박사학위논문
Ph.D. Dissertation

샘플링 기반 경로 생성 알고리즘과 구형 표현법의 혼성화

Hybridization of Sampling-based Motion Planning Algorithms and Spherical Representations

2019

김동혁 (金東赫 Kim, Dong-Hyuk)

한국과학기술원

Korea Advanced Institute of Science and Technology

박사학위논문

샘플링 기반 경로 생성 알고리즘과 구형 표현법의 혼성화

2019

김동혁

한국과학기술원

전산학부

샘플링 기반 경로 생성 알고리즘과 구형 표현법의 혼성화

김 동 혁

위 논문은 한국과학기술원 박사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2019년 5월 31일

심사위원장	윤 성 의	(인)
심 사 위 원	조 성 호	(인)
심 사 위 원	최 성 희	(인)
심 사 위 원	Martin Ziegler	(인)
심 사 위 원	김 영 준	(인)

Hybridization of Sampling-based Motion Planning Algorithms and Spherical Representations

Dong-Hyuk Kim

Advisor: Sung-Eui Yoon

A dissertation submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science

Daejeon, Korea
May 31, 2019

Approved by

Sung-Eui Yoon
Professor of School of Computing

The study was conducted in accordance with Code of Research Ethics[1].

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my advisor.

DCS
20145030

김동혁. 샘플링 기반 경로 생성 알고리즘과 구형 표현법의 혼성화 . 전산학부 . 65+iv 쪽. 지도교수: 윤성의. (영문 논문)
Dong-Hyuk Kim. Hybridization of Sampling-based Motion Planning Algorithms and Spherical Representations. School of Computing . 65+iv pages. Advisor: Sung-Eui Yoon. (Text in English)

초 록

샘플링 기반의 경로 생성 알고리즘은 확률적 완전성과 점진적 최적성을 보장하는 특징을 갖고 있어 로보틱스 분야에서 지난 수십년간 다양하고도 깊게 연구되어 온 분야 중 하나이다. 하지만 주어진 문제의 복잡도에 따라 증가하는 계산량으로 인해 최적해로의 수렴 속도를 높이기 위한 연구는 중요한 문제로 남겨져있다. 샘플링 기반 알고리즘의 주요 구성 요소는 **샘플링**, **충돌 검사**, **근접 이웃 탐색** 으로 이루어져있고, 이들을 통해 어떠한 형태의 랜덤 기하 그래프를 어떻게 구축하는지에 따라 전체 알고리즘의 성능이 크게 영향 받는다.

이에 따라 본 논문이 담고 있는 논제는 다음과 같다. 샘플링 기반 알고리즘의 최적해로의 수렴 속도를 개선하기 위해 3개의 주요 구성 요소를 분석하고 각각의 성능을 끌어올리기 위한 알고리즘 연구를 중심으로 한다. 특히 주어진 공간을 구형 표현법을 통해 다양한 형태로 이산화하고, 이의 효율적 활용 및 최적화 기반 경로 생성과의 혼성화를 통해 경로 생성 알고리즘의 전체적인 성능 향상을 추구하는 연구를 목표로 하고자 한다.

핵 심 낱 말 경로 생성, 구형 표현법, 점진적 최적성, 자유 공간 추정, 샘플링 기반

Abstract

Thanks to the probabilistic completeness and almost-sure asymptotic optimality, sampling-based motion planning algorithms have been widely and deeply studied for the past decades. It is, however, a well-known fact that the computational overhead of sampling-based algorithm can dramatically increase with respect to the complexity of a given problem. For this reason, improving the convergence speed toward the optimal solution in motion planning problem is a still-always open problem. In the aspect of computational importance, **Sampling**, **Collision Checking**, **Nearest Neighbor Search** have been considered major components in sampling-based approaches which can greatly affect the resulting random geometric graph.

To this context, the thesis of this dissertation is that in order to achieve the faster convergence speed toward the optimal solution we analyze the aforementioned three major components to improve the overall performance of the sampling-based planning, i.e., convergence speed toward the optimal solution. We particularly consider spherical representations to discretize space for various purpose and exploit those for biased-sampling, probabilistic collision checking, and adaptive sparse graph construction. We also hybridize sampling-based and optimized-based planning in conjunction with the spherical representations for faster convergence speed.

Keywords Motion Planning, Asymptotic Optimality, Free Space Approximation, Sampling-based planning, Hybridization

Contents

Contents i
List of Tables iii
List of Figures iv

Chapter 1. Introduction **1**

Chapter 2. Cloud RRT*: Sampling Cloud based RRT* **4**
2.1 BACKGROUND 4
2.1.1 Workspace Analysis 4
2.1.2 Optimality 4
2.2 OVERVIEW 5
2.2.1 Problem Definition 5
2.2.2 Review of RRT* and Its Convergence 5
2.2.3 Overview of Our Approach 6
2.3 ALGORITHM 6
2.3.1 Sampling Cloud 6
2.3.2 GVG-guided Initialization 7
2.3.3 Updating Sampling Cloud 8
2.3.4 Pruning Sampling Spheres 10
2.3.5 Cloud RRT* 11
2.4 RESULTS 11
2.4.1 Dubins Vehicle 12

Chapter 3. Dancing PRM*: Simultaneous Planning of Sampling and Optimization with Collision Free Space Approximation **18**
3.1 Introduction 18
3.2 Background 19
3.2.1 Sampling-based Motion Planning 19
3.2.2 Optimization-based Motion Planning 20
3.3 Algorithm 21
3.3.1 Overview 22
3.3.2 Configuration Free Space Approximation 23
3.3.3 Local trajectory optimization in Configuration Space 25
3.3.4 Batch Dancing Tree* 29
3.4 Experiments 31

3.4.1 Experiment setup . . . 31
3.4.2 Comparison against RABIT* . . . 31
3.4.3 Comparison in practical benchmarks . . . 33
3.4.4 Comparisons with varying batch sizes . . . 35
3.5 Analysis . . . 36
3.5.1 Almost-sure Asymptotic Optimality . . . 37
3.5.2 Computational Complexity . . . 37
3.5.3 Configuration free space approximation . . . 38
3.5.4 Radius compensation and dispersion . . . 40

Chapter 4. Volumetric Tree*: Adaptive Sparse Graph for Effective Exploration of Homotopy Classes 42
4.1 Introduction . . . 42
4.2 Algorithmic Background . . . 44
4.2.1 Problem Definition . . . 44
4.2.2 Sampling-based Motion Planning . . . 44
4.3 Algorithm . . . 45
4.3.1 Motivations . . . 45
4.3.2 Adaptive Sparse Tree Construction . . . 46
4.3.3 Path Optimization . . . 47
4.3.4 Shortest Path Computation with Dropout . . . 49
4.4 Experiment . . . 50

Chapter 5. Summary and Conclusion 58

Bibliography 59

Acknowledgments in Korean 63

Curriculum Vitae in Korean 64

List of Tables

2.1 Various statistics averaged over 100 trials with the running time budget of 5 seconds. Numbers in parentheses are standard deviations. 14
2.2 Statistics related to our GVG-guided initialization. 15

3.1 Notation summary table. 21

4.1 Statistics of experiment results. $|V|$ and $|E|$ are the cardinality of the vertex set V and edge set E, respectively. $|VC|$ and $|EC|$ stand for the number of vertex and edge collision checking, and $T(\cdot)$ is a computation time ratio of each component: CC = Collision Checking, NN = Nearest Neighbor search, SP = Shortest Path computation in DSPT and OPT = OPTimization. Note that the collision checking time during the optimization is also included in $T(OPT)$ and the number of NN-query is identical to $|VC|$ in volumetric tree*. In 6D manipulation and 7D Hubo arm planning problem, the result of RRT* is not reported due to a high ratio ($> 90\%$) of unsuccessful attempts. 52

List of Figures

1.1 An exemplar motion planning problem (left) and a random geometric graph constructed by a sampling-based approach (right). In the right figure, each black dot indicates a collision-free configuration and red polyominos depicts a corresponding robot shape in the workspace along the solution path connecting two given configurations, x_{init} and x_{goal}. 2

1.2 A hybridization of sampling-based motion planning and spherical representation. We utilize both the graph constructed by sampling-based motion planning and spherical representation to handle given space efficiently for various purpose. In the left figure, they are associated with each other and work complementarily according to their definition. The right figure shows two solution path in different homotopy classes where each class is colored by red and blue respectively. A sequence of hyperspheres along the path can be used for a probabilistic homotopy class identification. . . . 3

2.1 The left figure shows sampling spheres computed by our GVG-based initialization in a 2D example. Blue and black lines are obstacles and Voronoi edges, respectively. The right figure shows updated sampling spheres with a computed path between the initial position (shown in the red box) and the goal (the green box); see the pdf file for better visual quality. 9

2.2 Initial state of sampling cloud constructed over Fig. 2.3(a) (left) and the one after pruning (right). We can prune out unpromising sampling spheres using triangle inequality with respect to the cost of the best-so-far solution path, which allows the planner to focus on the promising regions. . . . 10

