to probabilistic forms of completeness. Therefore, designing deterministic sampling methods over the spaces arising in motion planning is required for resolution completeness guarantees. The
topology of the configuration spaces arising in motion planning present unique challenges in defining the appropriate uniformity measure and developing deterministic sampling methods. These issues are discussed in Chapter 4.

2.4.3 Guided Sampling for Efficient Exploration

If closure constraints are present, as was described in Section 2.1.1, the valid trajectories only belong to a lower dimensional subspace of the configuration space. In such cases special sampling techniques are required. Since the closure constraints do not explicitly define valid configurations, it is usually impossible to obtain an analytical representation. Naturally, sampling methods are used for computing valid configurations. Designing efficient sampling methods in this setting presents several challenges: uniformity measures, sampling criteria, and efficiency are major issues in developing motion planning methods. Chapter 5 presents our approach to these problems.
Chapter 3

Efficient Nearest Neighbor Searching for Motion Planning

The cost of nearest-neighbor calls is one of the bottlenecks in the performance of sampling-based motion planning algorithms. Therefore, it is crucial to develop efficient techniques for nearest-neighbor searching in configuration spaces arising in motion planning. In this chapter we present an algorithm that we have developed and implemented for performing nearest-neighbor queries in Cartesian products of \( \mathbb{R}, S^1 \) and \( \mathbb{RP}^3 \), the most common topological spaces in the context of motion planning. Our approach extends the algorithm based on kd-trees, called ANN, developed by Arya and Mount for Euclidean spaces. We prove the correctness of the algorithm and illustrate substantial performance improvement over the brute-force approach and several existing nearest-neighbor packages developed for general metric spaces. Our experimental results demonstrate a clear advantage of using the proposed method for both probabilistic roadmaps (PRMs) and Rapidly-exploring Random Trees (RRTs).

3.1 Introduction

Nearest-neighbor searching is a fundamental problem in many applications outside of robotics, such as pattern recognition, statistics, and machine learning. It is also an important component the ISS framework of sampling-based motion planning approaches, as we have shown in Chapter 2. Probabilistic roadmap approaches (Chapter 2.3.2), build a graph of collision-free paths that attempts to capture the connectivity of the configuration space. The vertices represent configurations that are generated using random sampling, and attempts are made to connect each vertex to nearby vertices. Some roadmaps contain thousands of vertices, which can lead to substantial computation time for determining nearest neighbors vertices. Approaches based on Rapidly-exploring Random Trees (Chapter 2.3.1) rely even more heavily on nearest neighbors. An RRT is a tree of paths that is grown incrementally. In each iteration, a random configuration is chosen, and the RRT vertex that is closest (with respect to metrics in Chapter 2.2) is selected for expansion. An attempt is made to
connect the RRT vertex to the randomly-chosen state.

An approach that efficiently finds nearest neighbors can dramatically improve the performance of these path planners. Several packages exist, such as ANN ([151], U. of Maryland) and Ranger (SUNY Stony Brook), which are designed for efficient nearest-neighbor generation in $\mathbb{R}^d$. These techniques, however, are developed exclusively for Euclidean spaces and cannot be applied directly to path planning algorithms because of the topologies of configuration spaces. The topologies that we consider in this chapter are $\mathbb{R}$, $S^1$, and $\mathbb{RP}^3$, for which metric information (Chapter 2.2) must be appropriately processed by any data structure that performs correct nearest-neighbor computations. Several other nearest-neighbor packages exist, such as sb(S) [47], and cover trees [25], that answer nearest-neighbor queries in general metric spaces. These packages use the metric function provided by the user as a “black box” for building a data structure based only on metric evaluations between the data points. Since any valid metric can be provided as the input, these methods are very general and usually introduce high computational overhead for Euclidean spaces and simple topological spaces that arise in motion planning.

Kd-trees [13, 72, 187] are well known for their good performance on Euclidean data sets. They usually outperform other approaches in practice, except in rare pathological cases. In this chapter, we show how the kd-tree-based nearest-neighbor algorithm and part of the ANN package of Arya and Mount [151] can be extended to handle topologies arising in motion planning. The resulting method retains the performance benefits of kd-trees by introducing very little computational overhead for handling the appropriate constraints induced by the metric and topology of the configuration space. First, we formulate the problem in Section 3.2. The appropriate metric spaces were defined in Chapter 2.2. A literature overview of existing techniques for nearest-neighbor searching is covered in Section 3.3. We then present our algorithm and prove the correctness of the approach in Section 3.4. We demonstrate the efficiency of the algorithm empirically in Section 3.5. Our experiments show the performance improvement of the proposed algorithm over using linear-time naive nearest-neighbor computations, the sb(S) library, and the cover-tree library. The speedup is a few orders of magnitude in some cases. We also present experiments that show substantial performance improvement in the PRM and RRT methods applied to difficult path planning examples. We have implemented the proposed method as a publicly available software package [211].
3.2 Problem Formulation

Consider one of the metric spaces described in Section 2.2, \( T = T_1 \times \cdots \times T_m \), in which each \( T_i \) is one of \( \mathbb{R}, \mathbb{S}^1 \) or \( \mathbb{R}^3 \). Consider the weighted metric defined on this manifold, \( \rho_T : T \times T \to \mathbb{R} \). Suppose that a set of \( n \) data points, \( S \), is a subset of \( T \). The problem is: given any query point \( q \in T \), efficiently report the point \( p \in S \) that is closest to \( q \).

Note that the brute-force computations of all the distances is one way of finding a correct nearest neighbor. However, our goal is to achieve significantly faster running times. We allow some preprocessing time for organizing the data points in a data structure. In return, we expect that the answer to the nearest-neighbor query will be found significantly faster than the brute-force computations.

3.3 Nearest-Neighbor Searching Overview

There has been a significant interest in nearest-neighbor and related problems over the last couple of decades. For Euclidean data sets, kd-tree-based methods proved to be one of the most effective in practice. The kd-tree data structure is based on recursively subdividing the rectangle enclosing the data points into subrectangles using alternating axis-aligned hyperplanes. Given the appropriate distance measure between points and rectangles in the space, kd-trees allow one to eliminate some of the points in the data set from the search during the query phase. Given a query point, \( q \), it may be possible to discard some of the points in the data set based only on the distance between their enclosing rectangle and the query point. That is, based on one metric computation, the whole set of points inside the rectangle is eliminated from the search. The classical kd-tree uses \( O(dn \lg n) \) precomputation time, and answers orthogonal range queries in time \( O(n^{1-1/d}) \). One of the first appearances of the kd-tree is in [72], and a more modern introduction appears in [55]. Improvements to the data structure and its construction algorithm in the context of nearest-neighbor searching are described in [187]. In [13] it is shown that using kd-trees for finding approximate nearest neighbors allows significant improvement in running time with a very small loss in performance for higher dimensions. Other data structures for nearest-neighbor searching in Euclidean spaces are used for high-dimensional problems [91], and for dynamic data [2].

Different techniques have been developed for nearest-neighbor searching in general metric spaces [48, 86]. Many efficient algorithms [25, 46, 107] were implemented and tested on various data sets [25, 47]. Most of these techniques consider the metric as a “black box” function provided to the
algorithm. Usually these methods group the points in such a way that it is possible to eliminate some groups of points from the search in the query phase based on some inexpensive test. This approach is similar to kd-tree-based approach, in which the points are eliminated from the search if they are enclosed by a rectangle far enough from the query point. However, since these techniques are more general and allow any metric space to be searched, they are usually not as efficient on Euclidean spaces as techniques designed primarily for Euclidean spaces, such as kd-trees [47].

The goal of this chapter is to show how to adapt kd-trees to handle spaces described from Chapter 2.2, introducing only a little computational overhead for handling topological constraints and, therefore, keeping the simplicity and efficiency of kd-trees. The next section introduces our method.

3.4 An Approach Based on Kd-trees

First, we elaborate on possible ways of using kd-trees for given spaces, and then we present our approach.

3.4.1 A Naive Way to Use Kd-Trees

To apply kd-tree-style reasoning to the metric spaces of interest, a naive approach would be to embed a given manifold into a higher-dimensional Euclidean space, and then treat the set of points lying on this manifold as a Euclidean data set. For example, the set of all rotations can be represented using $3 \times 3$ matrices, which places them in Euclidean space $\mathbb{R}^9$. The drawback of this approach is that the dimensionality of the space is significantly increased, which often implies worse performance of nearest neighbor methods. Moreover, the Euclidean metric in the resulting Euclidean space is different from the natural metric defined over quaternions. For many applications this is not tolerable, and kd-trees cannot be immediately applied. Next we show how a different approach can be taken so that the kd-tree data structure is adapted naturally and efficiently to the metric spaces of interest.

3.4.2 Representing the Spaces of Interest

Consider the metric spaces of interest before the identifications are done. That is, the circle is considered as a unit interval in $\mathbb{R}^1$ and the quaternion real projective space as a 3D sphere embedded in $\mathbb{R}^4$. The kd-tree can be first constructed inside $\mathbb{R}^3$ and $\mathbb{R}^4$. Next, to obtain a correct answer to
the nearest-neighbor query, identifications and the correct metric are used in the query phase. That
is, when computing distances from the query point to a point or an enclosing rectangle of a set of
points, the correct metric respecting the topology of the space is used. In this manner, a rectangular
decomposition is done on these non-Euclidean spaces, and, at the same time, the correct metric is
used throughout the search.

In the rest of this subsection we define the notion of enclosing rectangle, and distance between
a point and a rectangle in each of the defined metric spaces.

**Euclidean one-space**

The enclosing rectangles are regular intervals in \( \mathbb{R}^1 \), and the distance between a point, \( p \), and a
rectangle, \([a, b]\), is the usual Hausdorff metric:

\[
\rho_{\mathbb{R}}(p, [a, b]) = \inf_{r \in [a, b]} \rho_{\mathbb{R}}(p, r).
\]  
(3.1)

**Circle \( S^1 \)**

The enclosing rectangle for a set of points on the circle is any subinterval of \([0, 1]\). The distance
between a point, \( p \), and a rectangle, \([a, b]\), is the Hausdorff distance on \( S^1 \):

\[
\rho_{S^1}(p, [a, b]) = \inf_{r \in [a, b]} \rho_{S^1}(p, r).
\]  
(3.2)

**Real projective space \( \mathbb{RP}^3 \)**

Rectangles that enclose the data lying on the unit sphere \( S^3 \subseteq \mathbb{R}^4 \) are usual rectangular regions
\([a_1, b_1] \times \cdots \times [a_4, b_4]\) in \( \mathbb{R}^4 \). The distance between a point, \( p \), and a rectangle, \( R \), could be defined as the
Hausdorff distance between \( p \) and the intersection of \( R \) with the sphere. However, the distance
that we use in this work is more efficient to compute and guarantees the correctness of the nearest-
neighbor search, as we prove in Section 3.4.7. Essentially, the following is the Hausdorff distance
between \( p \) and \( R \) in \( \mathbb{R}^4 \), respecting the identifications of \( \mathbb{RP}^3 \):

\[
\rho_{\mathbb{RP}^3}(p, R) = \min(\rho_{\mathbb{R}^4}(p, R), \rho_{\mathbb{R}^4}(-p, R)).
\]  
(3.3)
Consider the topological space $T$, such that $T$ is a Cartesian product $T = T_1 \times \cdots \times T_m$ of copies of $\mathbb{R}$, $S^1$, and $\mathbb{RP}^3$. Enclosing rectangles for this space are those formed by enclosing rectangles in the projections of $T$ on each of $\mathbb{R}$, $S^1$, and $\mathbb{RP}^3$. The distance between a point and a rectangle is defined as

$$\rho_T(p, R) = \sqrt{\sum_i \mu_{T_i} \rho_{T_i}^2(p, R)},$$  \hspace{1cm} (3.4)$$

in which each $\mu_{T_i}$ is the constant defined in (2.25).

### 3.4.3 Kd-trees for the Spaces of Interest

The kd-tree-based approach for the nearest-neighbor problem formulated in Section 3.2 consists of first precomputing the data structure for storing points, and then searching this data structure when a query is given. In this subsection, we describe the kd-tree data structure for the manifolds of interest in more detail, and in the following subsections we provide the algorithms for the construction and query phases.

Consider the set of data points, $S$, lying inside a $d$-dimensional enclosing rectangle as described above. We build the kd-tree data structure inside this rectangle, and define it recursively as follows. The set of data points is split into two parts by splitting the rectangle that contains them into two child rectangles by a hyperplane, according to some specified splitting rule; one subset contains the points in one child box, and another subset contains the rest of the points. The information about the splitting hyperplane and the boundary values of the initial box are stored in the root node, and the two subsets are stored recursively in the two subtrees. When the number of the data points contained in some box falls below a given threshold, the node associated with this box is called a leaf node, and a list of coordinates for these data points is stored in this node.

We use splitting rules suggested in [151], which divide the current cell through its midpoint orthogonal to its longest side. If there are ties, it selects the dimension with the largest point spread. However, in the case in which points are all on one side of the splitting plane, the algorithm slides the plane toward the first encountered data point. According to [151] these rules perform very well with typical data sets in $\mathbb{R}^d$.

Figure 3.1 illustrates how the splitting is done, and how the corresponding binary tree looks for the data points on a torus.
3.4.4 Construction Phase

Our kd-tree is constructed using a recursive procedure, which returns the root of the kd-tree (see Figure 3.2). This construction algorithm is essentially identical to the case of constructing a kd-tree in a Euclidean space [151]. The identifications and proper metrics are not used in construction phase, and the points are treated as lying inside some $\mathbb{R}^d$ as described in the beginning of this section.

3.4.5 Query Phase

The query phase must be handled differently in comparison to a standard kd-tree, by incorporating the correct metrics defined in Sections 2.2 and 3.4 when traversing the tree. In everything else, the search proceeds in the same manner as the search in classical kd-trees. At first, the query algorithm descends to a leaf node that contains the query point, finds all distances from the data points in this leaf to the query point, and picks up the closest one. It then recursively visits only those surrounding rectangles that are closer to the query point than the closest point found so far (with respect to the correct metric). Those that are further away are discarded from consideration. Figure 3.3 describes the query algorithm.

We borrowed some efficient techniques from [12] to further speed up the computations. Using squared distances prevents calculating costly square roots. We have also modified the method of incremental distance calculation for speeding up the calculations of a distance between the query

![Figure 3.1: A kd-tree: a) how a torus is subdivided, b) the corresponding binary tree.](image-url)
BUILD_KD_TREE(P, d, T, m, b, s)

Input: A set of points, P, the dimension of the space, d, the topology of the space, T, the number of points to store in a leaf, m, the bounding box, b, for P, and the splitting rule, s.