2.3 These figures show two 2D environments for a kinematic car. Red and green squares indicate initial and goal positions, respectively. The car is drawn with its two circles of the minimum turning radius at its initial position. Computed near-optimal paths are shown in red curves. Obstacles in the scene of (c) are represented in blue dots, which mimic real-world data. 12

2.4 This figure shows a graph of costs of solutions as a function of computation time with different methods. Our method shows the best quality over other methods that are even combined with GVG-based sampling. 16

2.5 Computation time to reach a specific cost of the solution as a function of target costs. 17

3.1 These figures show a 2D manipulation problem with two joints α and β in a planar space. The left figure shows how the robot is represented for working with workspace obstacle information (left) and the signed distance field of the environment (right). In the left, a robot arm is simplified by a set of body points $b \in B$ (small squares), each of which is a center of swept-sphere volume (red circles). In the right, potentials against the obstacle are computed by the proximity of those in the workspace (black arrow). 22

3.2 The left figure shows the configuration space of Fig. 3.1. If we can represent the configuration space in a usable form, we can directly compute the obstacle potential and its gradient without any other conversions between the workspace and configuration space. The right figure shows the corresponding movement in the workspace between two configurations (α,β) and (α', β'). 23

3.3 The left is a visualization of $\widetilde{\mathbb{X}}_{free}$, regions covered by a set of light blue circles in 2D; for simplicity, we show the merged region of circles, instead of visualizing each circle. Each configuration $q \in V$ is associated with an approximate collision-free hypersphere in $\mathbb{X}$. Their radii are trimmed by *witness* (red cross symbol) which is a configuration in $\mathbb{X}_{obs}$ found during a local planning (dotted black segment on the left side) or a sample q_{sample} (the right side in the same figure). The right figure shows an example of local optimization for a trajectory ξ. Black arrows show the gradient of obstacle potential computed with our approximate configuration space and the red curved segment shows an optimized trajectory. Each red dot indicates an intermediate configuration $\xi(t)$ on the discretized ξ. 24

3.4 An abstraction of the proposed algorithm. The dotted boxes indicate proposed components in conjunction with a generic sampling-based motion planning algorithm with the solid boxes. . . . 25

3.5 A sequence of witness propagation for a sample configuration q_{sample} and its near neighbors within the light-blue circle with a radius of $\gamma \left(\frac{\log |V|}{|V|} \right)^{1/d}$. Pairs of configuration and its former witness are connected by red segments, and blue dotted lines indicate witness newly updated. q_{sample} inherits its witness $w_{q_{sample}}$ first (left) from its near neighbors. It is then propagated to other neighbors in the circle to ensure that near neighbors have a witness as the closest empirical collision. 26

3.6 Performance comparison over computation time for different algorithms in synthetic benchmarks. Results are averaged over 30 trials. In $\mathbb{R}^8$, RABIT + DF (Distance Field)s are not reported since the construction of the distance field with a reasonable resolution is intractable in a high-dimensional space. 32

3.7 Computation time breakdown of the proposed algorithm DancingPRM* measured in our benchmarks (Fig. 3.8). Each abbreviation in legend stands for OPTimization (OPT), Shortest Path tree update (SP), Nearest Neighbor search (NN), and Collision Detection (CD). Note that the computation time for $\widetilde{\mathbb{X}}_{free}$ construction is negligible ($< 1\%$) for all of three benchmarks. 32

3.8 Performance comparison over computation time (left) and the given environment (right). The plots show the performance of asymptotic optimal planners tested in our benchmark. The horizontal black dotted line shows the best solution cost achieved by algorithms without our local trajectory optimization. Figures on the right side are visualization of benchmarks. Results are averaged over 30 attempts, and RRT* is not reported for Fig. 3.8(c) and 3.8(e) due to a high performance gap. . 34

3.9 Performance comparison with varying the number of samples per batch in BDT*. The results are averaged over 30 attempts and tested in Fig. 3.8(b). A higher number of samples per batch provides a better convergence speed at the expense of anytime property. 35

3.10 Computation time breakdown of Batch Dancing Tree* tested in Fig. 3.8(b). The result shows the distribution of computation time spent on each component over varying batch size. Note that the majority of *Etc.* is a graph expansion based on LPA*, which corresponds to *SP* in Fig. 3.7. The overhead for the configuration free space approximation is measured $< 3\%$ in this experiment. . . 36

3.11 Mean squared errors measured at intermediate configurations during the local optimization by both our configuration free space approximation and the ball-tree algorithm with different initial R_0 values. Ours shows a monotonic decreasing result, while the ball-tree with fixed initial radii tends to converge at a particular value depending on the initial value. These plots use a logarithmic scale for the y-axis and a linear scale for the x-axis. 39

3.12 A concept of dispersion for the sample set V^* and the radius compensation. V^* consists of all sampled configurations in both $\mathbb{X}_{free}$ (blue) and $\mathbb{X}_{obs}$ (red) regardless of feasibility, and including samples for edge collision checkings (dotted segments). The radius of the red dotted circle on the right side stands for a dispersion of V^*, which is used as an approximate sparsity of V^*. The radius of the left circle represents a compensated radius (grey-dotted smaller circle), i.e., $\omega(|V^*|) \cdot r_q$, which is shrunk from the original radius r_q (grey-solid). 40

4.1 A heatmap-style visualization of the vertex set V, constructed by a conventional planner (top, $|V| = 14384$) and that of volumetric tree* (bottom, $|V| = 540$) in the same time budget. We can observe the volumetric tree* constructs a sparse graph, while capturing the samples around narrow passages or boundaries. The vertices close to the obstacles are encoded red; otherwise blue. . . . 43

4.2 The left figure illustrates two solution paths, shown red and blue solid lines, in two different homotopy classes with their local optimal paths (dotted). The right figure shows a solution path (solid black) covered by a sequence of collision-free balls and paths homotopic to the solution path (red and blue). These observations suggest that such a coarse-grained graph can be a sufficient representation if we can optimize each path toward the local optimum. 45

4.3 Naïve shortest path computation can miss to explore a solution path even homotopic to the optimal solution path (the blue dotted one), due to a sparse graph structure. We address this issue by using the dropout of vertices in solution paths observed during the execution. The dotted-filled ball in the right figure stands for an excluded vertex, $v_{dropout}$. The search tree constructed without $V_{dropout}$ allows our approach to find other solution paths (the blue dotted one) that can be homotopic to the optimal solution, the solid black line. 49

4.4 A 2-DoF mobile robot planning problem in a conference room with narrow passages under the chairs, resulting in difficult-to-sample homotopies and surrounding wide-open areas. x_{init} and x_{goal} are depicted in the green and red boxes, respectively. 53

4.5 A 6-DoF manipulation planning problem. x_{goal} (in the red box) is a pose of grasping the cup in the middle of the table, avoiding the other cups and the ceiling from x_{init} (in the green box). . . . 54

4.6 A 7-DoF Hubo robot arm planning problem. The goal is to grasp the target bottle (inside the red rectangle) in the middle of other objects on the shelf. 55

4.7 2D synthetic benchmark with 10 narrow gaps located in the middle of the hyper-cube. In the left figure, green squares indicate the obstacles and the black lines are the optimal solution path. The right figure shows intermediate configurations during (colored differently) the optimization iterations. The dots indicate vertices of our sparse graph, colored according to their radii (red for smaller values). 56

4.8 Performance comparison over computation time for different algorithms in synthetic benchmarks. 57

Chapter 1. Introduction

Path and Motion Planning has been considered one of the most fundamental research topic in the field of the autonomous system which is capable of planning and accomplishing tasks in consideration of the given constraints such as collision avoidance. The first classical formulation of the motion planning problem is also known as "Piano Mover's problem" [1] suggested by J. T. Schwartz and m. Sharir. In their original work, the piano mover's problem is formulated as:

> "Given a body B and a region bounded by a collection of "walls", either find a continuous motion connecting two given positions and orientations of B during which B avoids collision with the walls, or else establish that no such motion exists."

In recent literature and studies [2, 3], the formulation of motion planning problem has been generalized as follow.