Output: The root of a kd-tree storing P

1 if P contains less than m points
2 then return a Leaf storing these points
3 else split b into two subboxes, b₁, b₂, according to s by plane l, orthogonal to dimension k.
4 Find P₁ and P₂, the sets of the data points falling into boxes b₁ and b₂.
5 v₁ = BUILD_KD_TREE(P₁, d, T, m, b₁, s)
6 v₂ = BUILD_KD_TREE(P₂, d, T, m, b₂, s)
7 Create a Node v storing the splitting plane, l, the splitting dimension, k, the topology of the space T_K of this dimension, the projection of the box, b, on T_K, and v₁ and v₂, the children of v.
8 return v

Figure 3.2: The algorithm for constructing a kd-tree in a topological space T.

point and a rectangle. This method can be described as follows. Let T be a Cartesian product of several manifolds, T = T₁ × ⋯ × Tₘ, and let coordinate axis k correspond to some space Tᵦ. Suppose that a query point, q, and an enclosing rectangle for the data set, S, in T are given. Divide R with a plane orthogonal to coordinate axis k into two child rectangles R₁ and R₂. If it is known that ρ²(q, R) = d_box, then the squared distance from one of the rectangles (without loss of generality it can be R₁) to q is also d_box. To calculate ρ²(q, R₂), note that R₂ has the same projections as R on every Tᵢ except for Tᵦ, by definition of weighted metric in T (2.25). Therefore, if ρ²ₜₖ(q, R₁) = ρ²ₜₖ(q, R), then

\[ ρ²ₜₖ(q, R₂) = d_{box} - μₜₖ ρ²ₜₖ(q, R₁) + μₜₖ ρ²ₜₖ(q, R₂). \]

Therefore, calculating distance from a point to a rectangle node in d-dimensional space, T, takes O(d) time for the root node only, and for any other node the time is proportional to the time for calculating distance from a point to a rectangle in Tₖ, the subspace of T.

3.4.6 Making Kd-trees Dynamic

In some algorithms, such as RRTs, the number of points grows incrementally while nearest-neighbor queries are performed at each iteration. In this case, it is inefficient to rebuild the kd-tree at every iteration. One approach to make the nearest-neighbor algorithm dynamic is to use the point insertion
KDT::SEARCH(q)
Output: the closest to q point p stored in kd-tree.
1 Calculate squared distance $d_{box}$ from the box associated with the root node to q.
2 $p = \text{NULL}$
3 root $\rightarrow$ SEARCH($d_{box}$, $\infty$, p)
4 return p

Node::SEARCH($d_{box}$, $d_{best}$, p)
Input: squared distance, $d_{box}$, from q to the box containing the current Node, and squared distance, $d_{best}$, from q to the closest point, p, seen so far; $d_{best}$ and p are to be updated.
1 if $d_{box} < d_{best}$
2 Split $b_K$ (the projection of the current Node onto the space $T_K$, stored in this Node) into two subboxes, $b_{K1}$ and $b_{K2}$, by the splitting line $l$, corresponding to $v_1$ and $v_2$ respectively.
3 $d_1 = \rho_T^2(q, b_{K1})$
4 $d_2 = \rho_T^2(q, b_{K2})$
5 if $d_1 < d_2$
6 then $v_1 \rightarrow$ SEARCH($d_{box}$, $d_{best}$, p)
7 $v_2 \rightarrow$ SEARCH($d_{box} - d_1 + d_2$, $d_{best}$, p)
8 else $v_2 \rightarrow$ SEARCH($d_{box}$, $d_{best}$, p)
9 $v_1 \rightarrow$ SEARCH($d_{box} - d_2 + d_1$, $d_{best}$, p)

Leaf::SEARCH($d_{box}$, $d_{best}$, p)
Input: squared distance, $d_{box}$, from q to the box containing the current Leaf, and squared distance, $d_{best}$, from q to the closest point, p, seen so far; $d_{best}$ and p are to be updated.
1 Calculate squared distances from q to all the points in the current Leaf, and update p and $d_{best}$.

Figure 3.3: The algorithm portions for searching a kd-tree on the root level and internal and leaf nodes levels.

operation with tree rebalancing [155]. It is costly, however, to ensure that the trees are balanced.

Another approach, which we used in our implementation, is a standard method to perform static-to-dynamic transformation of a data structure, called the logarithmic method [24]. For n points, there is a tree that contains $2^i$ points for each “1” in the $i^{th}$ place of the binary representation of n. As bits are cleared in the representation due to increasing n, the trees are deleted, and the points are included in a tree that corresponds to the higher-order bit which changed to “1”. This general scheme incurs logarithmic-time overhead, regardless of dimension. It is also straightforward to implement, and leads to satisfactory experimental performance.

3.4.7 Analysis

Proposition 1 The algorithm presented in Figure 3.3 correctly returns the nearest neighbor.

Proof: We argue that the points in the kd-tree that are not visited by our algorithm cannot be
the closest neighbors to the query point, since they are always further from the query point than
some point that was already visited by the algorithm. At first, the search procedure descends to
the leaf node to which the query point belongs. Therefore, the closest point to the query point from
this leaf will be the first candidate to be the nearest neighbor. After searching this leaf node the
algorithm skips only those nodes (enclosing rectangles) that are further from the query point than
the candidate to the nearest neighbor seen so far. Any point inside a node that was skipped cannot
be the nearest neighbor, since the point inside a rectangle is further from the query point than the
rectangle itself. This holds true for Hausdorff distances defined on \( R^3 \) and \( S^1 \) by definition. It is
also true for the distance we used on \( \mathbb{R}P^3 \), since

\[
\rho_{\mathbb{R}P^3}(q, R) \leq \rho_{R^4}(q, p),
\]

for all \( p \in R \), by definition of \( \rho_{\mathbb{R}P^3}(q, R) \). Since the length of the arc along the sphere is longer than
the length of the corresponding chord, we obtain

\[
\rho_{\mathbb{R}P^3}(q, R) \leq \rho_{R^4}(q, p) \leq \rho_{\mathbb{R}P^3}(q, p),
\]

for all \( p \in R \).

**Proposition 2** For \( n \) points in dimension \( d \), the construction time is \( O(dn \lg n) \), the space is \( O(dn) \),
and the query time is logarithmic in \( n \), but exponential in \( d \).

**Proof:** This follows directly from the well-known complexity of the basic kd-tree [72]. Our approach
performs correct handling of the topology without any additional asymptotic complexity.

The metric evaluations are more costly due to identifications in the manifold definition; however,
this results only in a larger constant in the asymptotic analysis. For example, each \( S^1 \) subspace
requires two more operations per distance computation (see definitions in (2.13) and (3.2)), which
essentially does not affect the overall running time. For each \( \mathbb{R}P^3 \) there are sometimes 3 to 6
additional operations per distance computation (see (2.17) and (3.3)). Two of these operations
are \( \cos^{-1} \), which are expensive and sometimes take several orders of magnitude longer than basic
addition or multiplication operations. This results in higher constants in the asymptotic running
time for spaces containing \( \mathbb{R}P^3 \).

By following several performance enhancements recommended in [151], the effects of high dimen-
sionality on the query time are minimized, yielding good performance for nearest-neighbor searching
Table $\mathbb{R}^d$

<table>
<thead>
<tr>
<th>d</th>
<th>MPNN</th>
<th>naive</th>
<th>cover tree</th>
<th>sb(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.2 + 0.01</td>
<td>0.22</td>
<td>0.63 + 0.01</td>
<td>2.43 + 0.02</td>
</tr>
<tr>
<td>6</td>
<td>0.34 + 0.01</td>
<td>0.38</td>
<td>0.79 + 0.1</td>
<td>14.0 + 0.08</td>
</tr>
<tr>
<td>9</td>
<td>0.48 + 0.02</td>
<td>0.56</td>
<td>1.36 + 0.19</td>
<td>41.7 + 0.46</td>
</tr>
<tr>
<td>12</td>
<td>0.64 + 0.14</td>
<td>0.74</td>
<td>3.72 + 0.77</td>
<td>90.6 + 0.91</td>
</tr>
<tr>
<td>15</td>
<td>0.77 + 0.47</td>
<td>0.94</td>
<td>9.52 + 1.30</td>
<td>158.3 + 1.76</td>
</tr>
<tr>
<td>18</td>
<td>0.97 + 1.61</td>
<td>1.14</td>
<td>23.0 + 2.16</td>
<td>238.3 + 3.25</td>
</tr>
<tr>
<td>21</td>
<td>1.16 + 2.17</td>
<td>1.55</td>
<td>51.5 + 2.36</td>
<td>273.4 + 3.19</td>
</tr>
<tr>
<td>24</td>
<td>1.40 + 3.20</td>
<td>1.65</td>
<td>84.4 + 3.31</td>
<td>327.6 + 5.71</td>
</tr>
<tr>
<td>27</td>
<td>1.58 + 3.70</td>
<td>1.78</td>
<td>135.4 + 3.59</td>
<td>373.3 + 6.59</td>
</tr>
<tr>
<td>30</td>
<td>1.80 + 4.11</td>
<td>2.04</td>
<td>242.0 + 4.33</td>
<td>392.8 + 7.59</td>
</tr>
</tbody>
</table>

Table $(S^1)^d$

<table>
<thead>
<tr>
<th>d</th>
<th>MPNN</th>
<th>naive</th>
<th>cover tree</th>
<th>sb(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.21 + 0.01</td>
<td>0.26</td>
<td>0.69 + 0.02</td>
<td>2.97 + 0.01</td>
</tr>
<tr>
<td>6</td>
<td>0.32 + 0.01</td>
<td>0.44</td>
<td>1.06 + 0.04</td>
<td>13.0 + 0.07</td>
</tr>
<tr>
<td>9</td>
<td>0.48 + 0.01</td>
<td>0.69</td>
<td>2.34 + 0.15</td>
<td>39.8 + 0.46</td>
</tr>
<tr>
<td>12</td>
<td>0.63 + 0.05</td>
<td>0.85</td>
<td>7.31 + 0.87</td>
<td>88.5 + 1.06</td>
</tr>
<tr>
<td>15</td>
<td>0.77 + 0.38</td>
<td>1.04</td>
<td>17.8 + 2.11</td>
<td>156.9 + 1.77</td>
</tr>
<tr>
<td>18</td>
<td>0.98 + 0.89</td>
<td>1.21</td>
<td>45.4 + 2.89</td>
<td>239.7 + 2.9</td>
</tr>
<tr>
<td>21</td>
<td>1.20 + 1.83</td>
<td>1.37</td>
<td>104.8 + 3.73</td>
<td>321.7 + 3.26</td>
</tr>
<tr>
<td>24</td>
<td>1.40 + 2.68</td>
<td>1.58</td>
<td>186.9 + 6.21</td>
<td>411.0 + 8.83</td>
</tr>
<tr>
<td>27</td>
<td>1.62 + 3.30</td>
<td>1.72</td>
<td>289.7 + 5.62</td>
<td>478.5 + 9.07</td>
</tr>
<tr>
<td>30</td>
<td>1.80 + 4.03</td>
<td>1.89</td>
<td>545.6 + 8.43</td>
<td>499.2 + 9.25</td>
</tr>
</tbody>
</table>

Figure 3.4: Nearest-neighbor computations are shown for 50000 data points generated at random in Euclidean spaces $\mathbb{R}^d$. The time to perform 100 queries is shown for the naive, brute-force algorithm. For other methods the construction time is added to the time required to perform 100 queries.

Figure 3.5: Nearest-neighbor computations are shown for 50000 data points generated at random in spaces $(S^1)^d$. The time to perform 100 queries is shown for the naive, brute-force algorithm. For other methods the construction time is added to the time required to perform 100 queries.

in up to several dozen dimensions in both ANN and our algorithm. Performance can be further improved by returning approximate nearest neighbors, if suitable for a particular motion planning method.

### 3.5 Experimental Results

We have implemented our nearest-neighbor algorithm in C++ as part of the new library, MPNN, for nearest-neighbor searching on dynamic data sets in the context of motion planning. This library is publicly available [211]. The kd-tree implementation that we used in MPNN is borrowed from the ANN library. We then used MPNN in implementations of RRT-based and PRM-based planners in the Motion Strategy Library [117]. The experiments reported here were performed on a 2.2 GHz
Figure 3.6: Nearest-neighbor computations are shown for 50000 data points generated at random in spaces \((\mathbb{R}^3 \times \mathbb{R}P^3)^k\). The time to perform 100 queries is shown for the naive, brute-force algorithm. For other methods the construction time is added to the time required to perform 100 queries.

<table>
<thead>
<tr>
<th>d</th>
<th>k</th>
<th>MPNN</th>
<th>naive</th>
<th>cover tree</th>
<th>sb(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1</td>
<td>0.29 + 0.02</td>
<td>3.17</td>
<td>1.1 + 0.06</td>
<td>19.5 + 0.1</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>0.47 + 0.12</td>
<td>6.18</td>
<td>5.6 + 0.4</td>
<td>106 + 0.9</td>
</tr>
<tr>
<td>21</td>
<td>3</td>
<td>0.75 + 1.10</td>
<td>9.20</td>
<td>25.8 + 2.8</td>
<td>231 + 1.7</td>
</tr>
<tr>
<td>28</td>
<td>4</td>
<td>0.89 + 1.82</td>
<td>12.2</td>
<td>89.5 + 4.3</td>
<td>457 + 5.9</td>
</tr>
<tr>
<td>35</td>
<td>5</td>
<td>0.97 + 3.72</td>
<td>15.2</td>
<td>215 + 6.8</td>
<td>723 + 10.1</td>
</tr>
<tr>
<td>42</td>
<td>6</td>
<td>1.17 + 6.2</td>
<td>18.2</td>
<td>469 + 9.2</td>
<td>981 + 18.5</td>
</tr>
<tr>
<td>49</td>
<td>7</td>
<td>1.43 + 9.32</td>
<td>20.9</td>
<td>658 + 12.0</td>
<td>1205 + 21</td>
</tr>
<tr>
<td>56</td>
<td>8</td>
<td>1.63 + 11.2</td>
<td>24.0</td>
<td>1435 + 13.4</td>
<td>1374 + 27</td>
</tr>
</tbody>
</table>

Pentium IV running Linux and compiled under GNU C++.

In motion planning the points are usually added to the current data structure one by one, and the nearest neighbor query is performed at each iteration. When the number of points is small, each iteration is computationally inexpensive and all of the nearest neighbor methods perform well. However, many motion planning problems need thousands of nodes in order to find a solution. In this case, the performance of the nearest neighbor method becomes crucial at each iteration. Moreover, because the number of nodes changes dynamically, once in a while, reconstruction of the kd-tree for the set of all of the points is performed, as it was described in Section 3.4.6. Given this motivation, we performed our first set of experiments shown in Figures 3.4-3.6 for 50,000 points. We measured the construction times of different methods as well as the query times. Since the performance of the algorithm sometimes drastically depends on the choice of the query point, we average the query time results over 100 queries.