MOTION PLANNING PROBLEM

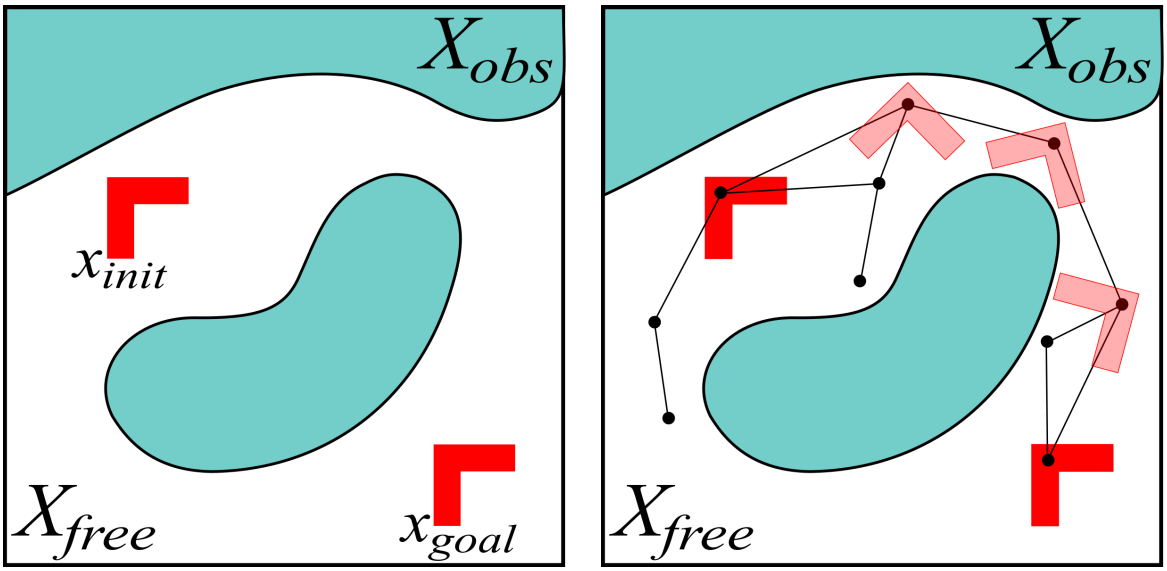
Input:

(i)	A Workspace $\mathcal{W}$ ($= \mathbb{R}^n$).
(ii)	An obstacle region $\mathcal{W}_{obs} \subset \mathcal{W}$.
(iii)	A robot defined in $\mathcal{W}$. The robot body B can be a rigid body, an assembly of joints (e.g., robot arm manipulator), or even a set of multiple separated bodies.
(iv)	The configuration space $\mathcal{X}$ where $\mathcal{X}_{obs} \subset \mathcal{X}$ is a set of all configurations with $B \cap \mathcal{W}_{obs} \neq \emptyset$, and $\mathcal{X}_{free}$ is its complement set.
(v)	An initial configuration $x_{init} \in \mathcal{X}_{free}$ and a goal configuration $x_{goal} \in \mathcal{X}_{free}$.
Objective:	Compute a sequence of configurations, $\tau : [0,1] \rightarrow \mathcal{X}_{free}$, such that $\tau(0) = x_{init}$ and $\tau(1) = x_{goal}$.

Fig. 1.1 depicts an instance of motion planning problem and a solution path (afterimages of the red polyomino on the graph)found by a sampling-based approach in an illustrative way.

There have been suggested various algorithms to efficiently solve the motion planning problem in terms of the computation time. Among them, sampling-based approaches have attracted substantial interests for past several decades thanks to its probabilistic completeness and broad applicability. Some prominent examples are RRT (Rapidly-exploring Random Tree) [4] and PRM (Probabilistic Roadmap Method) [5], which can be viewed as a random geometric graph construction in free space. In regard to the asymptotic optimal motion planning, Karaman et al. presented RRT*, PRM*, and RRG (Rapidly-exploring Random Graph) [6], which guarantee almost-sure asymptotic optimality. These optimal variants successfully opened a new research area in motion planning by providing a theoretical foundation, and have been applied to numerous follow-up works for real problems such as anytime [7, 8], kinodynamic [9, 10], and biased/adaptive sampling techniques [11, 12, 13] for faster convergence.

The convergence speed toward the optimal solution, however, remains a major task for optimal motion planning problem. Especially in high dimensional space, a planner requires a tremendous amount of time and memory to get closer to the optimal solution making sampling-based approaches impractical. To this end, the thesis of this dissertation is to achieve a better convergence speed toward the optimal solution by analyzing three major components in sampling-based approach; Sampling, Collision Checking, and Nearest Neighbor Search with spherical



(a) (b)

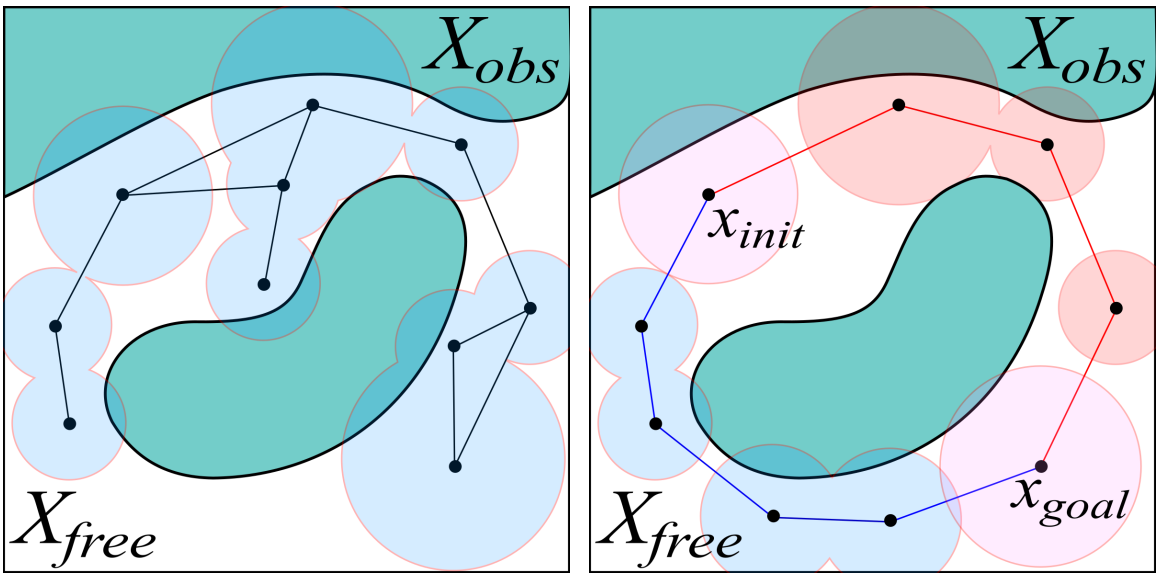
Figure 1.1: An exemplar motion planning problem (left) and a random geometric graph constructed by a sampling-based approach (right). In the right figure, each black dot indicates a collision-free configuration and red polyominos depicts a corresponding robot shape in the workspace along the solution path connecting two given configurations, x_{init} and x_{goal}.

representations. The hypersphere is basically the simplest geometrical object in terms of its geometric shape, inclusion and exclusion test, proximity computation, and scalability. Unlike the workspace, the corresponding configuration space tends to have a higher dimension in motion planning; thus scalable representations followed by scalable operations can significantly affect the complexity of both time and memory.

Fig. 1.2 shows some examples of hybridization between sampling-based motion planning and spherical representation. In the proposed algorithms, the spherical representations are not just used as data structures separated from a sampling-based motion planning. We first associate the resulting graph of sampling-based motion planning with spherical representation for easy manipulation. As observed in Fig. 1.2(a), the resulting graph can be viewed as a proximity graph where referencing local near neighbor can be done through the edges. Since the constructed graph itself represents a connection of free space, we exploit the properties of sampling-based motion planning for our spherical representation manipulation as well. Fig. 1.2(b) shows a different concept of spherical representation. The solution paths connecting x_{init} and x_{goal} are in two different homotopy class where each path is associated with a sequence of hyperspheres. These representations can be used for a homotopy class identification in order to guide the sampling toward a preferred pattern or region for efficient and selective exploration on the given space. We will also discuss some useful algorithmic methods to initialize, compute or approximate the shape of spherical representations in subsequent chapters.

In this dissertation, we use spherical representations to represent a sampling region for biased-sampling (Cloud RRT*, Chapter 2), approximate collision-free region for probabilistic collision checking (Dancing PRM*, Chapter 3) and a volumetric configuration for sparse graph construction to reduce the asymptotic complexity of nearest neighbor search (Volumetric Tree*, Chapter 4). On top of that, we consider hybridization of sampling-based and optimization-based planning in Chapter 3 and 4 to leverage the synergy between them.

The detail of each algorithm will be discussed throughout the dissertation.



(a) (b)

Figure 1.2: A hybridization of sampling-based motion planning and spherical representation. We utilize both the graph constructed by sampling-based motion planning and spherical representation to handle given space efficiently for various purpose. In the left figure, they are associated with each other and work complementarily according to their definition. The right figure shows two solution path in different homotopy classes where each class is colored by red and blue respectively. A sequence of hyperspheres along the path can be used for a probabilistic homotopy class identification.

Chapter 2. Cloud RRT*: Sampling Cloud based RRT*

Most early works for sampling-based techniques focused on finding the existence of a solution for their problems. Some of recent works [14, 11, 15] introduced different sampling heuristics to accelerate the convergence to the optimal solution, while maintaining the optimality guarantee. These techniques proposed explicit or implicit ways to approximate or detect promising areas, where the optimal solution is likely to exist, and biased their sampling patterns. While these prior approaches provide improvement on the convergence speed in some extent, we have found that they can be less efficient to find better solutions in other homotopy classes (Sec. 2.4). Especially for environments with complex configurations of obstacles and narrow passages, these prior techniques may not work well.

2.1 BACKGROUND

In this section we discuss prior works on sampling based motion planning, and techniques to estimate promising area and speedup the convergence toward the optimal.

2.1.1 Workspace Analysis

Workspace analysis [16, 17, 18] is mainly used as a process of analyzing a given environment to relieve hardness of finding narrow passages and to generate a safe path to the goal with enough clearance to the nearest obstacles.

The narrow passage problem still remains a challenging issue in the motion planning field, and many algorithms have been proposed to address this issue. Since a sampling policy considering the surrounding environment can generate a collision-free path more effectively by avoiding oversampling in undesired regions, many techniques have been proposed to utilize various information derived from the workspace. At a high level, they include filtering samples to explore important regions more [19, 20, 21], utilizing free space information [22], and retraction-based approaches [23] that generate more samples on the boundary of the obstacle space.

To perform various workspace analysis and use its derived information many approaches have used diverse decomposition methods such as Generalized Voronoi Graph (GVG) [17, 16], uniform grid [24], and quad/octree [18]. After constructing such data structures planners use biased sampling towards the medial axis that has the maximum clearance to the obstacles or other desired directions guided by decomposition. As another technique of decomposition, there is a sphere expansion [25] constructing a tree, where each node represents a sphere-shaped free space. A set of these spheres can be considered as an approximation of continuously connected free space volume, and few works [26, 27] use this technique to analyze a given workspace.

For initializing the sampling cloud in our method we take advantage of GVG, whose vertices and edges are associated with distance to the nearest obstacles to estimate collision-free space and promising area.

2.1.2 Optimality

Recently optimizing paths in terms of various measures including path lengths has been receiving growing attention because of its usability in various motion planning problems or theoretical interest. Some of them proposed a set of heuristics to optimize the path lengths or a principled approach for showing probabilistic optimality.

These algorithms can be roughly categorized into two large groups. One group is based on exploitation, i.e. local biasing techniques on a specific homotopy class of solutions such as sampling nearby current best solutions or shortcutting [28, 11, 15]. Meanwhile, [29] presented an on-line obstacle avoidance algorithm that divides the entire path planning problem into a sequence of simpler problems of avoiding one obstacle for each step. In this way they could generate a near optimal path very quickly under some assumptions by reducing the complexity of the entire problem.

Another group is based on exploration to find a new homotopy class. Techniques in this group typically adopt workspace/geometry analysis [30, 31] to get more information from the given space. It is well known that there is a trade-off relationship between exploitation and exploration [30], and the balance control is a key to address the optimality issue [11].

In our method we design our sampling update method based on milestones for achieving a rapid convergence toward a local optimum, while ensuring the possibility of finding the global optimum by maintaining sampling probabilities in other homotopy classes.

2.2 OVERVIEW

We first define our motion planning problem, followed by a brief review of RRT* and its problems. We then give an overview of our method.

2.2.1 Problem Definition

Let $\mathbf{X}$ and $\mathbf{U}$ be the state space and control space, respectively. Let $x_0 \in \mathbf{X}$ the *initial state*. We then have the following dynamical system describing the relationship between control and state:

$$\dot{x}(t) = f(x(t), u(t)), \;\; x(0) = x_0, \tag{2.1}$$

where $x(t) \in \mathbf{X}$ and $u(t) \in \mathbf{U}$ are the state of the robot and a control input at time t, respectively. f is a continuously differentiable function with respect to its variables. Let $\mathbf{X}_{obs}, \mathbf{X}_{goal} \subset \mathbf{X}$ to represent *obstacle* and *goal* regions, respectively. The *obstacle-free* region then can be denoted by $\mathbf{X}_{free} = \mathbf{X} \backslash \mathbf{X}_{obs}$.

A path that connects x_0 to x_n consists of a sequence of control inputs $u_0, u_1, \ldots, u_n$ and corresponding state x_i can be obtained sequentially by the integration of f. Let $g : \tau(\mathbf{t}) \to \mathbb{R}$ be a cost function that associates a non-zero cost, where $\tau : [0 : T] \to X_{free}$ indicates a continuous measurable trajectory passing $x(t)$. An example of $g(\tau(t))$ can be simply a distance between two states for a rigid body.

Optimal motion planning. Given the dynamical system described in Eq. 2.1, the motion planning problem is to find a continuous path, $\tau : [0, T] \to \mathbf{X}_{free}$ such that $\tau(0) = x_0$ and $\tau(T) \in \mathbf{X}_{goal}$, while satisfying the control constraints or corresponding $u(t), \forall t \in [0, T]$. For an optimal motion planning, there is an additional constraint to minimize the integral of the cost function over the entire path, $\int_0^T g(\tau(t))dt$.

In this chapter we focus on the optimal motion planning for a rigid body. Before we present our method, we first give a brief review on RRT*, designed for the optimal motion planning with single queries.

2.2.2 Review of RRT* and Its Convergence

RRT* is an incremental sampling based motion planning algorithm, and unlike the original RRT it asymptotically converges to the optimal solution. Given a randomly sampled configuration, the original RRT finds its nearest configuration among existing nodes in the RRT to determine its parent. If a feasible trajectory connecting

from the parent to the sample is found by a local planner, it then inserts an edge that connects the sample and the parent into the RRT. Once the relationship between the pair of sample and parent nodes is determined, it is fixed throughout the entire execution.

RRT*, however, takes into account a cost from the initial state to reach each parent candidate, which is located within a circle with a certain radius to determine the optimal parent in terms of cost. Furthermore, by using the *rewire* routine, it can compute the optimal solution by finding shorter paths between existing configurations, while growing the RRT.

RRT* provides an optimality guarantee for the solution as we have an infinite amount of time. In practice, how quickly it converges toward the optimal solution is a matter of concern. Especially for mobile robots or vehicles, it is critical to compute high-quality solutions in an efficient manner.

Recent prior techniques proposed local biasing techniques [28, 11, 15] for providing improvement on the convergence speed to some extent. We have found that these approaches are good at refining a path in one homotopy class, but have less tendency on discovering solutions in other homotopy classes. As a result, in a complex work space where various homotopy classes exist and the optimum lies on a specific one with narrow passage, it might not provide noticeable improvement.

2.2.3 Overview of Our Approach

We present a novel biased sampling technique to efficiently compute a high-quality collision-free path, while maintaining the asymptotic convergence to the optimal solution. We aim to allocate a high sampling weight for a region that is promising to compute the optimal solution. To achieve the goal we first decompose the C-space as a set of spheres with varying radii, *sampling cloud*, denoted by $\mathcal{S}$.

To construct initial sampling cloud and compute an initial path in an efficient manner, we construct Generalized Voronoi Graph (GVG) as a geometric analysis of the given work space.

Whenever we found a better path during the execution of planner, we define a *milestone*, which contains configurations from the current best path over all the prior paths. We exploit the milestone to guide the current sampling cloud for refining the current best solution. For this we generate additional new spheres to cover configurations on the milestone.

Meanwhile we sample other regions with existing sampling spheres to identify a better solution that has a different homotopy to that of the current best path.

2.3 ALGORITHM

In this section we discuss each component of our method.

2.3.1 Sampling Cloud

We use our sampling cloud for representing our sampling distribution on the C-space. Since we need to use it for every sample generating, it should be very efficient and flexible for representing different sampling distributions. Given these requirements we decide to represent our sampling cloud $\mathcal{S}$ with a set of spheres.

Each sphere of the sampling cloud contains a portion of the C-space, which is projected to a collision-free spherical region in the workspace.

Specifically, a subset, $\mathbf{X}_s$, of C-space associated with a sphere s is defined as follow:

$$\mathbf{X}_s := \{x \mid Proj(x) \in s \wedge x \in \mathbf{X}\}, \tag{2.2}$$

where $Proj(\cdot)$ is a projection function from the C-space to the workspace. We utilize $\mathbf{X}_s$ to generate samples given the sampling sphere s.

A sampling sphere, s, is associated with its center position and radius defined in the workspace; the radius of s is denoted as r_s. Additionally the sphere s is associated with an importance value, i_s, and an orientation range for each orientation part in the C-space. The importance value i_s of a sampling sphere s represents the sampling probability of s among all the spheres of $\mathcal{S}$. The importance value is used at the sampling phase of our method. i_s is also initialized proportional to the volume of sphere s in the workspace. The orientation range is defined by two radian value: a main orientation, ϕ_s, and its deviation value, θ_s. We then generate samples in the range of $[\phi_s - \theta_s, \phi_s + \theta_s]$.

In our method we generate random samples in the C-space according to $\mathcal{S}$ instead of the C-space $\mathbf{X}$. Specifically we first choose a sphere s among $\mathcal{S}$ according to their importance values i_s. We then generate a configuration covered by the sphere s in a uniform manner. In particular we generate two dimensional positions within the spherical region of the sphere s and then generate an orientation with the orientation range associated with s.

Given this sampling procedure we can easily adjust our sampling distribution by adding or deleting spheres to/from the sampling cloud. Multiple spheres can have overlap. In this case the overlapped region has a higher probability to be generated compared to other non-overlapped regions.