We have compared the performance of the MPNN library to the brute-force algorithm as well as two general metric space nearest-neighbor libraries, cover trees [25] and sb(S) [47]. Figures 3.4-3.6 indicate the performance of these methods in various topological spaces. The performance improvement of the kd-tree-based approach over other methods is several orders of magnitude in some cases. As the dimension of the space increases, though, the brute-force algorithm outperforms all the methods, as well as kd-trees, because of the hidden exponential dependencies on the dimension in these methods. However, the data sets in motion planning often have small intrinsic dimensionality. Obstacles and other constraints, such as kinematic or differential constraints, reduce the effective dimension of the problem. Our experiments (see Figure 3.6) also suggest that kd-trees outperform other methods in up to 56 dimensions on randomly generated sets in \((\mathbb{R}^3 \times \mathbb{R}P^3)^k\) due...
to the choice of the constants in weighted metric. There is no general method for choosing the constants, however, rotations are usually given smaller weight than translations in motion planning problems. For example, $\mu_{R^3} = 1, \mu_{RP^3} = 0.15$ are the standard values used in MSL. This works to the advantage of kd-tree-based approaches, and, therefore, makes them potentially applicable to many motion planning problems with high-dimensional configuration spaces.

Figures 3.7 and 3.8 show performance of the methods in bidirectional RRT-based planners for 3-dof and 48-dof problems, respectively. Performance for a basic PRM applied to a 6-dof example is shown in Figure 3.9. These experiments suggest that the MPNN library can be effectively used in up to 56-dimensional spaces with considerable running time improvements in the performance of RRT-based and PRM-based planning algorithms. It is important to note, however, that nearest-neighbor searching does not represent the only bottleneck in motion planning. Sampling strategies and collision detection issues are also critical. For the experiments, we focused on examples that lead to a large number of nodes so that the nearest-neighbor searching would dominate. In general, the development of the most efficient algorithms should involve consideration of all of these issues.
Figure 3.8: This 56-dimensional problem involves exchanging positions of 8 L-shaped objects contained in a rectangular box. It was solved using RRTConCon [122].

<table>
<thead>
<tr>
<th>$(\mathbb{R}^3 \times \mathbb{R}P^3)^8$</th>
<th>MPNN</th>
<th>naive</th>
<th>cover tree</th>
<th>sb(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodes</td>
<td>18,920</td>
<td>18,485</td>
<td>20,907</td>
<td>22,210</td>
</tr>
<tr>
<td>time (sec)</td>
<td>2,055.57</td>
<td>4,152.19</td>
<td>5,273.14</td>
<td>6,161.47</td>
</tr>
</tbody>
</table>

Figure 3.9: This example is solved using the PRM approach [101]. The goal is to move the 3D rigid object out of the cage.

<table>
<thead>
<tr>
<th>$\mathbb{R}^3 \times \mathbb{R}P^3$</th>
<th>MPNN</th>
<th>naive</th>
<th>cover tree</th>
<th>sb(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodes</td>
<td>37,634</td>
<td>37,186</td>
<td>35,814</td>
<td>37,922</td>
</tr>
<tr>
<td>time (sec)</td>
<td>191.96</td>
<td>2,302.49</td>
<td>187.99</td>
<td>361.43</td>
</tr>
</tbody>
</table>
Chapter 4

Deterministic Sampling Methods for Configuration Spaces

The configuration space that arises in motion planning (Chapter 2.2) is usually uncountably infinite. As we have shown in Chapter 2.3, the way many sampling-based motion planning algorithms search through the configuration space is by considering a discrete set of samples. If the algorithm was to run forever, the number of samples it considered would be at most countable. This defines a mismatch between the cardinality of the configuration space, and the set of samples. The performance of the planning algorithm, therefore, crucially depends on the performance of the sampling method.

Many other important algorithms developed in robotics and related areas also require careful sampling over spheres. General sampling over spheres arises in many forms of planning and optimization in which some number of directions are locally explored. For example, some potential field approaches [19, 102] involve sampling local directions to obtain an approximate gradient descent. The exact expression of the gradient may be too costly or even unavailable. One important special case of sampling over spheres is sampling over the 3D rotation group, $SO(3)$, which involves sampling over half of the three-sphere, $S^3$. One of the main motivations for this chapter is the problem of motion planning for a rigid body in $\mathbb{R}^3$.

We are particularly interested in the development of deterministic sampling methods. Although most existing motion planning methods currently use random sampling, they are limited to probabilistic forms of completeness. With deterministic sampling, resolution completeness guarantees are possible. This is particularly valuable in the area of system verification, in which one must guarantee that a system behaves correctly under all possible trajectories. The intractability of most of these problems leads naturally to sampling-based approaches. While it may be valuable to verify a system down to some level of resolution, random sampling might leave doubts about whether the space was adequately covered. In some cases, deterministic sampling has even led to practical performance improvements in comparison to random sampling [131, 142, 152]. The techniques presented in this chapter build on recent work to develop uniform, deterministic sampling techniques for motion planning [31, 120, 131].
The particular problem of sampling over spheres presents many unique challenges. The vast majority of sampling literature considers placing points in a unit \(d\)-dimensional cube, \([0,1]^d \subset \mathbb{R}^d\) (see [120, 152]). This might correctly capture some configuration spaces that arise in robotics; however, the majority of applications involve other topological spaces, such as \(\mathbb{RP}^3\), which arises from rigid body rotations, or toroidal manifolds, which arise from a series of revolute joints of a manipulator. In these cases, special sampling techniques should be developed because quality measures for sets of samples depend on the topology. For example, the maximum distance that a configuration could be from its nearest sample depends on the metric, which is induced partly by the topology.

In addition to topological issues, the way that a configuration space is parametrized is of critical importance to defining notions of uniformity. A collection of samples that are uniform with respect to one parametrization of the configuration space might seem extremely biased using another parametrization. It might seem that there is no way to avoid this frustrating issue, but fortunately for the case of \(SO(3)\), there is an intrinsic notion of uniformity that is given by the Haar measure [67] (this was defined in Chapter 2.2). Using this notion, the natural parametrization of \(SO(3)\) is the set of unit quaternions (with antipodal identification), and our sampling methods will be developed to achieve rigorous notions of uniformity in this case.

To maximize the potential for impact on motion planning and related areas, our goal has been to develop a sampling method that achieves 1) uniformity, 2) lattice structure, and 3) incremental quality. Uniformity means good covering of the space is obtained without unwanted bias, clumping or gaps. This can be formulated in terms of optimizing discrepancy or dispersion [142, 152, 26]. The uniformity notion considered here is actually more “uniform” than what is obtained by random sampling. Lattice structure means that for every sample, the location of nearby samples can easily be determined as part of a regular pattern (as in neighbors on a grid, for example). Incremental quality means that if the sampling method is considered as an infinite sequence, then the sequence may be truncated after any finite number of samples and good coverage will be obtained. This is an important characteristic of pseudo-random number sequences, making them desirable for many past motion planning algorithms [5, 27, 101, 181, 215]. We would like to obtain the same behavior, even though the sequence is deterministic, uniform, and has lattice structure.

We start this chapter by overviewing basic terminology of sampling theory in Section 4.1. We also formulate the sampling problem arising in the context of the incremental sampling and searching framework from the Chapter 2.3. Remember, that the configuration spaces that we consider are
defined in Chapter 2.2. In this chapter we make emphasis on sampling spheres and $SO(3)$, and overview available methods for sampling other configuration spaces, which has been the subject of active research in sampling literature. We overview the background literature in Section 4.2. Sections 4.3-4.4 present our technical approach for sampling configuration spaces arising in motion planning and experimental results.

4.1 Sampling Terminology and Problem Formulation

Since the sampling-based planning algorithms are often terminated early, the particular order in which samples are chosen becomes crucial. Sampling literature distinguishes between a sample *set* and a sample *sequence*. For a sample set, the number of points, $n$, is specified in advance, and a set of $n$ points is then chosen to satisfy the requirements of the method. The notion of ordering between points is not defined for a sample set but becomes important for sequences. Successive points in a sequence should be chosen carefully so that the resulting sample sets are all of good quality. Sequences are particularly suitable for motion planning algorithms, in which the number of points needed to solve the problem is not known in advance.

Now that the background definitions for the configuration space $C$ have been presented in Chapter 2.2, to generate samples over $C$ we need to formulate the desirable properties for the samples. The first requirement is that samples form a sequence. We also require that samples get arbitrarily close to every point in $C$, i.e. that the sequence of samples is *dense* in $C$. Next we formulate several requirements on the uniformity properties of the samples.

4.1.1 Discrepancy and Dispersion

Additional requirements that the sequence needs to satisfy are described by the uniformity measures, *discrepancy* and *dispersion*.

Intuitively, discrepancy can be thought of as enforcing two criteria: first, that no region of the space is left uncovered; and second, that no region is left too full. Dispersion eliminates the second criterion, requiring only the first. It can be shown that low discrepancy implies low dispersion [152].

To define discrepancy formally, choose a range space, $\mathcal{R}$, as a collection of subsets of $C$. Let $R \in \mathcal{R}$ denote one such subset. Reasonable choices for $\mathcal{R}$ in Euclidean spaces include the set of all axis-aligned rectangles, the set of all balls, or the set of all convex subsets. The range spaces that are typically considered on spheres are the set of spherical caps (intersections of the 3-sphere with
Figure 4.1: An illustration of the notions of dispersion and discrepancy for a set of points on a 2-sphere. (a) The discrepancy searches for the subset \( R \) for which the deviation from the measure of \( R \) to the number of samples placed inside \( R \) is the largest. (b) The dispersion searches for a point \( q \) on the sphere which is the farthest from the sample points.

Let \( \mu(R) \) denote the measure (see Section 2.2.2) of the subset \( R \). If the samples in the set \( P \) are uniform in some ideal sense, then it seems reasonable that the fraction of these samples that lie in any subset \( R \) should be roughly \( \mu(R) \) divided by \( \mu(\mathcal{C}) \) (the measure of the whole configuration space). We define the discrepancy [206] to measure how far from ideal the sample set \( P \) is:

\[
D(P, R) = \sup_{R \in \mathcal{R}} \left| \frac{|P \cap R|}{N} - \frac{\mu(R)}{\mu(\mathcal{C})} \right| \tag{4.1}
\]

in which \(| \cdot |\) applied to a finite set denotes its cardinality. Figure 4.1 (a) demonstrates the notion on the 2-sphere.

While discrepancy is based on measure, a metric-based criterion, dispersion, can be introduced:

\[
\delta(P, \rho) = \max_{q \in \mathcal{C}} \min_{p \in P} \rho(q, p). \tag{4.2}
\]

Above \( \rho \) denotes any metric on \( \mathcal{C} \) that agrees with the measure, such as any of the metrics defined in Chapter 2.2. Intuitively, this corresponds to the spherical radius of the largest empty ball (assuming all ball centers lie in \( \mathcal{C} \)). See Figure 4.1(b) for an illustration.

### 4.1.2 Problem Formulation

In summary, the goal for the rest of this chapter is to define sequences of elements from \( \mathcal{C} \) which are:
Figure 4.2: The comparison of different sampling methods related to the problem of Section 4.1.2. The rows correspond to the desired properties of these methods. The columns represent different methods.

- incremental,
- deterministic,
- minimize the dispersion (4.2) and discrepancy (4.1) on $C$,
- has a grid structure with respect to a metric defined in Chapter 2.2 on $C$.

### 4.2 Sampling Methods Overview

Our work was influenced by many successful sampling methods developed recently for spheres and $SO(3)$. As demonstrated in the table of Figure 4.2, several of them are highly related to the problem formulation in Section 4.1.2. However, none of the methods known to date has all of the desired properties.

#### 4.2.1 Low-Discrepancy Sequences

Motivated by integration and optimization problems, sampling issues have been studied extensively in the applied mathematics community. Sample sets and sequences were developed to replace the random sequences traditionally used for these applications; they received the name quasi-Monte Carlo to denote this connection. Due to the fundamental importance of numerical integration, and the intricate link between discrepancy and integration error, most of the quasi-Monte Carlo literature focuses on the discrepancy measure.

Low-discrepancy sampling methods can be divided into three categories: Halton/Hammersley sampling, lattices, and $(t, s)$-sequences and $(t, m, s)$-nets. The first category represents one of the earliest methods, based on the original ideas of van der Corput [200]. The Halton sequence is a $d$-dimensional generalization that uses van der Corput sequences of $d$ different bases, one for each coordinate [78]. Figure 4.3 (a) shows the first 196 Halton points in $\mathbb{R}^2$. 

Figure 4.3: Different sequences of samples inside unit cubes. The Halton (a) and Hammersley (b) points achieve better discrepancy and dispersion than random (c) samples. Sukharev grid (d) and latices (e) provide optimal dispersion.

First, choose $d$ relatively prime integers $p_1, p_2, \ldots, p_d$ (usually the first $d$ primes, $p_1 = 2, p_2 = 3, \ldots$). To construct the $i^{th}$ sample, consider the digits of the base $p$ representation for $i$ in the reverse order (that is, write $i = a_0 + p a_1 + p^2 a_2 + p^3 a_3 + \ldots$, where each $a_j \in \{0, 1, \ldots, p\}$) and define the following element of $[0, 1]$:

$$r_p(i) = \frac{a_0}{p} + \frac{a_1}{p^2} + \frac{a_2}{p^3} + \frac{a_3}{p^4} + \cdots.$$

The $i^{th}$ sample in the Halton sequence is

$$(r_{p_1}(i), r_{p_2}(i), \ldots, r_{p_d}(i)), \quad i = 0, 1, 2, \ldots.$$

The Hammersley point set is an adaptation of the Halton sequence [79]. For the $i^{th}$ sample of a Hammersley point set with $N$ elements, the first coordinate is $i/N$ and the last $d - 1$ coordinates are the same as the $i^{th}$ sample of a $(d - 1)$-dimensional Halton sequence. Using only $d - 1$ distinct
primes, the $i^{th}$ sample in a Hammersley point set with $N$ elements is

$$\left(\frac{i}{N}, r_{p_1}(i), \ldots, r_{p_{d-1}}(i)\right), \quad i = 0, 1, \ldots, N - 1.$$ 

Figure 4.3(b) shows the Hammersley set for $n = 2$ and $k = 196$.

The construction of Halton/Hammersley samples is simple and efficient, which has led to their widespread application. However, the constant in their asymptotic analysis increases superexponentially with dimension [152].

The second category is lattices, which can be considered as a generalization of grids that allows nonorthogonal axes [142, 184, 204]. As an example, consider Figure 4.3 (e), which shows 196 lattice points. Rank-1 lattices were introduced by Korobov [106]; a rank-1 lattice of $N$ points is the set \(\{ih/N : i = 0, \ldots, N - 1\}\), in which $h$ is a $d$-dimensional generating vector of integers (depending on $N$) and \(\{\cdot\}\) represents the fractional part of the real value (modulo-one arithmetic). While historically lattices have required the specification of $N$ in advance, making them examples of low-discrepancy point sets, there has been increasing interest in extensible lattices, which are infinite sequences [85].

The third category is $(t, s)$-sequences and $(t, m, s)$-nets [152]. The key idea for these techniques is to enforce zero discrepancy over a particular subset of axis-aligned rectangles known as canonical rectangles, and all remaining elements of the range space will contribute only small amounts to the overall discrepancy. The most famous and widely-used $(t, s)$-sequences are Sobol’ and Faure (see [152]). The Niederreiter-Xing $(t,s)$-sequence has the best-known asymptotic constant, \((a/d)^d\), among all low-discrepancy sequences; in the expression, $a$ is a small constant [153].