2.3.2 GVG-guided Initialization

We propose a GVG-guided initialization for our sampling cloud.

In the case of two dimensional problem, GVG has a nice property that the global optimal solution is always homotopic to one of the path in GVG, if it exists [32]. While GVG does not capture all the connectivity of the free-space with dimensions higher than two, it can still provide a reasonable view on the connectivity.

We therefore use GVG to guide an initial distribution of our sampling spheres and cover important regions such as narrow passages for computing collision-free paths.

Many different construction techniques for GVG have been available for two and three dimensional cases [33, 34]. Some of them can be performed efficiently on GPUs for two and three dimensional workspaces [34].

For our problem we assume that input environments are two dimensional and consist of points, line segments, and their combinations.

The pseudo code of our GVG-based initialization is given in Algorithm 1. Spheres constructed by GVG in a simple scene are depicted in Fig 2.1-(a).

We first compute Voronoi vertices and edges given obstacle information. This is shown in the line number 1 in Alg. 1, which is denoted by [1:Alg. 1] hereafter. These Voronoi vertices and edges are shown in black curves/lines in Fig. 2.1.

As a position of our first sampling sphere, we find a point, v_{init}, on GVG that is visible from the starting position of a robot. For computing the visible point, we generate random lines from the starting position and perform collision detection between the lines and obstacles. The main reason why we find the visible point is to avoid computing a point of GVG that is inside an obstacle.

Once the first sphere is located at the visible point, we then compute its radius such that it contains a largest collision-free workspace. The maximum radius of the first sphere, r_{init}, is computed by computing the nearest neighbor among obstacles from the visible point. This process is performed in the function of *DistanceToClosestObstacle* [4:Alg. 1]. To construct other spheres we put our initial sphere in a queue.

We continue our GVG-based initialization by fetching a sphere s from the queue. We then construct another sphere to cover other Voronoi vertices and edges.

Algorithm 1: GVG-BASED INITIALIZATION

Input: Obs, a set of obstacles

Output: S, a set of collision-free spheres

$(V_{GVG}, E_{GVG}) \leftarrow ConstructVoronoiGraph(Obs)$
$\mathcal{S} \leftarrow \emptyset$; $Q \leftarrow \emptyset$
$v_{init} \leftarrow FindVisiblePoint(V_{GVG}, E_{GVG})$
$r_{init} \leftarrow DistanceToClosestObstacle(v_{init}, Obs)$
$Q.push((v_{init}, r_{init}))$ // push a sphere with v_{init} and r_{init}
while Q *isNotEmpty* **do**
    $(v, r) \leftarrow Q.pop()$ // fetch a sphere
    $V \leftarrow ConstructNeighborSpheres((v, r), V_{GVG}, E_{GVG})$
    for ***each*** $v \in V$ *// per each intersected vertex* **do**
        if $IsNotContainedInOtherSpheres(v, \mathcal{S})$ **then**
            $r \leftarrow DistanceToClosestObstacle(v, Obs)$
            $\mathcal{S} \leftarrow \mathcal{S} \cup \{v, r\}$
            $Q.push((v, r))$
return $\mathcal{S}$

Among possible locations we incrementally compute candidate positions for a new sphere in a way that the new sphere has an overlap with the current sphere s. For this process we compute intersection points between the sphere s and Voronoi edges of GVG

[8: Alg. 1]. We check whether these intersection points are contained in existing spheres [10: Alg. 1]. If they are not contained, we construct spheres centered at those locations in the same manner to that of constructing our first sphere. We also push constructed spheres into the queue. We continue this process until the queue has no more spheres.

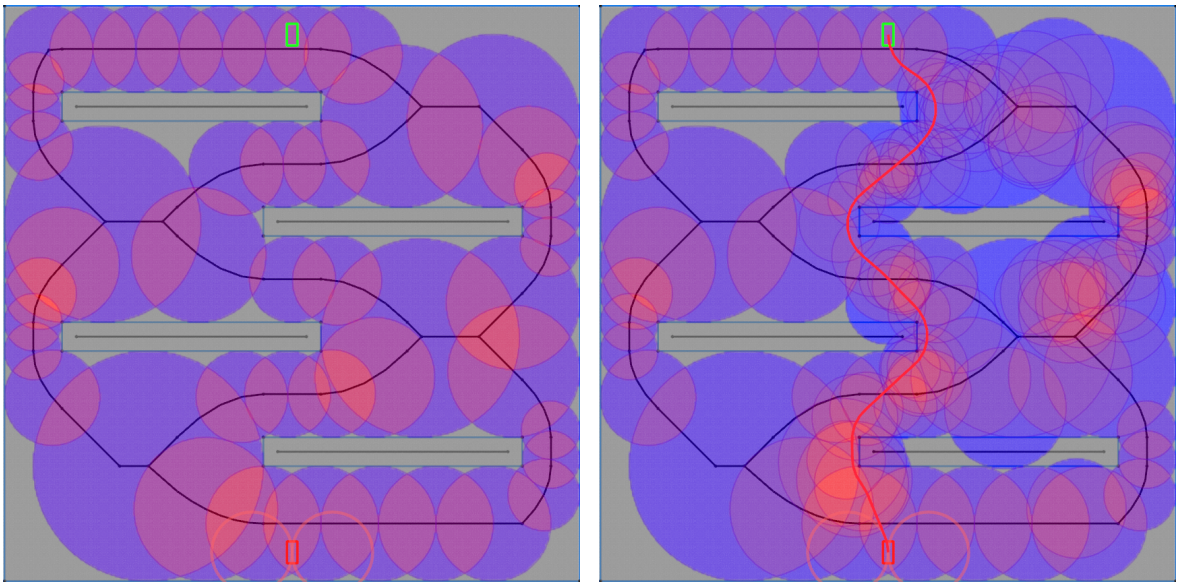
Fig. 2.1-(a) shows our initial sampling cloud in a simple scene. In this figure regions shown in red colors have a higher probability to be sampled than those with blue ones.

2.3.3 Updating Sampling Cloud

As we exploit and refine the current best path or explore unvisited regions, we can find a better path with a smaller cost than the current one. We update our sampling distribution to reflect the newly identified shorter path.

Given prior and new paths we define a *milestone* to contain a set of new configurations from the new path, but that are not included in all the prior old paths. As a result the milestone of the new path represents a new path segments that were not discovered in prior paths. At a high level we would like to generate more samples near regions around the milestone, while maintaining sampling probability for regions that are far away from the milestone. In summary, we aim to preserve the global sampling distribution, while focusing more on the milestone locally.

For each configuration of the milestone, it is guaranteed that the configuration is contained in at least one or multiple sampling spheres, denoted by $\mathcal{S}_c$; the subscript c of $\mathcal{S}_c$ denotes Containing spheres. This is because the configuration is generated from the current sampling cloud. We generate a new sphere, n, centered for each configuration of the milestone to locally sample more on regions related to the milestone. The new sphere n should be more localized than spheres of $\mathcal{S}_c$. We therefore compute the radius, r_n, of n by reducing radii of $\mathcal{S}_c$ by a factor



(a) An initial sampling cloud (b) An updated one

Figure 2.1: The left figure shows sampling spheres computed by our GVG-based initialization in a 2D example. Blue and black lines are obstacles and Voronoi edges, respectively. The right figure shows updated sampling spheres with a computed path between the initial position (shown in the red box) and the goal (the green box); see the pdf file for better visual quality.

of $\alpha \in (0,1)$. We then set the importance, i_n, of the new sphere to be computed relatively to importances of $\mathcal{S}_c$ based to their radii. In summary, r_n and i_n of the new sphere n are defined as the following:

$$r_n = \frac{1}{|\mathcal{S}_c|} \sum_{s \in \mathcal{S}_c} r_s \cdot \alpha, \tag{2.3}$$

$$i_n = \frac{1}{|\mathcal{S}_c|} \sum_{s \in \mathcal{S}_c} \frac{i_s \cdot r_n}{r_s + r_n}. \tag{2.4}$$

We then reduce the importance of $\mathcal{S}_c$ by Δi_s. Since we want to maintain the sum of all the importance values before and after the update operation, we set the sum of Δi_s to be equal to the added importance of the newly generated spheres, i.e. i_n, and thus have the following equation:

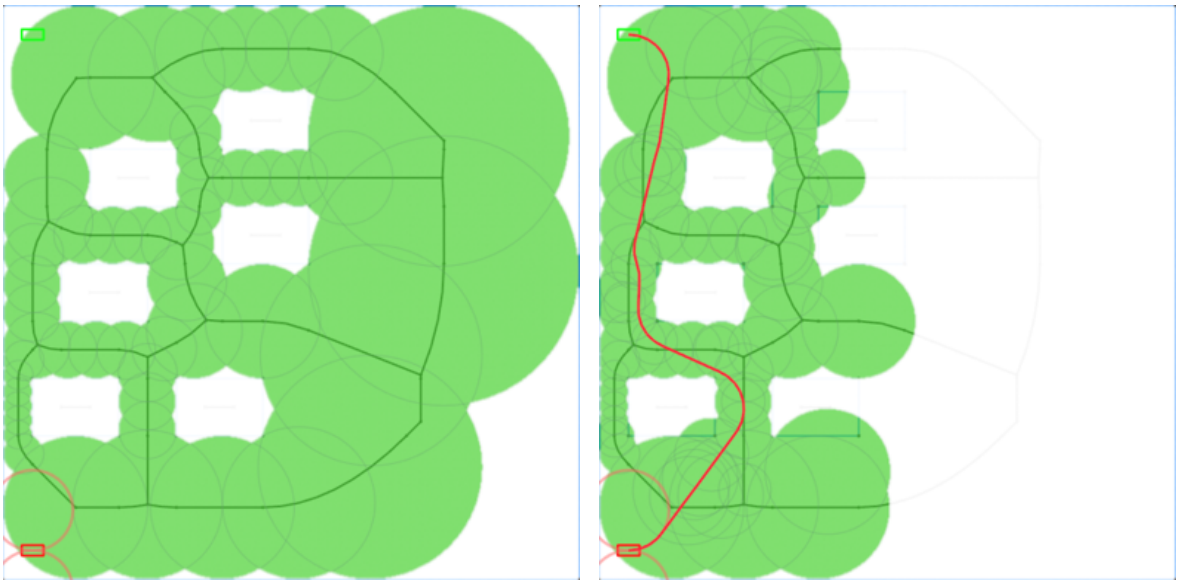
$$\Delta i_s = \frac{i_s \cdot r_n}{(r_s + r_n) \cdot |\mathcal{S}_c|}, \forall i_s \in \mathcal{S}_c. \tag{2.5}$$

One can easily see that the sum of Δi_s is equal to i_n.

The main orientation ϕ_n of n follows the orientation of the configuration in the milestone. The deviation value θ_s is calculated in the same manner to the radius as follows:

$$\theta_n = \frac{1}{|\mathcal{S}_c|} \sum_{s \in \mathcal{S}_c} \theta_s \cdot \alpha. \tag{2.6}$$

This process results in increasing the sampling probability more locally on the milestone based on the new sphere n, but reducing on its neighboring regions covered by $\mathcal{S}_c$, while maintaining the global sampling distribution to other regions (Fig. 2.1-(b)). Because of maintaining the global sampling distribution in other homotopy classes, we can effectively explore such regions, while focusing on the current best homotopy class containing the milestone.



(a) Initial (b) Pruned

Figure 2.2: Initial state of sampling cloud constructed over Fig. 2.3(a) (left) and the one after pruning (right). We can prune out unpromising sampling spheres using triangle inequality with respect to the cost of the best-so-far solution path, which allows the planner to focus on the promising regions.

Note that a milestone is designed to include newly identified configurations compared to those included in all the prior milestones. Alternatively, a milestone may include all the configurations from those prior milestones. We have found that this approach results in excessive biasing toward a local optimum.

2.3.4 Pruning Sampling Spheres

As we identify better solutions, we update our milestone and sampling cloud to cover regions localized near the milestone. In this process we generate additional spheres while maintaining the global sampling distribution. Some sampling spheres that seem promising for computing better solutions, however, turn out to be ineffective later during the optimization process.

In order to design more effective sampling process, we prune sampling spheres among $\mathcal{S}$ that cannot provide a better solution than the current one as shown in Fig. 2.2. For this purpose we propose a simple pruning method.

Given a sphere, we compute a line based path starting from the start position to the goal by passing a point in the sphere. When we identify a point that minimizes the length of the line-based trajectory, the length of the trajectory serves as a lower bound of a solution that any samples of the sphere can achieve. When the length is bigger that the length of the current best solution, we can conservatively prune the sphere, since it cannot contribute to optimize the path.

The point in the sphere minimizing the line-based trajectory is the intersection point between the sphere and the shortest line generated from the center of the sphere to another line segment consisting of the start and goal positions.

We perform our pruning method when a sphere is chosen from our sampling cloud. Since we do not change its position, all the computation of our pruning method is performed only one time and recorded. We then check its lower bound with the length of the current best path. As a result, its computation overhead is negligible.

Algorithm 2: Cloud RRT*

Input: A sampling cloud $\mathcal{S}$, an init. configuration, q_{init}, and a goal region, Q_{goal}

Output: A random tree, $\mathcal{T}$, and a solution path, Q_{sol}

1 $\mathcal{T} \leftarrow \{q_{init}\}$ and $Q_{sol} \leftarrow \emptyset$
2 **while *not*** *termination conditions are satisfied* **do**
3 s$\leftarrow SampleSphere(\mathcal{S})$
4 $q_s \leftarrow Sample(\text{s})$
5 $q_n \leftarrow Nearest(q_s)$
6 $(q_{new}, u_{new}) \leftarrow Steer(q_n, q_s)$
7 **if** $IsCollisionFree(q_{new}, u_{new}, q_n)$ **then**
8 $Q_{near} \leftarrow Near(q_{new})$
9 $q_{min} \leftarrow ChooseParent(Q_{near}, q_n, q_{new})$
10 $\mathcal{T} \leftarrow InsertNode(q_{min}, q_{new}, \mathcal{T}, s)$
11 $\mathcal{T} \leftarrow Rewire(\mathcal{T}, Q_{near}, q_{min}, q_{new})$
12 **if** *A better solution is found* **then**
13 $Q_{sol} \leftarrow UpdateSolution(\mathcal{T})$
14 $M \leftarrow UpdateMilestone(Q_{sol})$
15 $\mathcal{S} \leftarrow UpdateSamplingCloud(\mathcal{S}, M)$
16 **return** $\mathcal{T}$

2.3.5 Cloud RRT*

In this section we explain an overall flow of cloud RRT*, whose pseudocode is shown in Alg. 2. At a high level the structure of our cloud RRT* is similar to that of RRT* except operations related to the sampling cloud and milestone. We first choose a sphere, s, from the sampling cloud $\mathcal{S}$ according to importance values of spheres, and prune s, if possible. We then generate a random sample, q_s [3-4:Alg. 2], and compute a nearest neighbor node, q_n, from the sample q_s by using $Nearest(\cdot)$, and compute a feasible trajectory from q_s to q_n based on $Steer(\cdot)$. If the trajectory does not have any collisions, compute nearby neighbor nodes within a radius by using $Near(\cdot)$. We then compute an optimal parent node and its edge to the tree, followed by the rewire operation [9-11: Alg. 2].

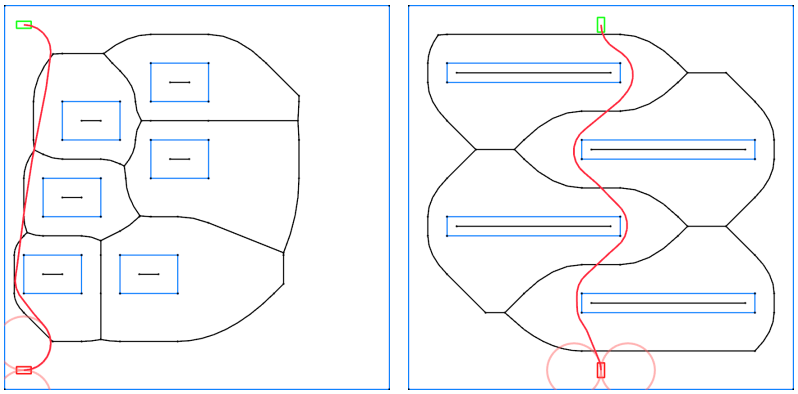
Whenever a better solution is found, we recompute a milestone and then update our sampling cloud [15:Alg. 2], as mentioned in Sec. 2.3.3.

2.4 RESULTS

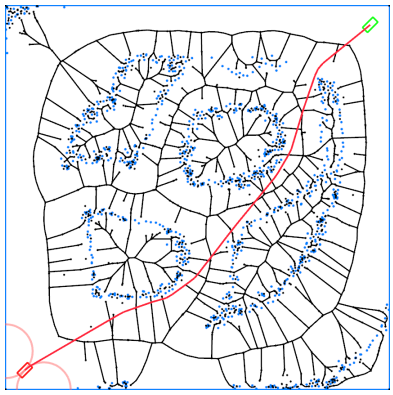
We have tested our method on a machine that has 3.5GHz Intel i7-2700K processor. We use a Voronoi diagram builder in the well-known Boost library [35] for our GVG-guided initialization.

To demonstrate benefits of our method, we have compared our method with the original RRT*, RRT*-Smart [28], and local biasing and node rejection techniques, denoted as LN, proposed by Akgun et al. [11]. We have tested different methods for

a kinematic motion planning problem with the Dubins vehicle model [36].



(a) Six squares (b) Four squares



(c) Many points

Figure 2.3: These figures show two 2D environments for a kinematic car. Red and green squares indicate initial and goal positions, respectively. The car is drawn with its two circles of the minimum turning radius at its initial position. Computed near-optimal paths are shown in red curves. Obstacles in the scene of (c) are represented in blue dots, which mimic real-world data.

2.4.1 Dubins Vehicle

We aim to generate optimal collision-free paths in an efficient manner for autonomous vehicle models. For the problem we use the Dubins vehicle model and need to respect kinematics of the model for computing collision-free paths. While our initial path is computed by only considering the geometry of obstacles based on GVG, we would like to demonstrate how our method behaves well with such kinematic constraints.

The Dubins vehicle is a simple car model that moves at a constant forward speed and has a maximum steering angle, ϕ, which implies the minimum turning radius, $\rho = L/tan(\phi)$, where L is a distance between front and rear axles, and $\phi < \pi/2$. Its system can be described by the following equations:

$$\begin{aligned} \dot{x} &= v_c cos\theta, \\ \dot{y} &= v_c sin\theta, \\ \dot{\theta} &= \frac{v_c}{L} \cdot tan(u), \end{aligned} \tag{2.7}$$

where v_c is a constant speed of the vehicle, and u is a steering input within $U = [-\phi, \phi]$. Its C-space is $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}$, and a configuration in the C-space can be denoted by $q = (x, y, \theta) \in \mathbf{X}$.

In this case $Proj(q)$ used for defining $\mathbf{X}_s$ (Eq. 2.2) is set to be (x, y). A local planner for the Dubins vehicle is implemented based on the kinodynamic planner presented by Karaman et al. [37]. We have tested three different

environments (Fig. 2.3) for the kinematic motion planning problem.

In addition to testing the original RRT* with uniform sampling (Uniform), RRT*-Smart, LN, and ours, we have also tested Uniform combined with GVG (Uniform + GVG), RRT* + GVG, and LN + GVG, since GVG can be combined with the tested prior methods. For prior methods combined with GVG, we use our initial sampling cloud computed from GVG (Sec. 2.3.2), but do not use our update nor pruning methods. By comparing our method with those prior methods combined with GVG, we can observe benefits of our update and pruning methods.

For the experiments, we set biasing factor, b, for RRT*-Smart, to be 5, where 1 divided by b is a frequency of intelligent sampling. As a result, for RRT*-Smart, we use its intelligent and uniform sampling in 1:4 ratio; we have tried many different values and decided 5 for b, since it gives the best result. For RRT*-Smart + GVG, we use intelligent sampling and our GVG based sampling in a 1:4 ratio. We set the biasing factor β for LN as 0.2, i.e., use its sampling and uniform sampling in a 1:4 ratio, since it also gives the best result. For LN + GVG, we use our GVG-based sampling, not the uniform sampling. Prior methods have their own parameters and we set their values based on ones reported in their original papers.

Fig. 2.4 show how different methods reduce the cost of solutions as we have more computation time up to five seconds. The computation time for our method includes all the operations of our method including the GVG construction and the initialization/update/pruning operations for our sampling cloud. Our method clearly outperforms all the prior methods. Furthermore, our method is able to compute shorter paths over all the prior methods that are even combined with our sampling cloud constructed with GVG.

This demonstrates benefits of updating and pruning our sampling cloud for achieving better convergence rate.

In the scene with four squares, most methods find a path that is homotopic to the optimal path, when they find a path whose cost is less than 20. Our method finds such a path faster than other methods. This is mainly because we maintain the global sampling distribution that tends to explore unvisited areas and prune unpromising sampling spheres. Furthermore, our method keeps to reduce its cost by finding shorter paths in the same homotopy class. Note that as we have better solutions, we create more localized sampling spheres for milestones by using smaller radii over prior spheres containing those milestones. Also, reducing orientation ranges of newly created spheres results in the same localized sampling. As a result, we can exploit newly better solutions and refine them in their homotopy classes.

Fig. 2.5 shows how long each tested method takes to compute a solution path that has a given cost (shown in the X-axis). This graph is another visualization of prior graphs of Fig. 2.4, which show the computational time in the X-axis. By summarizing results obtained from the tested three benchmarks, performances of different methods reaching to a particular cost can be ranked as follows: Uniform sampling $<$ Uniform + GVG $<$ LN $\simeq$ RRT*-Smart $<$ RRT*-Smart + GVG $<$ LN + GVG $<$ Ours. Overall our method shows performance improvement in a range between three and five times over RRT*-Smart and LN. Over those prior methods combined with GVG, our method shows improvements in a range between two and three times.

Discussions. Our method shows higher robustness over all the other tested methods. For example, the performance ranking of LN and RRT*-Smart shown in Fig. 2.4(a) varies depending on the available computation budget. Uniform sampling combined with GVG shows even better performance than LN+GVG and RRT*-Smart around 600 ms, as shown in Fig. 2.4(b). Also, in the scene with many points, the uniform sampling is better than RRT*-Smart and LN for solutions with less than 20.25 (Fig. 2.5(b)). On the other hand, our method shows higher performance over all the other methods across most available time budgets and given costs.

To confirm this, we have measured the standard deviation of costs of solutions among 100 attempts. Our method shows the lowest values over all the other methods across the three tested environments, while achieving the shortest paths. More information can be found in Table 2.1 showing various results including a cost of the

Table 2.1: Various statistics averaged over 100 trials with the running time budget of 5 seconds. Numbers in parentheses are standard deviations.

(a) Six squares (Ground truth optimal cost $\approx$ 14.480858)

	C_{final}	N_{update}	N_{node}	$N_{iteration}$
Uniform	20.39(10.21)	9.77	1099.83	2230.08
Uniform + GVG	15.46(0.57)	12.14	1261.92	2434.47
RRT*-Smart	18.26(2.41)	18.70	1566.66	3652.32
RRT*-Smart + GVG	15.44(0.55)	17.93	1344.37	2701.56
LN	17.21(6.23)	27.88	1250.41	4583.76
LN + GVG	14.95(0.37)	24.51	1236.80	4477.52
Ours	14.60(0.08)	21.65	1220.07	2982.75

(b) Four squares (Ground truth optimal cost $\approx$ 15.557956)

	C_{final}	N_{update}	N_{node}	$N_{iteration}$
Uniform	19.08(1.79)	10.66	1624.96	3836.78
Uniform + GVG	17.36(0.81)	11.57	1792.89	3521.99
RRT*-Smart	18.18(3.10)	17.96	1573.74	3716.09
RRT*-Smart + GVG	16.82(0.91)	19.74	1704.48	3482.12
LN	18.34(3.35)	20.01	1506.20	3866.89
LN + GVG	16.70(0.68)	18.61	1519.68	3862.09
Ours	16.33(0.34)	19.13	1552.79	3261.21

(c) Many points (Ground truth optimal cost $\approx$ 18.9)

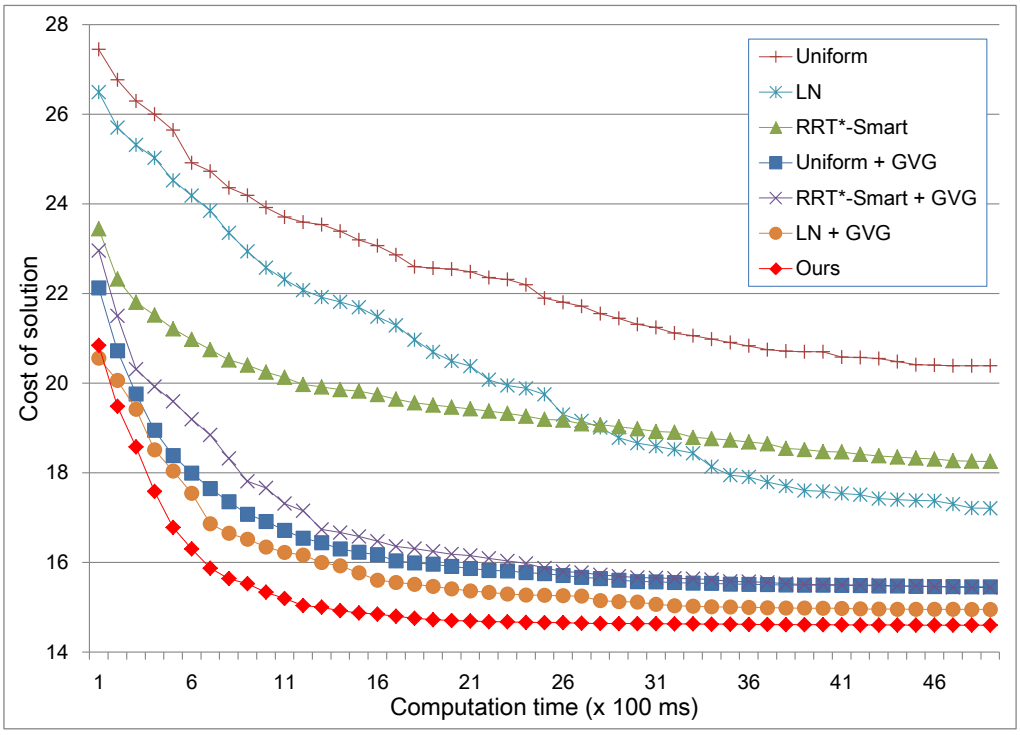
	C_{final}	N_{update}	N_{node}	$N_{iteration}$
Uniform	21.82(0.578)	4.80	705.74	2906.62
Uniform + GVG	20.79(1.013)	8.55	750.70	2378.11
RRT*-Smart	22.08(2.24)	11.11	682.21	2613.84
RRT*-Smart + GVG	20.95(1.10)	13.55	735.30	2241.22
LN	22.16(1.84)	10.78	747.11	3096.56
LN + GVG	20.38(1.23)	16.40	689.75	2590.31
Ours	19.81(0.32)	15.10	750.70	2467.45

final solution, C_{final}, with its std. deviation, the average number of nodes, N_{node}, the number of solution updates, N_{update}, the number of iterations, $N_{iteration}$, and ground truth optimal path costs that are computed by running our method in a few hours.