### 4.2.2 Layered Sukharev Grid Sequence

Optimizing dispersion results in the point distribution which is often better for motion planning purposes [119]. For a given number of samples, $k$, the optimal way to reduce dispersion in $[0,1]^d$ with respect to $l^\infty$ metric is obtained by partitioning $[0,1]^d$ into a grid of cubes and placing a point at the center of each cube, as shown for $d = 2$ and $k = 196$ in Figure 4.3 (d). As in case of optimal discrepancy sets, the problem of minimizing dispersion has been mostly studied for the case of unit cubes [152].

When designing sequences that optimize dispersion, it is useful to consider multiresolution grid sequences [131]. A multiresolution grid of resolution $l$ is a grid with $2^l$ points per axis and $2^{dl}$
From this definition it follows that a grid of resolution \( l \) contains all of the points from resolution \( l - 1 \). The natural way to make this grid incremental is to build it one resolution at a time. During construction of the points from the same resolution level, the recursive procedure at each step adds those points that maximally decrease the discrepancy of the sequence, which extends van der Corput’s one-dimensional sequence [200].

As an example, consider a square, \([0,1]^2\), with four grid points inside. The best order of placing these points is: \((0, 0), (0.5, 0.5), (0, 0.5), (0.5, 0)\). To add the next 12 points from resolution 3, what point should be placed first, second, and third out of this sequence? The idea is that every four points should follow the same ordering of quadrants as the first four points (i.e., the first point should fall into the left-bottom rectangle, the next into right-top, and so on). Where exactly the point should be placed within the left-bottom rectangle should be decided by the same criterion that was used to place the first 4 points. In this case the next point is \((0.25, 0.25)\).

The resulting sequence has several important properties: it is incremental, it has low dispersion at each resolution level, it has optimal discrepancy with respect to the set of canonical rectangles, it has lattice structure, and there are efficient methods for generating the sequence and performing nearest neighbor queries on it [131]. This makes multiresolution grid sequences particularly useful for motion planning applications.

For our developments later in the chapter we will borrow the idea of the layered version of this sequence. A layered Sukharev grid of resolution \( l \) is a point set containing all the points of Sukharev grids of resolutions 1, 2, 4, ... \( 2^l \). It follows that this grid has \( n = \sum_{i=0}^{l} (2^i)^d = (2^{d(l+1)} - 1)/(2^d - 1) \) points total.

A layered Sukharev grid sequence builds one Sukharev grid of resolution \( 2^i \) at a time, \( i = 1, 2, .... \). Points from each of these grids then are generated by the same procedure as for building multiresolution grid sequences.

In Section 4.3 we show how to generalize layered Sukharev grid sequence to the sphere \( S^d \). We first show how the points should be generated in each of the spherical cubes, and then how all these points can be combined into one sequence on the sphere.

### 4.2.3 Random Sequences of Elements from \( S^d \)

To generate uniformly distributed random points on a hypersphere \( S^d \), spherical symmetry of the multidimensional Gaussian density function can be exploited [65]. For each of the \( d + 1 \) coordinates use a zero-mean Gaussian distribution with the same variance for each of the coordinated to generate...
This is done approximately by generating \( k \) uniformly distributed values from the interval \([-1, 1]\) and adding them following the Central Limit Theorem. In practice, any \( k \geq 12 \) is a reasonable choice. Then the normalized vector \( (x_i/\|x_i\|) \) is uniformly distributed over the hypersphere \( S^d \).

### 4.2.4 HEALPix

The HEALPix package [76] was designed for efficient and incremental discretization of full-sky maps in application to the satellite missions to measure the cosmic microwave background in astrophysics. It provides a deterministic, uniform, and multiresolution sampling method for the 2-sphere. Moreover, it possesses additional qualities, such as equal area partitioning of the 2-sphere, and isolatitude sampling on the 2-sphere, which make computations of the spherical harmonics integrals even more efficient.

This method takes advantage of the property of the cylindrical coordinates of the 2-sphere, which is measure preserving. That is, equal area partition on the cylindrical projection results in the equal area partition on the surface of the sphere. The distribution of one of the coordinates of the cylindrical projection is uniform, if the samples are uniformly distributed on the 2-sphere.

These are intrinsic properties of the 2-sphere that cannot be generalized directly to higher dimensional spheres. However, this work shows that an extremely uniform grid can be constructed on such a non-trivial curvature space as the 2-sphere. It is also not difficult to make this grid incremental using the method from [134].

### 4.2.5 Random Sequence of \( SO(3) \) Rotations

There are several ways of sampling the space of rotations uniformly at random [10, 180, 189, 217]. The main difficulty in doing so is the choice of a convenient parametrization of \( SO(3) \). If a parameter space is sampled uniformly, the resulting samples on \( SO(3) \) are not necessarily uniform. As was shown in Chapter 2.2, not all of the parametrizations of \( SO(3) \) are natural representations of rotations, and some of them lead to measure distortions, and even singularities. Only few parametrizations, such as the Hopf coordinates, result in a local isometry to \( SO(3) \).

It is easy to make the mistake of sampling rotations using a wrong parametrization [9]. The subgroup algorithm [57] for selecting random elements for \( SO(3) \) is the correct and most popular method for uniform random sampling of \( SO(3) \). It uses the fact that any Lie group can be uniformly sampled, by combining elements from a subgroup (in case of \( SO(3) \) it is \( S^1 \)), and the quotient, or coset space (\( S^2 \)) at random. Essentially, this method utilizes the Hopf coordinates.
Random sequences of rotations are used in many applications, however, they lack deterministic uniformity guarantees, and the explicit neighborhood structure.

4.2.6 Successive Orthogonal Images on $SO(n)$

Related to the subgroup method for generating random rotations is the deterministic method of Successive Orthogonal Images [149], which generates lattice-like sets with a specified length step based on uniform deterministic samples from the subgroup, $S^1$, and the coset space, $S^2$. The method is also generalized to arbitrary $SO(n)$.

The deterministic point sets can be applied to the problems in which the number of the desired samples is specified in advance. If the sample on $S^2$ is chosen so that it has a grid structure, the resulting sample on $SO(3)$ has the explicit neighborhood structure. Part of our work will be in applying this method in a way that provides the incremental quality necessary for our motion planning applications.

Successive Orthogonal Images is the deterministic method which, similarly to the random sequences, utilizes Hopf coordinates. This method was our motivation for designing the uniform deterministic sequences on $SO(3)$.

4.3 Our Approach: Layered Sukharev Grid Sequences for Spheres and $SO(3)$

Our general approach to sampling is based on Platonic solids, which we describe next in Section 4.3.1. In Section 4.3.2 we describe a particular sequence we propose for sampling hyperspheres. This approach is based on inscribing a hypercube and applying Layered Sukharev Grid sequence on the barycentric coordinates induced by the hypercube vertices. In Section 4.3.3 we propose an extension of the Layered Sukharev Grid sequence to Cartesian products of spaces arising in motion planning. Section 4.3.4 returns to the issue of sampling the space of all rotations, $SO(3)$. By considering the intrinsic structure of $SO(3)$ captured by Hopf fibration, we are able to design a sequence with minimal distortions on the grid structure in this space.

4.3.1 Exploiting the Regularity of Platonic Solids

In $\mathbb{R}^3$, a Platonic solid, or regular polyhedron, is a polyhedron for which every face is a copy of a regular polygon, fixed over all faces, and the degree of every vertex is fixed. Let $(v, e, f)$ denote
the numbers of vertices, edges, and faces of a regular polyhedron. Although there is an infinite number of regular polygons, there are only five regular polyhedra: tetrahedron (4,6,4), cube (8,12,6), octahedron (6,12,8), icosahedron (12,30,20), and dodecahedron (20,30,12). The notion of a regular polyhedron can be generalized to higher dimensions to obtain a regular polytope. In $\mathbb{R}^4$, it turns out that there are six regular polytopes: simplex (5,10,10,5), cube (16,32,24,8), cross polytope (8,24,32,16), 24 cell (24,96,96,24), 120 cell (600,1200,720,120), 600 cell (120,720,1200,600). The fourth element in each sequence denotes the number of 3D cells (which are regular polyhedra). Finally, in $\mathbb{R}^d$ for any $d > 4$, there are only three regular polytopes: simplex, cube, and cross polytope.

We first address the problem of generating a uniformly distributed set of points over $S^d$. Consider inscribing any $(d + 1)$-dimensional regular polytope inside of $S^d$, so that all of its $n$ vertices lie in $S^d$. The set of vertices are beautifully arranged around $S^d$ so that the points are evenly spaced. Furthermore, the edges of the polytope yield a regular lattice structure that is natural for building roadmaps in planning problems. For the case of sampling $SO(3)$, we simply use a set of vertices that lie in one hemisphere (making sure that no antipodal pairs of points appear in the set). The edges can be obtained directly from the polytope by making the appropriate identification of antipodal pairs.

Unfortunately, there are only a few combinations of $n$ and $d$ for which these ideal samples may be constructed for $S^d$ and $SO(3)$. This might be suitable for some applications, such as picking a set of candidate directions from $S^d$ for gradient descent of a potential function; however, in general, we would like to have a nice distribution of points for any value of $n$.

To the best of our knowledge, it is impossible to perfectly space $n$ points around $S^d$, for any $n$ and for $d > 1$. One simple idea that increases the number of samples is to place one point in the center of each of the $c$ $d$-cells of some regular polytope, and lift it to $S^d$. If we take the union of these points with the set of $v$ polytope vertices, a nice point set of size $c + v$ may be obtained. If more points are placed; however, the problem becomes more complicated. Therefore, we are willing to tolerate some distortion in the distribution of points. It still seems useful, however, to borrow some of the properties of the regular polytopes to generate good samples. The general idea pursued in our approach is to sample uniformly on the surface of the regular polytope, and then transform generated distribution to the surface of the sphere. We next describe this general method and discuss the induced distortion.

Consider a $(d + 1)$-dimensional regular polytope inscribed in the sphere $S^d$. Suppose there exists
a good method of sampling the surface of this polytope. The faces (d-dimensional cells) of the polytope, if projected outward to the surface of the sphere, form a tiling of the surface with the d-dimensional spherical polytopes. Consider some particular face, $F$, and its corresponding spherical face, $F'$. Each point inside $F$ can be described by a barycentric coordinate system induced by vertices of $F$ after its triangulation.

Now imagine that a set of points is generated inside of $F$. Each of the points in this distribution can be obtained through several steps of linear interpolation between the vertices of the barycentric coordinate system. The set of points on $F'$ can then be obtained through similar steps of interpolating between the vertices of $F'$, except that the interpolation should be done on the surface of the sphere [179]. This idea is similar to the one proposed in [11] for stratified sampling of spherical triangles. As an example, consider a cube inscribed in the sphere $S^2$, and sample the surface of the cube by putting the Sukharev grid [120, 192] on each square face. Using the proposed method we get a distribution of samples on $S^2$ as shown on Figure 4.4.

The set of points on the sphere $S^d$ obtained by this method are distorted, because spherical arcs corresponding to the intervals of the same length inside $F$ may have different lengths in $F'$. The amount of the distortion, and therefore bounds on the dispersion and discrepancy, can be obtained through the analysis of the resulting arc differences.

This idea can also be adapted to $SO(3)$ (and in general to the projective space of any dimension).
Take a four-dimensional regular polytope inscribed in $S^3$, and use only half of the faces to generate the distribution on the surface. Pick the faces so that in the set of used faces, there must not exist a pair of antipodal points, one from each of two different faces. This way the obtained samples cover exactly half of the sphere, which forms $SO(3)$ surface.

Next we show how to generate a layered Sukharev grid sequence on $S^d$ based on the inscribed cube and the bounds on the dispersion and the discrepancy of this sequence.

### 4.3.2 A Sample Sequence Based on Inscribed Cubes

In this section we first present a particular sequence adapted to spheres using the proposed general method, and next we analyze the uniformity properties of this sequence.

**Layered Sukharev Grid Sequence for a Spherical Cube**

Consider a face, $F$, of a $(d+1)$-cube inscribed in a sphere $S^d$. $F$ is a $d$-dimensional cube, which in each of its corners has $d$ edges. If we project all of these edges onto the surface of the sphere they form arcs, which delineate a spherical $d$-cube, $F'$. The lengths, $\alpha$, of these arcs are equal for all edges of $F$. If we consider those equatorial angles that correspond to the edges extending from a common vertex of $F$, we can define an *angular coordinate system* for the spherical face $F'$. Indeed, the coordinates $(x_1, x_2, \ldots, x_{d-1})$ with all possible values $x_i \in [0, \alpha]$ specify all possible points of $F'$.

The construction of the sequence, $T$, essentially follows the construction of the layered Sukharev grid sequence for the unit cube, except that instead of the Euclidean coordinate system we use the angular coordinate system defined above.

To analyze the dispersion and discrepancy of this sequence we need several definitions. Define the points of the *Sukharev spherical grid* of resolution $2^d$ as follows:

$$P^d_l = \left\{ \left( \frac{i_1 \alpha}{2^l} + \frac{1}{2^l+1}, \frac{i_2 \alpha}{2^l} + \frac{1}{2^l+1}, \ldots, \frac{i_d \alpha}{2^l} + \frac{1}{2^l+1} \right) : i \in \mathbb{Z}, 0 \leq i \leq 2^d - 1 \right\}.$$

Next we define the set of spherical canonical rectangles, which is an extension to the canonical rectangles defined in [131].

**Definition 4.1** Given positive integers $d$ and $m$, let $Q^d_m$ be the following family of the $d$-dimensional
spherical canonical rectangles:

\[ Q^d_m = \left\{ \left[ \frac{i_1 \alpha}{2^m}, \frac{(i_1+j_1) \alpha}{2^m} \right] \times \ldots \times \left[ \frac{i_d \alpha}{2^m}, \frac{(i_d+j_d) \alpha}{2^m} \right] : i, j \in \mathbb{Z}, 0 \leq i \leq 2^m - 1, 1 \leq j \leq \min(2^m - i, 2) \right\}. \]

The following results can be stated about the dispersion and discrepancy of \( T \).

**Proposition 4.2** The dispersion of the sequence \( T \) at the resolution level, \( l \), is

\[ d_\rho(T) \leq \frac{2\pi}{\sqrt{n(2^d - 1)} + 1}. \]

**Proof:** The largest spherical cap that does not contain any of the points in \( T \) will be smaller than the spherical cap with the center at \((\alpha/2, \alpha/2, \ldots, \alpha/2)\) and the spherical radius \( \pi/2^l \). Since \( 2^l = \left( \frac{\sqrt{n(2^d - 1)} + 1}{2} \right) / 2 \) we have that the dispersion is not bigger than \( \pi/2^l = \frac{\pi}{\sqrt{n(2^d - 1)} + 1} \).

**Proposition 4.3** The relationship between the discrepancy of the sequence \( T \) at the resolution level, \( l \), taken over \( \tilde{Q}^d_l = \bigcup_{m=0}^{l} Q^d_m \) and the discrepancy of the optimal over \( \tilde{Q}^d_l \) sequence, \( T_o \), is:

\[ D_{\tilde{Q}^d_l}(T) \leq D_{\tilde{Q}^d_l}(T_o) + (V_{\text{max}} - V_{\text{min}}). \]

**Proof:** The optimal sequence, \( T_o \), may place the points in some different order than \( T \). The maximal change in discrepancy that may occur in \( T \) comparing to \( T_o \) is the difference between the maximal, \( V_{\text{max}} \), and the minimal, \( V_{\text{min}} \), volumes of the spherical canonical rectangles. Therefore, \( D_{\tilde{Q}^d_l}(T) \leq D_{\tilde{Q}^d_l}(T_o) + (V_{\text{max}} - V_{\text{min}}) \).

**Proposition 4.4** The sequence \( T \) has the following properties:

- The position of the \( i \)-th sample in the sequence \( T \) can be generated in \( O(\log i) \) time.
- For any \( i \)-th sample any of the \( 2d \) nearest grid neighbors from the same layer can be found in \( O((\log i)/d) \) time.
Proof: For the $i$-th sample it takes $O(\log_{2^d} i) = O((\log i)/d)$ to find its resolution level $l$. Once $l$ is found, the corresponding point in Sukharev grid of resolution $2^l$ needs to be generated. It was proven in [131] that this takes $O(\log i)$. Therefore, the total running time for generating one point is $O((\log i)/d + \log i) = O(\log i)$.

The layer of the $i$-th sample is the Sukharev grid of resolution $2^l$. Any of the $2d$ nearest grid neighbors from this layer can be found in $O((\log i)/d)$ using the algorithm described in [131].

In our analysis we essentially ignored all of the points from the layers below the $i$-th sample layer, since the number of them is not significant. In practice, it may be efficient to use other layers for generating nearest neighbors. Better bounds on dispersion and discrepancy may also be achieved then.

Layered Sukharev Grid Sequence for $S^d$

Now, that we have defined a sequence for each of the spherical cubes, we need to define an ordering in which all of the points from those sequences will be placed on the surface of the sphere. One straightforward way to do this is to place one point from each of the faces’ sequences at a time. The order in which each face should be considered is decided from the following considerations.

Let the union of all of the spherical canonical rectangles determine the range space for the whole sphere. Using the criterion of optimizing the discrepancy over the range space, the ordering of the first $2(d+1)$ points for the resolution level 0 of the sphere can be explicitly computed. Hence, from this point on we can assume that we have such an ordering. Therefore, each next set of $2(d+1)$ points from each of the sequences should follow the same ordering, since this will minimize the discrepancy over the range space. This will guarantee that Proposition 4.3 holds for the generated sequence on the sphere.

Our ongoing research is directed on proving that the same result holds for the larger range spaces, i.e., the ones that include combinations of the spherical rectangles from different spherical cubes.

We can state the following result for the dispersion of the sequence, $T_s$, on the sphere:

**Proposition 4.5** The dispersion of the sequence $T_s$ at the resolution level $l$ containing

$$n = 2(d + 1) \cdot (2^{d(l+1)} - 1)/(2^d - 1)$$
\[
d_{\rho}(T) \leq \frac{2\pi}{\sqrt{n/(n^2 - 1)}} + 1.
\]

**Proof:** Applying the same argument as in the proof of Proposition 4.2, and considering that now

\[
2^l = \left(\sqrt{n/(n^2 - 1)}/(2(d + 1)) + 1\right)/2,
\]

we obtain the desired bound.

### 4.3.3 Sampling Inside Cartesian Products

We have shown how to define multiresolution grid sequences for the unit cube (Section 4.2.2), and the sphere \(S^d\) (Section 4.3.2). The spaces that arise in robotics are often the Cartesian products of these. For example, the set of all rotations and translations of a 3d rigid body, denoted as \(SE(3)\), can be represented by \(\mathbb{R}^3 \times SO(3)\). The rotations and translations of \(m\) multiple rigid bodies are represented by \((\mathbb{R}^3 \times SO(3))^m\).

When designing uniform sequences for such spaces the parametrization of the space together with the choice of measure and metric on the space should be defined carefully. Unfortunately, for \(SE(3)\) neither Haar measure nor natural metric exist. Instead, the weighted sum of the metrics on \(\mathbb{R}^3\) and \(SO(3)\) is usually used in practice. While the weighted metric can be defined on general Cartesian products of the spaces (which is assumed in the construction below), in some applications different techniques for designing sequences might be advantageous.

In what follows we construct the multiresolution grid sequence for the space that is a Cartesian product of multiple copies of \(\mathbb{R}^n\) and \(S^d\). We define it inductively, starting with any tuple of multiresolution grid sequences.

Let \(T_1\) and \(T_2\) be two multiresolution grid sequences. Let \(T_1\) be defined over the space \(X_1\), and \(T_2\) be defined over the space \(X_2\). Either of \(X_1\) or \(X_2\) may be \(\mathbb{R}^n\) or \(S^d\) (or \(SO(3)\)). Let \(\dim(T_1) = d_1\) and \(\dim(T_2) = d_2\) be the dimensions of these sequences. Let \(m_1\) \((m_2)\) be the number of points at the resolution level 0 of sequence \(T_1\) \((T_2)\) respectively. When weighted metric is defined on \(X_1 \times X_2\) the values for \(m_1\) and \(m_2\) can be chosen so that they respect the appropriate weights of \(X_1\) and \(X_2\).

Then the number of points at the resolution level \(l\) is \(m_1 \cdot 2^{l d_1}\) and \(m_2 \cdot 2^{l d_2}\) for sequence \(T_1\) and \(T_2\) respectively.

With each point \(p = (p_1, ... p_d)\) at the resolution level \(l\) one may define the grid region associated
GET SAMPLE($d$, $L$, origin, factor)
1  Point $sample \leftarrow$ origin;
2  $index \leftarrow n \% |L|$; // remainder of integer division
3  $nextN \leftarrow n \div |L|$; // quotient of integer division
4  $sample \leftarrow sample + (factor \times L[index])$;
5  if ($nextN = 0$)
6    return $sample$
7  else
8    $f \leftarrow factor \div 2$;
9    return GET SAMPLE($nextN$, $L$, $sample$, $f$)

Figure 4.5: Recursive generation of a new sample from the multiresolution grid sequence

with this point as $G_{p,l} = [p_1, p_1 + a/2^l] \times ... \times [p_d, p_d + a/2^l]$, in which $a = 1$ for the unit cube sequence and $a = \alpha$ (Section 4.3.2) for the spherical sequence.

Next consider the space $X = X_1 \times X_2$. The multiresolution grid sequence that we define for this space has $m_1 \cdot m_2 \cdot 2^{(d_1+d_2)}$ points at the resolution level $l$. Each of these points can be expressed as $p = (p_1, p_2) = (p_{1,1}, ... p_{1,d_1}, p_{2,1}, ... p_{2,d_2})$, in which $p_1 \in T_1$ and $p_2 \in T_2$. Each of these points has an associated grid region: $G_{p,l} = [p_{1,1}, p_{1,1} + a/2^l] \times ... \times [p_{1,d_1}, p_{1,d_1} + a/2^l] \times [p_{2,1}, p_{2,1} + b/2^l] \times ... \times [p_{2,d_2}, p_{2,d_2} + b/2^l]$.

The sequence for $X$ is constructed one resolution level at a time. The order in which the points from each resolution level are placed in the sequence can be described as follows. The ordering, $L_X()$, of the first $m_1 \cdot m_2 \cdot 2^{(d_1+d_2)}$ points determine the order of the grid regions within $X$ and should be precomputed in advance. Every successive $m_1 \cdot m_2 \cdot 2^{(d_1+d_2)}$ points in the sequence are placed in these grid regions in the same order. Where exactly each point is placed within each of the grid regions is determined by the recursion procedure defined for $[0,1]^{(d_1+d_2)}$ (see Section 4.2.2).

The $i$-th sample of this sequence is generated by the following function call:

$$GET\_SAMPLE(i \div |L_X|, L, L_X[i \mod |L_X|], 1),$$

in which the algorithm $GET\_SAMPLE$ is shown in Figure 4.5.

### 4.3.4 Sampling on $SO(3)$ Using the Hopf Fibration

Discretization of $SO(3)$, the space of 3D rotations, is a difficult problem that arises not only in motion planning, but also in numerous engineering and scientific fields. Examples include biological protein docking problems, aerospace trajectory design, and quantum computations. Typical operations on this space include numerical optimization, searching, integration, sampling, and path generation.
Multiresolution grids are widely used for many of these operations if the space is nicely behaved, as in the case of rectangular subsets of $\mathbb{R}^2$ or $\mathbb{R}^3$. It would be wonderful to achieve the same for $SO(3)$; however, the space of 3D rotations is substantially more complicated.

Due to widespread interest in discretizing $SO(3)$ in numerous fields, there have been considerable efforts in the past. The problem of generating point sets on spheres minimizing such criteria as energy functions, discrepancy, dispersion, and mutual distances has been extensively studied in mathematics and statistics [82, 139, 167, 185, 193, 202]. Random sampling methods were also developed in [10, 180, 189, 217]. Problems of sampling rotational groups and spheres have been studied and applied in the context of computational structural biology, physics, chemistry, computer graphics and robotics [53, 76, 141, 162, 165, 188, 203, 209, 210].

In Section 4.3.2 we have shown how to generate a grid sequence on $SO(3)$ taking advantage of the structure provided by the inscribed cube into a 3-sphere. However, this method results in distortions in the middle of the faces of the spherical hypercube, which get worse with the dimension. This is because the arcs of different lengths are getting subdivided into a grid on the spherical faces of the $d$-sphere. In case of $SO(3)$ some of the grid cells become four times the volume of others.

In this section, we introduce the best-known deterministic method to date for $SO(3)$ in terms of providing:

1. incremental generation,
2. optimal dispersion-reduction with each additional sample,
3. explicit neighborhood structure,
4. lowest metric distortion for grid neighbor edges,
5. equivolumetric partition of $SO(3)$ into grid regions.

**Method Overview**

Our method takes advantage of the fact that the fiber bundle structure of $SO(3)$ locally behaves exactly as the Cartesian product of two spaces, $S^1$, and $S^2$. Therefore, the method presented in Section 4.3.3 for constructing multiresolution grid sequences for Cartesian products of spaces, can be used for constructing a grid sequence on $SO(3)$. The resulting rotations are computed using the Hopf coordinates, as was first described in [149]. It is a much simpler problem to construct nicely behaved grids on the 1-sphere and 2-sphere. Hopf coordinates allow the two grids be lifted to the space of rotations without loss of uniformity. Next we outline the details of this construction.
Let $\psi$ be the angle parametrizing the circle, $S^1$, and $(\theta, \phi)$ be the spherical coordinates parametrizing the sphere, $S^2$. Using these coordinates, define $T_1$ to be the multiresolution grid over the circle, and $T_2$ be the grid over the sphere. Let $m_1$ and $m_2$ be the number of points at the base resolution 0 of the grids $T_1$ and $T_2$ respectively.

There are numerous grids that can be defined on $S^2$ (see Figure 4.6 for an illustration of some). In this work we have selected the HEALPix grid [76] on $S^2$, and the ordinary grid for $S^1$. Both of these grids are uniform, have simple neighborhood structure, and can have multiple resolutions.

Next consider the space $S^2 \otimes S^1$. The multiresolution grid sequence that we define for $SO(3)$ has $m_1 \cdot m_2 \cdot 2^3$ points at the resolution level $l$, in which every successive $2^3$ points define a cube in Hopf coordinates. Each element of the sequence is obtained by combining the corresponding coordinates in the subspaces, $p = (\theta, \phi, \psi)$. If the grid regions are defined on the two subspaces $S^1$ and $S^2$, the corresponding grid regions are also obtained on $SO(3)$ by combining the corresponding coordinates. The dispersion, and discrepancy of the resulting sequence can be easily computed using the representation for the metric and volume element from equations (2.18), and (2.22).

Choosing the Base Resolution

One of the issues arising when combining the two grids from $S^1$ and $S^2$ is the step length of a grid cell alone each of the coordinates. For this we have to match the number of cells in each base grids on both of the subspaces, so that they have cell sides of equal lengths [149]. That is the following equation should hold for $m_1$ and $m_2$:

$$\frac{2\pi}{m_1} = \sqrt{\frac{4\pi}{m_2}},$$

in which $2\pi$ is the circumference of the circle $S^1$, and $4\pi$ is the surface area of $S^2$.

In our particular case, the base HEALPix grid consists of $m_2 = 12$ cells, and the volume of each cell is equal to $4\pi/12 = \pi/3$ (Figure 4.7). Therefore, the length of the side of each grid cell is approximately the square root of that value, that is 1. Then, the number of points in the base resolution of the grid on $S^1$ needs to be $m_1 = 6$, since it should be close to the length of the circle, which is $2\pi$. Therefore, the base grid of the sequence for $SO(3)$ consists of $m_1 \cdot m_2 = 6 \cdot 12 = 72$ points (the projections of the grid regions on the Hopf coordinates are shown on Figure 4.8).

Choosing the Base Ordering

The next step is to choose the ordering of the $m = m_1m_2$ points within the base resolution on $SO(3)$. In general, the initial ordering will influence the quality of the resulting sequence, and a
method similar to [134] can be used for deciding the ordering of the general base sequences.

In our case we have to define the ordering on the first 72 points of the sequence (see Figure 4.8 for the illustration of the associated grid regions). To do this, it is important to notice that there are antipodal grid cells in both of the subspaces. Antipodal cells are the maximally distanced regions on both of $S^1$ and $S^2$; they should alternate in the final sequence. Therefore, it is only necessary to select an ordering on the first 18 points of the base resolution grid of $SO(3)$.

In our preliminary experiments in the application to motion planning problems (Section 4.4) we have manually selected such an ordering. However, it is possible to design a simple program that would run through the orderings and select the ones that minimize the discrepancy. For the further analysis results we assume that the optimal ordering function $f_{\text{base}} : \mathbb{N} \rightarrow [1, \ldots 72]$ is given.
Figure 4.7: Base grid of the HEALPix sequence consists of 12 points. The cylindrical projection of the grid cells from $S^2$ to $(\cos(\theta), \phi)$ coordinates is shown. Each next resolution subdivides each of the spherical squares into 4 squares of equal area. Figures are borrowed from [76].

The Sequence

The sequence for $SO(3)$ is constructed one resolution level at a time. The order in which the points from each resolution level are placed in the sequence can be described as follows. The ordering $f_{\text{base}}()$ of the first $m$ points in the base resolution determines the order of the grid regions within $SO(3)$ and is taken from the previous section. Every successive $m$ points in the sequence should be placed in these grid regions in the same order. Each of the grid regions is isomorphic to the $[0, 1]^3$, and is subdivided into 8 grid regions in each successive resolution. Where exactly each point should be placed within each of the grid regions is determined by the ordering $f_{\text{cube}} : \mathbb{N} \to [1, \ldots 8]$ and recursion procedure defined for the cube $[0, 1]^3$ in [131].

The resulting procedure for obtaining the coordinates of the $i$th element in the sequence is the following:

1. Assign $f_{\text{base}}(i)$ to be the index of the base grid region that the $i$-th element has to be placed within.
Figure 4.8: The base grid of the proposed SO(3) sequence consists of 72 points. For the Hopf coordinates \((\theta, \phi, \psi)\) the projections of the grid regions on each of the coordinates are shown. Grid regions for \(\psi\) are chosen according to the ordinary grid on \(S^1\). The grid regions for \((\cos(\theta), \phi)\) are obtained using the HEALPix method.

2. Assign the ceiling of the division, \(i_{\text{cube}} = \lceil i/m \rceil\), be the index that determines the subregion of the region \(f_{\text{base}}(i)\) that the \(i\)-th element has to be placed within.

3. Call the recursive procedure from [131] to determine the coordinates of the subregion of the cube \([0,1]^3\) determined by the index \(i_{\text{cube}}\) and the ordering \(f_{\text{cube}}\). The \(i\)-th element is then placed within this subregion of the \(f_{\text{base}}(i)\) region.

Analysis

Several claims, similar to those obtained for the Layered Sukharev Grid sequences, can be made for the new approach. The most important distinction is that the new sequence provides equal volume partition of the \(SO(3)\) which results in strong dispersion guarantees.

**Proposition 4.6** The dispersion of the sequence \(T\) at the resolution level \(l\) satisfies:

\[
\delta(T) \leq 2 \sin^{-1} \left( \frac{1}{2} \sqrt{\delta^2(T_2) + \left( \frac{\pi}{m1^2 l} \right)^2} \right),
\]

in which \(\delta(T_2)\) is the dispersion of the sequence \(T_2\) defined over \(S^2\).

**Proof:** The bound follows directly from the Pythagorean theorem [149], and the dispersion bound on the ordinary grid \(T_1\) at the resolution level \(l\).

**Proposition 4.7** The sequence \(T\) has the following properties:
Figure 4.9: Different sets of samples on $SO(3)$ (a) 2000 random samples; (b) 2048 Sukharev grid samples.

- It is discrepancy-optimal with respect to the set of grid regions defined over $S^1$ and $S^2$.
- The position of the $i$-th sample in the sequence $T$ can be generated in $O(\log i)$ time.
- For any $i$-th sample any of the $2^d$ nearest grid neighbors from the same layer can be found in $O((\log i)/d)$ time.

Proof: The proof directly repeats the reasoning in [131].

Visualization of the Results

To visualize our sequence and compare it with other sequences designed for $SO(3)$, we use the angle and axis representation from Chapter 2.2. It was shown in [44, 210, 150] that if the rotations are uniformly distributed, then the distribution of an angle is $(\sin(\theta) - \theta)/\pi$. This allows us to draw the elements of $SO(3)$ as the points inside a ball in such a way that every radial line has uniform distribution of elements. This provides a more intuitive visualization, which partially preserves the uniformity. See Figure 4.10 for visualization of several of the methods of sampling over $SO(3)$, compared to the proposed approach. Specifically, the images show points in the direction of the axis of rotation and with with distance to the origin equal to $(\sin(\theta) - \theta)/\pi$. Using this representation, the distribution of points increases linearly as a function of distance from the origin. In comparison,
a set of points that was uniform with respect to the measure on $\mathbb{R}^3$ would have a distribution that varies as the cube of distance from the origin.

### 4.4 Experimental Results: Application to Motion Planning

We have implemented our algorithm in C++ and applied to implementations of PRM-based planner [101] in the Motion Strategy Library. The experiments reported here were performed on a 2.2 Ghz Pentium IV running Linux and compiled under GNU C++.

It is important to note that the experiments we present here are just one of possible applications of the developed sequences to motion planning problems. It is possible that there exist a better use of the samples in other areas of computer science, or related fields.

Performance results are shown in Figures 4.11-4.12. The robots in the models on Figure 4.11 (a), (b) are allowed only to rotate; therefore, the configuration space is $SO(3)$. The robots in the models on Figure 4.12 (c), (d) are allowed to translate and rotate; therefore the configuration space is $\mathbb{R}^3 \times SO(3)$. We compared the number of nodes generated by the basic PRM planner using a pseudo-random sequence with quaternion components [180], with Euler components, the layered Sukharev grid sequences from Section 4.3.2, and the sequence for $SO(3)$ based on Hopf fibration from Section 4.3.4. The results for pseudo-random sequences were averaged over 50 trials. When we
tested the deterministic sequences, we made sure that each particular problem does not have any advantage due to coincidental alignment with the grid directions of the sequence. Therefore, in each trial a fixed, random quaternion rotation was premultiplied to each sample, to displace the entire sequence. The results obtained were averaged over 50 trials (a different random rotation was used in each).

Based on our experiments we have observed that the performance of the deterministic sequence is equivalent to the performance of the random sequence for the PRM-based planner, which makes it an alternative approach to random sampling. It is important to note, however, that for some applications, such as verification problem, only deterministic guarantees are acceptable, making random sequences inappropriate.

The results we obtained for the problem in Figure 4.11 (b) using Euler angles emphasizes the importance of using quaternions and sampling in a way that respects the Haar measure. This problem was never solved using the random Euler angles. The experiment was running for several days, generated 80000 nodes, but never found the solution. It is generally known that Euler angle parametrization has its drawbacks, such as gimbal lock and interpolation problems. However, in
Figure 4.12: Problems involving: (a) moving a robot from one corner of a 3d grid to the opposite corner; (b) moving an L-shaped object through the holes in the obstacles. The comparisons of the number of nodes generated by different sampling strategies are shown in the table.

<table>
<thead>
<tr>
<th></th>
<th>Random Sequence</th>
<th>Layered Sukharev Grid Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c)</td>
<td>3460</td>
<td>3285</td>
</tr>
<tr>
<td>(d)</td>
<td>3481</td>
<td>3202</td>
</tr>
</tbody>
</table>

motion planning, it has been a popular way to parametrize rotations. This example demonstrates the inadequacy of Euler angles parametrization. The interpolation method, ignoring the dependence between the three rotations (yaw-pitch-roll), tries to rotate around three axes simultaneously. In the configuration space with the narrow corridor this results only in those configurations that are in collision.
Chapter 5

Motion Planning for Highly Constrained Spaces

5.1 Introduction

In many motion planning problems, the feasible subspace becomes thin in some directions. This is often due to kinematic closure constraints, which restrict the feasible configurations to a lower-dimensional manifold or variety. This may also be simply due to the way the obstacles are arranged. For example, is planning for a closed chain different from planning a sliding motion for a washer against a rod? An illustration for the two instances of such problems is shown on Figure 1.5. Traditionally, the two problems are solved with different methods in motion planning. However, the two seemingly different problems have similar algebraic and geometric structure of the sets of feasible configurations. In either case, there may exist functions of the form $f_i(q) \leq \epsilon_i$ that contain most or all of the feasible set (note that the $f_i$ are not necessarily explicitly defined). Typically, $\epsilon_i$ is very small, and in the case of closed chains, $\epsilon_i = 0$. When this occurs, a region of the feasible space has small intrinsic dimensionality, in comparison to the ambient configuration space. In Chapter 2.1.1 we have formulated the problem of planning on such spaces formally.

In automated manufacturing, and manipulation planning problems similar to the two discussed above are abundant. In some cases the functions $f_i$ are explicitly provided. A large set of mechanisms, such as PUMA robots, or humanoid robots are well studied, with readily available expressions for the forward kinematics, and grasping configurations. Many practical solutions exists for solving such problems. Very often, however, the expression for the constraints is not readily available, such as for the example of the washer and the rod. There are thousands of CAD models of mechanisms, and objects (including rods) of different shapes, designed for manipulation systems. There are fewer methods available for such problems, unless the constraints are explicitly modeled; it is frustrating that no technique exists for solving these problems in a unified way. In this chapter we propose a single framework for solving the problem formulated in Chapter 2.1.1.

Traditionally, the problem of planning for closed chains is solved differently from the classical
motion planning problem, since the kinematic constraints are given a priori. Analytical approaches construct explicit geometrical and topological representation of the closure set [37, 135, 81], but are usually inefficient in practice. Practical sampling-based methods [208, 22] usually project the closure set on the subset of parameters, on which the planning is performed [80, 50]. An inverse kinematics solver is used in these approaches as a black box to get the solution back on the configuration space. These approaches are quite successful in practice, however, there are disadvantages associated with using inverse kinematics:

- Analytical solutions for inverse kinematics for arbitrary manipulators are prohibitively complex and can only be applied to relatively simple mechanisms with low number of degrees of freedom [58].

- Numerical techniques exist for solving inverse kinematics for arbitrary mechanisms [161]; however, they are not presently at the stage of being applied in practice.

- Even if inverse kinematics solutions were readily available, the choice of the subset of parameters on which projection is made may significantly affect the performance of a planner. Planning becomes inefficient around singularities associated with the projection on the chosen subset of parameters. There is no general method, though, for the choice of parameters. In practice, a careful analysis of the linkages is usually required before the method is applied [50].

Numerous techniques were also developed for motion planning problems with constrained geometries, such as the example of sliding the washer against the rod. Several approaches were proposed recently [4, 22, 64, 88, 216] for solving such problems. However, they can not handle kinematic constraints.

In this chapter we propose a motion planner that can handle planning under the set of constraints of the form $f_i(q) \leq \epsilon_i$. It needs neither an inverse kinematics solver, nor an explicit expression for $f_i$, which provides maximum flexibility to the user. The method keeps the representation of the set of feasible configurations in the dynamic domain, which captures the intrinsic dimensionality of the subsets in an ambient space. We propose two ways of implementing the dynamic domain. Both of the approaches rely on nearest neighbor computations to maintain the feasibility region, we use kd-trees for efficient implementation in one of the approaches. The planning is done locally using the RRT planner [108] inside the dynamic domain. Our method uses the aggressive exploration properties of RRTs to focus the sampling and searching only inside the represented portion of the feasible set in configuration space.
Figure 5.1: Trees constructed by an RRT-based planner and the Voronoi regions associated with the nodes of the trees. (a) A bug trap problem in high dimensions can be a challenging problem for this planner. (b) It becomes much more challenging when the region for sampling is enlarged.

We have tested the method on a dozen problems, including both basic motion planning and planning under closed-chain kinematic constraints. We observed that the performance of the method improves over the original RRT planner on most of the problems, often by an order of magnitude. In rare cases, when there is no improvement in the running time, the proposed method works only slightly worse than the RRT.

In this chapter we propose two ways to explore thin feasible spaces. In Sections 5.2-5.3 we describe our first approach, which is applicable to low dimensional configuration spaces. We present experimental results for this approach in Section 5.4. In Sections 5.5-5.7 we present the description and experimental results for the generalized implementation of the first approach, which is based on kd-trees from Chapter 3.

5.2 Maintaining the Dynamic Domain

We start the description of our first method by re-examining the Voronoi biased exploration strategy of the RRTs on one challenging example. In the end of this section, we formulate the problem of controlling Voronoi bias for more efficient exploration of the configuration space.
5.2.1 Motivating Example

Consider a problem shown in Figure 5.1(a). The task is to move the robot outside of the bug trap\(^1\). Since the size of the free space inside the trap is considerably larger than the narrow opening, in high dimensions it can be a very challenging problem for any motion planner. Now consider the Voronoi regions of the set of sample points built by the RRT planner (Section 2.3.1) inside the trap. There is a considerable bias toward the points near the obstacles. That is, most of the points on the boundary of the trap will have higher probability of being selected for extension than the points near the opening accordingly to the sizes of the corresponding Voronoi regions.

The problem becomes even more challenging if the sampling region is enlarged. It may become so difficult that a regular RRT will not be able to solve the 2-dimensional problem shown in 5.1(b) in any reasonable time. This is explained by an even larger bias toward the points near the obstacles. In fact, the Voronoi regions of these points grow with the size of the environment. Meanwhile, the tree in the middle of the bug trap does not grow at all.

The obvious solution to this problem would be to limit the sampling region to fit the bug trap. However, while this would help in this easy case, the approach would not be general enough to deal with other motion planning problems.

In what follows we distinguish different types of nodes in the tree. We call frontier nodes the vertices in the tree that have their Voronoi regions growing together with the size of the environment. The boundary nodes are those that lie in some proximity to the obstacles. It is important to note that the frontier vertices provide especially strong bias toward the unexplored portions of the configuration space. In many cases this helps the tree to rapidly grow. However, this may cause a slow-down in the performance when a frontier point is also a boundary point. For example, in the Figures 5.1(a) and (b), all the frontier points are also boundary points. The problem is that they have high probability of being chosen for extension in the direction of the obstacles, but most of the times the extension cannot be performed.

In general, boundary points are given more Voronoi bias than they can explore. As a consequence, prohibitively many expensive operations are being performed during the execution of the RRT planners. For example, interpolation between the new configuration and its nearest node in the tree is required at each iteration in the holonomic version of the RRTs. This involves several collision checks being performed along the interpolation path, which may be very expensive in the environments with complicated geometry [64]. Inverse kinematics, together with interpolation,

\(^1\)This name is inspired by actual devices for catching bugs. Bugs can enter the device easily, but it is hard for them to escape. The analogy was suggested by James O'Brien in a conversation with James Kuffner.
should be performed at each iteration of some RRT-based planner for systems with closed linkages [51]. For nonholonomic planners [124, 42], some integration is also required at each iteration.

Thus, the goal of this chapter is to find a way of reducing the number of expensive iterations in RRTs by controlling the Voronoi bias of the nodes in the tree. We define the problem more formally in the next subsection.

5.2.2 Problem Definition

Let $C$ be an $n$-dimensional configuration space, and $C_{obs}$ be the set of obstacles in this space. Let $V$ be a set of $N$ collision free points lying inside $C_{free} = C \setminus C_{obs}$ (i.e. the current RRT’s nodes), and $D$ be the Voronoi diagram of $V$.

**Definition 5.1** Let $\mathcal{L}$ be any local method that computes a path $\mathcal{L}(v, v')$ (e.g., the straight line segment) between two given nodes in the tree $v$ and $v'$. We define the visibility domain of a point $v$ for $\mathcal{L}$ as in [181]:

$$\text{Vis}_{\mathcal{L}}(v) = \{v' \in C_{free} \text{ such that } \mathcal{L}(v, v') \in C_{free}\}$$

**Definition 5.2** For a point $v \in V$ and its Voronoi region $D(v)$ define the visible Voronoi region of $v$ to be $O(v) = \text{Vis}(v) \cap D(v)$. 

Figure 5.2: For a set of points inside a bug trap different sampling domains are shown: (a) regular RRTs sampling domain, (b) visibility Voronoi region, (c) dynamic domain.
Now consider the union of all the visible Voronoi regions $\bigcup_{v \in V} O(v)$, which together comprise the visibility region of the set of points $V$. A uniform distribution over $\bigcup_{v \in V} O(v)$, which we call a visibility distribution, would be more appropriate for the RRT planner than the uniform distribution over the configuration space. Indeed, this distribution does not have bias for exploring toward the obstacles. At the same time it includes the important bias toward the unexplored parts of the configuration space. Examples of the visibility distribution for a linear local planner and the uniform distribution over all configuration space for the set of points inside the bug trap are shown in Figures 5.2(a) and 5.2(b).

Ideally, the distribution over the visible Voronoi diagram should not be uniform. It should concentrate more points inside the narrow corridors and less points in larger open areas of the space. This by itself is a very hard problem, since finding narrow corridors may be harder than solving the original motion planning problem. Producing a uniform distribution from the visible Voronoi regions is already quite a challenging task. In this work we propose another distribution, which is easier to compute but retains some useful properties of the visibility distribution. The next section describes this approach.

### 5.3 Dynamic-Domain RRT Path Planner

Although the visibility distribution provides the necessary bias to the RRT planners, computing it may be expensive. Consider computing the visible Voronoi region $O(v)$ of some node $v$ in the tree. Finding the points in $D(v)$ that are visible from $v$ may require performing interpolation and, therefore, expensive collision detection calls. This is exactly what planners should try to avoid when the interpolation is expensive. Therefore, we consider another distribution which we define as follows.

**Definition 5.3** Given a boundary point $v$ at distance at most $\epsilon$ from an obstacle in $C_{\text{obs}}$, define the boundary domain for $v$ as the intersection of the Voronoi region of $v$ and an $n$-dimensional sphere of radius $R$, centered at $v$.

**Definition 5.4** The dynamic domain of radius $R$ for the set of points $V$ is the boundary domains of the boundary points combined with the Voronoi regions of all other points. The uniform distribution over this domain is called the dynamic domain distribution.
The differences between the original RRT's sampling domain, the visibility region and dynamic domain for a set of points inside a bug trap can be seen on Figure 5.2. When a point is quite far from the obstacles, its boundary domain is the same as the RRT's sampling domain, that is, the whole Voronoi region (Figure 5.3). On the other hand, when a point is a boundary point, the boundary domain may be much smaller than both the visible Voronoi region and the whole Voronoi region (Figure 5.4).

The $\epsilon$ parameter should be chosen carefully. It should be sufficiently small, so that the point becomes a boundary point when most of the attempts to interpolate from it fail. On the other hand, it should not be smaller than the resolution of the tree, to reduce the number of redundant nodes. Therefore, $\epsilon$ can be naturally chosen as the interpolation step in the connect function of the RRT planner. The radius $R$ is chosen as a multiple of $\epsilon$. 

---

Figure 5.3: For a non-boundary point different sampling domains are shown: (a) regular RRTs sampling domain, (b) visible Voronoi region, (c) dynamic domain.

Figure 5.4: For a boundary point different sampling domains are shown: (a) regular RRTs sampling domain, (b) visible Voronoi region, (c) dynamic domain.
BUILD_DYNAMIC_DOMAIN_RRT(q_{init})
1  \mathcal{T}.init(q_{init});
2  for k = 1 to K do
3      repeat
4          q_{rand} ← RANDOM_CONFIG();
5          q_{near} ← NEAREST_NEIGHBOR(q_{rand}, T);
6      until dist(q_{near}, q_{rand}) < q_{near}.radius
7  if CONNECT(\mathcal{T}, q_{rand}, q_{near}, q_{new})
8      q_{new}.radius = \infty;
9      \mathcal{T}.add_vertex(q_{new});
10     \mathcal{T}.add_edge(q_{near}, q_{new});
11  else
12      q_{near}.radius = R;
13  Return \mathcal{T};

Figure 5.5: The dynamic domain RRT construction algorithm

5.3.1 Implementation

To obtain the dynamic domain distribution, we generate a distribution from the configuration space \( C \), and then restrict it to the dynamic domain. Given that the original distribution was uniform, the obtained restriction is also uniform. Computing this restriction corresponds to the lines 3-6 in the Figure 5.11. The radius field of each point stores value \( R \), if it is a boundary point, and value \( \infty \) otherwise. It is important to note, that for this step to be efficient the random configurations should be chosen from the area closely fitting the dynamic domain. In our experiments this area is the smallest bounding box among all boundary domains.

Next, we show how to incorporate sampling from the dynamic domain in the RR T-CONNECT planner. The complete pseudocode is shown on figure 5.11. The algorithm updates the information about the boundary points on the fly. At the beginning of the exploration of the tree, all points are considered to be non-boundary. As soon as the interpolation from one of the nodes fails (meaning that the distance to the obstacles is at least \( \epsilon \), where \( \epsilon \) is length of the interpolation step) the point becomes a boundary point. This corresponds to the lines 11-12 in the code. The radius field of this point is updated to \( R \). Next time the samples from the Voronoi region of this point are restricted to its boundary domain.

It is important to note that the dynamic domain RRTs retain the probabilistic completeness of the original RRTs. The argument follows closely to the one presented in [108]. Deterministic, resolution completeness can alternatively be obtained (which results in an RDT) by using a dense sample sequence in the place of the random sequence [119].

Another note is that the proposed method changes its behavior with the size of the radius parameter \( R \). When the radius is chosen to be \( \infty \), this dynamic domain RRT is the same as the
original RRT. As the radius value becomes smaller, the behavior of the dynamic domain RRT is more greedy. The tree tends to produce more nodes in the free space, since the bias of the boundary points is reduced. Therefore, efficient nearest neighbor methods adapted to the topology of the configuration space should be used [15]. Unfortunately, we do not have theoretical characterization of the dependence of the performance of the dynamic domain RRT from the radius parameter. However, there is a strong relationship between this parameter and the interpolation step size of the RRT. For all our experiments we set $R = 10\epsilon$.

### 5.4 Experimental Results

We have implemented our algorithm in C and incorporated it into the software platform Move3D [182] developed at LAAS. The experiments were performed on a 333 MHz Sunblade 100 running SunOS 5.9 and compiled under gcc 3.3. We have compared the performance of the bidirectional balanced RRT-CONNECT algorithm and Dynamic-Domain bidirectional RRT-CONNECT. For each
of the experiments we show the running times, the number of nodes in the solution trees and the number of the collision detection (CD) calls during the construction process, averaged over 50 runs.

We first show the results obtained for a bug trap in two dimensions (Figure 5.6). We have picked several different environments to demonstrate the deterioration of the performance of the classical bi-RRT planner with the environment size. In comparison, the average running times of the dynamic domain RRT do not change.

Next experiment was performed on the molecule model shown in Figure 5.7. Since the molecule is modeled as a 3D rigid body, the configuration space is 6-dimensional. The task is to compute the pathway of a ligand (i.e. the small molecule displayed in black) to the active site located inside the protein model. The motion planning problem is relatively easy here, since the number of nodes required to find a solution by a regular bidirectional RRT is 400. However, since the collision detection calls are very expensive, the performance of the regular bi-RRT is poor. Therefore, setting up a small radius parameter, which results in a larger number of nodes constructed by the dynamic domain RRT, results in much faster running times.

Figure 5.7: Molecule example. The task is to compute the pathway of a small molecule to the active site located inside the protein model.
Next example in Figure 5.8 is a benchmark from the automotive industry. Automotive industry provides important applications for motion planning, since the industrial companies can verify the manufacturing process and/or the maintainability of the assembly by computing the solution to the motion planning problem. The goal in this example is to find a collision free path dismounting a wiper motor from a car body. This is a real industrial problem which is highly constrained and was solved before with the method described in [64]. A weighted metric that highly favors translations is usually used for this environment. The original bi-RRT is not able to solve this problem after running for several days. The dynamic domain RRT solves it on average in several minutes.

The problem shown in Figure 5.9 is a motion planning problem for a two link articulated body with 4 degrees of freedom. The goal is to move the robot from one corner of the labyrinth to another. The collision checks are relatively cheap in this problem and there are several narrow passages. This problem is easily solved by the original RRT method, however, the dynamic domain RRT is still advantageous.
5.5 Using Kd-trees as Dynamic Domain

The method we have just described works well for low dimensional configuration spaces. When the dimension increases, the volume of the spheres in the dynamic domain decreases with respect to the volume of the enclosed rectangle, and the computation of the loop in Lines 3-6 of the algorithm on Figure 5.11 becomes prohibitively expensive.

In this section we propose another way of implementing the dynamic domain, by using kd-tree based representation of the feasible configuration spaces. Kd-trees [13] can handle up to 25-50 degrees of freedom and millions of nodes, are able to significantly speed up nearest-neighbor calculations in motion planning applications (Chapter 3), and have intrinsic dimensionality reduction abilities.

We describe the new dynamic-domain RRT in Section 5.5.2, and the kd-tree data structure for the dynamic domain in Section 5.6. First we review a simple adaptation of the basic RRT algorithm to solve the motion planning problem on feasible spaces.
5.5.1 The RRT Algorithm

Consider the pseudocode for building an RRT in $C_{feas}$ shown on Figure 5.10. At iteration $k = 1$ an RRT contains only the initial configuration, $q_{init}$. At each iteration the RRT grows, until either it contains the goal configuration (that is, a path from $q_{init}$ to $q_{goal}$ is found which is a branch in the RRT), or a limit on the number of iterations is reached. To grow the RRT, a random configuration, $q_{rand} \in C$, is chosen in Line 3. This configuration is not added to the tree, thus the constraints $f_i(q) \leq \epsilon_i$ need not be satisfied at this step. In Line 4, the closest to $q_{rand}$ configuration $q_{near}$ from the nodes in RRT is selected. The connection from $q_{near}$ to $q_{rand}$ is attempted in Line 5. This corresponds to the interpolation between $q_{near}$ and $q_{rand}$, such that the furthest configuration $q_{new} \in C_{feas}$ along the interpolation path from $q_{near}$ is returned. If the interpolation step is successful, the new vertex $q_{new}$, and corresponding edge are added to the RRT. The function CONNECT performs interpolation and validation of the constraints $f_i(q) \leq \epsilon_i$. The geometrical illustration of the algorithm is shown on Figure 5.10.

This algorithm requires neither parametrization of the points in $C_{feas}$, nor an inverse kinematics solver in case of planning for closed chains. This makes each line in the procedure very efficient. Given that $n$ points were built by an RRT at a particular iteration, and the dimension of the
configuration space $C$ is $d$, the sampling step in Line 3 takes $O(d)$ time, and the nearest neighbor call in Line 4 takes $O(2^d n \log n)$ running time [15]. The computational time in this algorithm is not spent at any particular line of the pseudocode, but on the number of iterations needed for solving a problem. The drawback of this approach is similar to the one outlined in [212]. That is, the Voronoi bias of the points in the RRT determine the exploration behavior of the RRT. Since the sampling from the configuration space $C$ in Line 3 does not take into account neither obstacles of configuration space, nor the constraints $f_i(q) \leq \epsilon_i$, the same extensions are repeatedly attempted towards invalid configurations. This increases the number of iterations needed to solve a problem. The next section presents a better suited approach for constrained feasible spaces.

5.5.2 Dynamic Domain Sampling for RRT

To improve the Voronoi bias in the RRT exploration the approach in [212] proposes to maintain a dynamic domain $D$, that approximates the feasible configuration space with a simple shape, such as a collection of balls. This significantly reduces the effect of local minimum problem for RRTs. The sketch of the algorithm and its geometrical illustration are shown on Figure 5.11. The difference from the baseline RRT algorithm is in Line 3, in which dynamic domain is used for sampling instead

---

**Figure 5.11:** The DDRRT-CONNECT construction algorithm.
BUILD_KD_TREE\( (P, d, m, b, r) \)

*Input:* A set of points \( P \), the dimension of the space \( d \), the number of points to store in a leaf \( m \), the bounding box \( b \) for \( P \), and the thickness parameter \( r \).

*Output:* The root of a kd-tree storing \( P \)

1. \( \text{if} \ P \ \text{contains less than} \ m \ \text{points} \)
2. \( \text{return} \ \text{a leaf storing these points,} \)
3. \( \hspace{1cm} \text{and an} \ r \text{-bounding box for} \ P \)
4. \( \text{else} \ \text{Split} \ b \ \text{into two boxes,} \ b_1, \ b_2. \)
5. \( \hspace{1cm} \text{Find} \ P_1 \ \text{and} \ P_2, \ \text{the sets of the data points} \)
6. \( \hspace{2cm} \text{inside boxes} \ b_1 \ \text{and} \ b_2. \)
7. \( \hspace{1cm} v_1 = \text{BUILD_KD_TREE}(P_1, d, m, b_1, r) \)
8. \( \hspace{1cm} v_2 = \text{BUILD_KD_TREE}(P_2, d, m, b_2, r) \)
9. \( \hspace{1cm} \text{Create node} \ v \ \text{storing the splitting plane} \ l, \)
10. \( \hspace{2cm} \text{splitting dimension} \ k, \ \text{bounding box} \ b, \)
11. \( \hspace{2cm} \text{and nodes} \ v_1 \ \text{and} \ v_2, \ \text{the children of} \ v. \)
12. \( \hspace{1cm} \text{area} = v_1.\text{area} + v_2.\text{area} \)
13. \( \hspace{1cm} \text{height} = \text{max}(v_1.\text{height}, v_2.\text{height}) + 1 \)
14. \( \text{return} \ v \)

Figure 5.12: The algorithm for constructing a kd-tree for a set of points \( P \).

of the configuration space. Assume again that \( n \) points were built by an RRT at a particular iteration, and the dimension of the configuration space \( C \) is \( d \). For the kd-tree dynamic domain, the uniform sampling in Line 3 of the algorithm on Figure 5.11 is performed in \( O(\log n) \) running time, the nearest-neighbor call in Line 4 requires \( O(2^d n \log n) \) time, and the update function in Line 8 takes \( O(\log n) \) time. Using the kd-tree data structure, the cost of each iteration is slightly increased, comparing to the original RRT algorithm on Figure 5.10. However, the number of iterations is usually reduced in the planning process, and therefore, the running time improves overall.

The next section describes the implementation details for the kd-tree data structure to perform efficient sampling and update on the explored portion of the feasible configuration space.

### 5.6 Representing Feasible Configuration Spaces with

**Kd-trees**

Consider the set \( S \) of data points lying inside a \( d \)-dimensional enclosing rectangle. We build the kd-tree inside this rectangle, and define it recursively as follows. The set of data points is split into two parts by splitting the rectangle that contains them into two children rectangles by a hyperplane, according to a specified rule; one subset contains the points in one child box, and another subset contains the rest of the points. The information about the splitting hyperplane and the boundary values of the initial box are stored in the root node, and the two subsets are stored recursively in the
UPDATE_KD_TREE(p, d, m, v, b)

Input: The point to be added $p$, the number of points to store in a leaf $m$, the node $v$ (initially the root), and the bounding box $b$ for $v$.

Output: The root of a kd-tree which includes $P \cup p$

1. if $v$ is a leaf
2. if $(v.size > 2m)$
3. BUILD_KD_TREE($v.P \cup p, d, m, b, r$)
4. else consider the two subboxes, $b_1, b_2$ of $b$
5. $d_1 = \text{dist}_{TK}(q, b_1)$
6. $d_2 = \text{dist}_{TK}(q, b_2)$
7. if $d_1 < d_2$
8. then $v_1 = \text{UPDATE_KD_TREE}(p, m, b_1, v_1)$
9. else $v_2 = \text{UPDATE_KD_TREE}(p, m, b_2, v_2)$
10. if $(v_1.height > 2v_2.height)$ or $(v_2.height > 2v_1.height)$
11. BUILD_KD_TREE($v.P \cup p, d, m, b, r$)
12. area = $v_1.area + v_2.area$
13. height = max($v_1.height, v_2.height$) + 1
14. return $v$

Figure 5.13: The algorithm for updating the kd-tree with a new point $p$.

two subtrees. When the number of data points contained in some box falls below a given threshold, $m$, the node associated with this box is called a leaf node, and a list of coordinates for these data points is stored in this node.

We divide the current cell through the median of the points orthogonally to the cell’s longest side. If there are ties then we select the dimension with the largest point spread. This ensures that the resulting kd-tree for $n$ data points is balanced, with the height of the tree equal to $O(\log n/m)$.

We introduce the notion of $r$-bounding rectangle for a set of points $P$ at a node $m$ of the kd-tree, as an intersection of the bounding rectangle of the node $m$ and a rectangle with the two opposite points $p_1$ and $p_2$ defined as $p_1 = (\min\{p^j_i | p^j_i \in P\}_{j=1}^d - \vec{r})$, and $p_2 = (\max\{p^j_i | p^j_i \in P\}_{j=1}^d + \vec{r})$. Here, $r$ is the parameter that determines the thickness of the dynamic domain for the points in the RRT. This parameter is manually selected in our current implementation. However, for more effective performance, an automatic parameter tuning similar to [95] can be implemented.

The kd-tree dynamic domain is defined as the collection of all of the $r$-bounding rectangles of the points at the leaves of the kd-tree.

Next, we outline the four main functions for the kd-tree needed to implement the algorithm on Figure 5.11.
5.6.1 Construction

Our kd-tree is constructed using a recursive procedure, which returns the root of the kd-tree (see Figure 5.12). This construction algorithm is essentially identical to the case of constructing a kd-tree in an Euclidean space [151]. The differences in the implementation come from the need to maintain additional information, such as $r$-bounding rectangles, and the heights of the tree and each of the subtrees. This is needed for efficient sampling (Section 5.6.4) and update (Section 5.6.2) of the kd-tree.

The running time for building a kd-tree for a set of $n$ points is $O(n \log n)$ [13].

5.6.2 Dynamic Update

When a new point is added at line 8 on Figure 5.11, the kd-tree is updated according to the algorithm shown on Figure 5.13. First, the algorithm descends to a node, to which the new point belongs, and such that the height balance at line 10 of the algorithm would become invalid for the two children.
KDTree::SAMPLE()
Output: sample q
1  q = root.SAMPLE()
2  return q

Node::SAMPLE()
1  \( p_1 = \frac{v_1.\text{area}}{\text{area}} \)
2  \( p_2 = \frac{v_2.\text{area}}{\text{area}} \)
3  with probability \( p_1 \)
4  \( q = v_1.\text{SAMPLE()} \)
5  with probability \( p_2 \)
6  \( q = v_2.\text{SAMPLE()} \)

Leaf::SAMPLE()
1  \( q = \text{RANDOM_CONFIG(bnd\_box)} \)

Figure 5.15: The algorithm portions for searching a kd-tree on the root level and internal and leaf nodes levels.

nodes after adding the point. Next, the construction procedure is called on this node. The area of the \( r \)-bounding boxes and the heights of the subtrees are then updated.

Given that there are \( n \) points in the kd-tree, the time to perform one update consists of \( O(\log n) \) to descent to the node that needs rebalancing, and of \( O(\hat{n} \log \hat{n}) \) to rebalance this node, given that it has \( \hat{n} \) points. The running time of the algorithm depends on the node which is rebalanced. Over many runs, the nodes with large number of points are rebalanced fewer times than the leaves of the tree. In fact, the node with \( n \) points in the worst case is rebalanced only during every \( n \)-th run. Therefore, the amortized analysis yields \( O(\log n) \) worst case running time for this procedure.

Figure 5.14 (a) illustrates how the update procedure works, and how the corresponding binary tree looks like for 800 data points incrementally added to the tree. Figure 5.14 (b) shows two kd-trees for the set of points among the obstacles in configuration space. Each kd-tree in this figure corresponds to a different \( r \)-parameter.

5.6.3 Nearest-Neighbor Query

The query phase is performed identically to the procedure outlined in [13] and [15]. Therefore, we omit the discussion about it here. We only note that the query is performed in \( O(2^d \log n) \) time [13], where \( d \) is the dimension of the configuration space \( C \). This is the most expensive operation in a single iteration of the RRT algorithm on Figure 5.11.
5.6.4 Uniform Sampling

To sample the area represented by the kd-tree, the algorithm first descends to a leaf, with probability corresponding to the area of the $r$-bounding rectangle of the leaf. Next, it returns a sample from the $r$-bounding box of the leaf. The procedure is outlined on Figure 5.15. The running time of the procedure is $O(\log n)$, since only one descend along the tree is needed. The resulting samples are guaranteed to be uniformly distributed over the collection of $r$-bounding boxes, since the boxes from different leaves do not intersect.

5.7 Experimental Results

We have compared the performances of the two RRT algorithms described in Figures 5.10 and 5.11. For each of the experiments, we show the running times, the number of nodes in the solution trees and the number of collision detection calls (CD) during the construction process averaged over 50 runs. The $r$-parameter was manually selected in these experiments.

The first experiment is for a 2d closed chain consisting of 12 identical rectangular links. All of the joints are revolute. The performance comparison of both of the RRT algorithms is shown on Figure 5.16. The improvement of the kd-tree-based approach is around 30 times over the original RRT in this example.

The second experiment does not involve closed chains explicitly (Figure 5.17). The goal in this example is to move a closed chain with 12 links and revolute joints from the left to the right part of the environment through a narrow opening.

<table>
<thead>
<tr>
<th>2dLoop3</th>
<th>RRT</th>
<th>DDRRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>time/sec</td>
<td>170.70</td>
<td>5.5</td>
</tr>
<tr>
<td>nodes</td>
<td>4,347</td>
<td>881</td>
</tr>
<tr>
<td>CD calls</td>
<td>20,667</td>
<td>11,408</td>
</tr>
</tbody>
</table>
The goal is to use the PUMA robot to move the toroidal object from one rod in the work space to the other. Using the kd-tree approach there is an improvement of an order of magnitude in the running time for this experiment.

The third example involves unfolding a 3d closed chain with revolute joints and 10 degrees of freedom through a cloud of obstacles (Figure 5.18). The performance improvement in this experiment is not as significant as in the other examples. We speculate that adaptive tuning of the $r$-parameter is significant for solving the problem efficiently. There are several thin sheets in the configuration space of this problem, each of which requires a different $r$-parameter value. There is an improvement of 11% in this example using the current implementation of the kd-tree approach.

The final experiment involves two PUMA robots holding an object, which needs to be moved from the back to the front of the environment (Figure 5.19). The kd-tree approach gives an 8 times improvement in the running time of the RRT algorithm.

Besides the experiments reported in this chapter, we conducted around twenty other experiments for both basic motion planning problems, and problems involving closed kinematic chains. We observed that the approach using kd-trees provides significant improvement on most of the problems we have tested. The improvement is either by orders of magnitudes, or just by several times.
Figure 5.18: The goal in this example is to unfold a 3d-closed chain with 12 links amidst the cloud of obstacles.

<table>
<thead>
<tr>
<th>3dLoop1</th>
<th>RRT</th>
<th>DDRRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>time(sec)</td>
<td>178.6</td>
<td>124.85</td>
</tr>
<tr>
<td>nodes</td>
<td>138</td>
<td>208</td>
</tr>
<tr>
<td>CD calls</td>
<td>4,143</td>
<td>2,481</td>
</tr>
</tbody>
</table>

Only for three out of twenty problems we noticed either no significant improvement, or a slight deterioration (not more than 10%) in the performance of the kd-tree-based algorithm. This suggests that our approach is promising to provide uniform running time improvement on a large set of motion planning problems.
Figure 5.19: This experiment involves two PUMA robots holding an object, which needs to be moved from the back to the front of the environment.
Chapter 6

Conclusions

In summary, the thesis presented solutions to several fundamental problems in motion planning. By a careful consideration of the topology of common configuration spaces, we achieved better understanding and more efficient practical implementations of the ISS framework in motion planning. The core issues addressed were: fast nearest neighbor generation, uniform deterministic sampling techniques, and guided sampling and efficient exploration in configuration spaces arising in motion planning.

To address the first issue, in Chapter 3 we presented a practical algorithm for efficient nearest-neighbor search for the most common manifolds arising in motion planning. We illustrated the importance of performing efficient nearest-neighbor computations in the context of path planning. Our method extends previous techniques that were designed for Euclidean spaces by building kd-trees that respect topological identifications and the resulting distance metrics. Our method has been implemented, and was observed to be orders of magnitude faster than the naive nearest-neighbor searching in up to 50-dimensional configuration spaces. Note that it is substantially faster even in high-dimensional spaces, which are of great importance in motion planning. We evaluated the implemented algorithm as a means to accelerate performance in both PRM and RRT algorithms. Substantial improvement was observed in both cases; however, it is important to note that the benefits are substantial only if the nearest-neighbor computations dominate the total running time, which happens when a large number of nodes is generated by a planning algorithm. Collision detection is a competing bottleneck in path planning algorithms; therefore, strong performance benefits can be expected in cases in which the number of PRM or RRT nodes is large in comparison to the number of primitives in the geometric models used for collision detection.

To achieve resolution completeness guarantees, this thesis proposed several uniform deterministic sampling methods for some of the spaces arising in motion planning. We presented a general framework for performing deterministic uniform sampling over spheres and $SO(3)$, together with a particular sequence which extends the layered Sukharev grid sequence, which was designed for
the unit cube. We have tested the performance of the sequence in PRM-like motion planning algorithms, which demonstrated that this sequence is a useful alternative to random sampling. This is in addition to the advantages that this sequence has over random sampling, such as deterministic resolution completeness guarantees and the regular lattice structure. We have also developed and implemented a deterministic incremental grid sequence on $SO(3)$ that is highly uniform, can be efficiently generated, and divides the surface of $SO(3)$ into regions of equal volume. This sequence achieves the lowest metric distortions for grid neighbor edges on $SO(3)$ known to date.

To achieve efficient exploration in the ISS framework, we have considered the Voronoi biased exploration strategy of the RRTs and characterized the weaknesses of this strategy when the obstacles in the configuration space are not taken into account and/or the sampling region is inappropriately chosen. We then proposed a general framework for addressing these problems by considering a new sampling strategy based on the visibility region of the nodes in the tree. The new planner, called DD-RRT, adaptively controls the Voronoi bias of the nodes which results in a better exploration. Our experimental results show that the planner successfully solves the problems with constrained geometries as well as instances of the classical motion planning problem, sometimes by several magnitudes faster than the original RRTs. This suggests that the new planner is suited for a large set of motion planning problems. We also presented a general method for solving motion planning problems which involves constrained feasible configuration spaces. The approach builds a kd-tree representation of the explored part of the configuration space, which enables the RRT to use local information to rapidly explore on the feasible space. The experimental results suggest that it gives a uniform improvement over a large class of motion planning problems. The algorithm resulted in a slightly worse running time only on few out of a dozen experiments.

Future Directions

**Nearest neighbor searching.** Several directions are possible for future work on nearest neighbor search in the context of motion planning. The extension to different topological spaces can also be applied to other extensions of the kd-tree that have been used for nearest-neighbor searching, such as the relative neighbor graph [12] and balanced box-decomposition tree [14]. It was recently shown that it is possible to remove exponential dependencies in dimension from the nearest-neighbor problem [54, 92, 104]. Powerful new techniques are based on approximate distance-preserving embeddings of the points into lower-dimensional spaces. It remains to be seen whether these theoretical ideas will lead to practical algorithms, and whether they will yield superior performance for the dimensions
and number of points that are common in path planning problems. In path planning problems that involve differential constraints (nonholonomic and kinodynamic planning), it might be preferable to use complicated distance functions [70]. Such functions should be well-suited for a particular nonlinear system, and they might not even be symmetric. In these difficult cases, it remains open to find practical, efficient nearest-neighbor algorithms.

**Uniform deterministic sampling methods.** There is a number of ways to improve current sampling methods in motion planning. Nicely distributed grids are not yet developed for general \( n \)-spheres, \( n > 3 \). Implicitly defined manifolds, such as the ones arising from motion planning for closed linkages, are very hard to sample efficiently and uniformly. Such manifolds also arise as the conformation spaces of protein loops. In such cases, efficient parametrization is the bottleneck for developing sampling schemes. It might also be important to evaluate the general rate of convergence for motion planning methods using various sampling sequences.

**Guided exploration.** A next possible development would address the implementation of the adaptive tuning of the parameter used for kd-tree representing the sampling region. This can be done using the information history from the collision detector. Another important research direction is to address systems with differential constraints. The expressions for kinematic closure constraints are similar to differential constraints, which suggests that our approach may be beneficial for a broader class of motion planning problems. An extension of our method to other kinematics and dynamics constraints is an important future direction.

Kd-tree-based implementation of our algorithm provides good performance on up to 50-dimensional problems. For problems involving higher dimensions, other techniques need to be developed. An obvious future direction is to use a dimensionality reduction technique, which would project all of the space on significant dimensions, after which the kd-tree approach is applied. This could address the problem of motion planning for thousands of links.

**Motion planning.** In addition to the scope of this research there are many open problems in the area of motion planning. The motion planning problem for closed chains offers many open questions. Mathematical understanding of the arising spaces is needed to design efficient parametrization of the spaces for fast sampling. Novel dimensionality reduction methods are needed for high dimensional configuration spaces. The area of designing good motion primitives for motion planning is largely open [32]. Currently the use of the motion primitives is found only in several research works, in most of which they are picked manually [69, 75, 159]. Sampling issues arise in many configuration spaces, including implicitly defined algebraic variates. These spaces have the complication that the
uniformity criteria are also unknown.

Most of the complete algorithms designed for the general motion planning problem were never implemented due to their complexity. However, special instances of the problem (in which dimension is fixed) could have benefited from such implementations. Such an approach could be especially useful for planning under kinematic constraints, in which characterization of the feasible configuration space could be obtained from the combinatorial representation from such algorithms.
References


Author’s Biography

Ganna Yershova was born and grew up in Kharkiv, Ukraine. She graduated from Kharkiv National University in 1999 with Bachelor of Science degree in Applied Mathematics. Ganna obtained her Master of Science in Computer Science in 2003 from Iowa State University and her Doctor of Philosophy degree in Computer Science from University of Illinois in 2008. Her research interests include robotics, motion planning, computational geometry, and artificial intelligence.

In September 2008 Ganna joined Duke university as a postdoctoral associate to work on problems in structural computational biology.