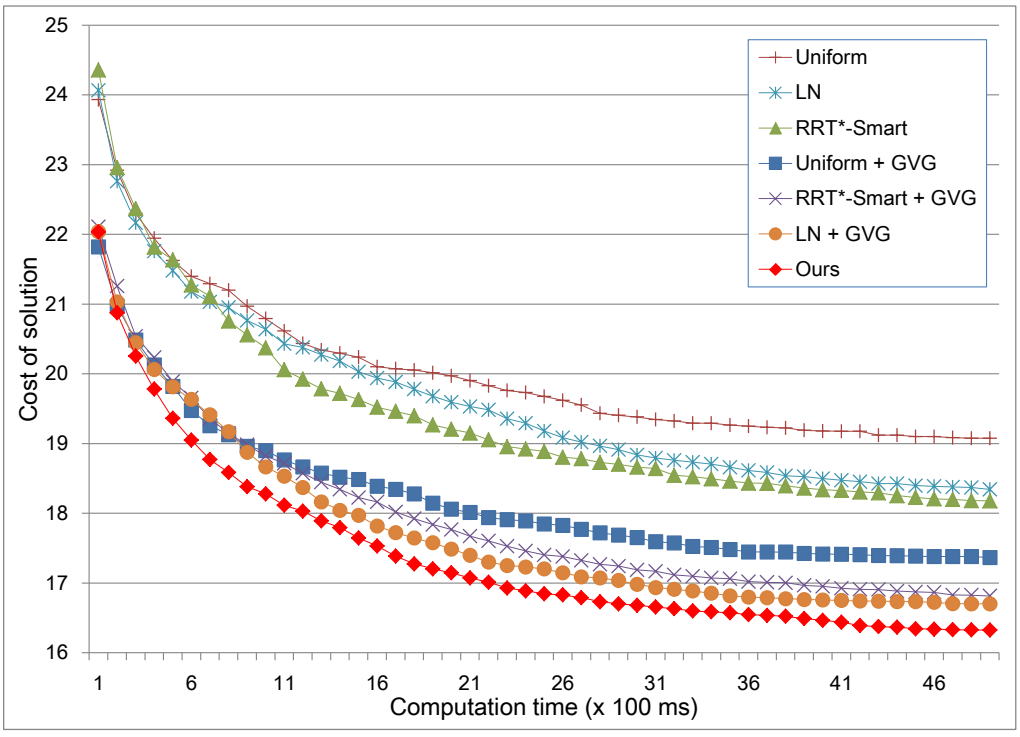
Table 2.2 shows the GVG construction time, T_{gvg}, and our GVG-guided initialization time, T_{init}. Overall they take a small portion, less than 10 ms, in our tested environments. The table also shows the number of computed Voronoi vertices and edges, N_{ver} and N_{edge}, as well as the number of spheres, N_{init}, constructed by the GVG-guided initialization.

Table 2.2: Statistics related to our GVG-guided initialization.

	T_{gvg}	T_{sphere}	N_{ver}	N_{edge}	N_{init}
Six squares	0.73 ms	2.34 ms	94	298	51
Four squares	0.56 ms	1.75 ms	168	214	47
Many points	4.92 ms	3.31 ms	1554	4670	71

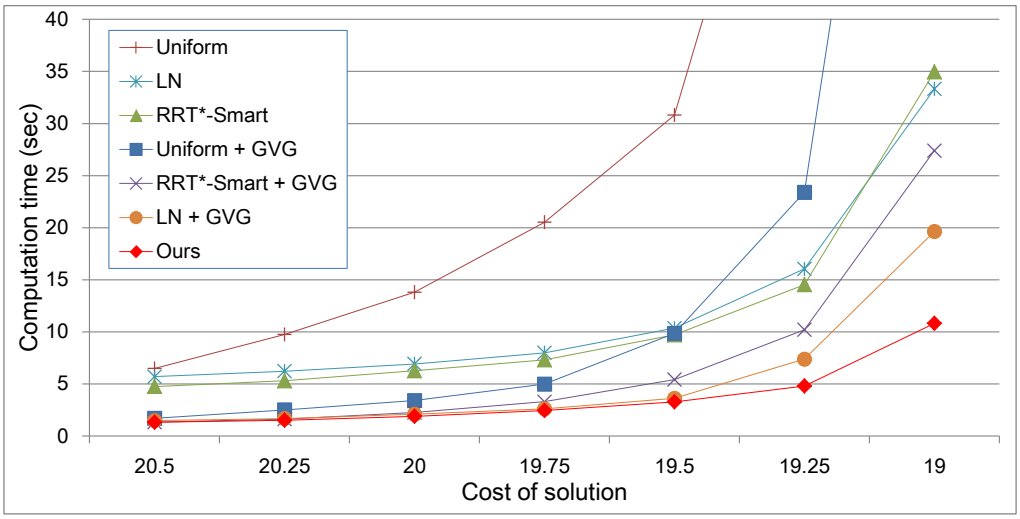


(a) Six squares

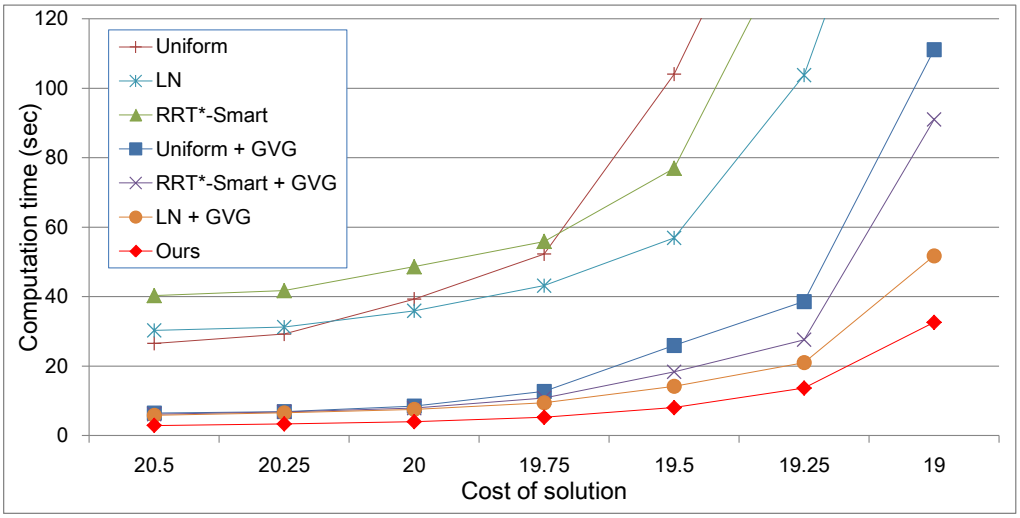


(b) Four squares

Figure 2.4: This figure shows a graph of costs of solutions as a function of computation time with different methods. Our method shows the best quality over other methods that are even combined with GVG-based sampling.



(a) Four squares



(b) Many points

Figure 2.5: Computation time to reach a specific cost of the solution as a function of target costs.

Chapter 3. Dancing PRM*: Simultaneous Planning of Sampling and Optimization with Collision Free Space Approximation

3.1 Introduction

Sampling-based motion planning algorithms have been well studied for the past several decades thanks to their probabilistic completeness and wide applicability. Some prominent examples are RRT [4] and PRM [5], which can be viewed as random geometric graph construction in the configuration-free space.

For the asymptotic optimal motion planning, Karaman and Frazzoli presented RRT*, PRM*, and RRG [6], which guarantee almost-sure asymptotic optimality. These optimal variants successfully opened a new research area in motion planning by providing a theoretical foundation and have been applied to practical solutions for real problems [7, 38, 39].

In contrast to the sampling-based planners, optimization-based planners convert a non-convex motion planning problem into a sequence of convex problems for quickly finding a locally optimal solution [40, 41, 42]. These approaches mainly aim to minimize an objective function with respect to planning constraints, such as smoothness for optimality, while estimating the gradients of obstacle potential for feasibility, i.e., the non-collision constraint.

Some papers have studied the configuration-free space approximation for various purposes [31, 43, 44, 45, 46, 47, 48, 49]. Among them, Shkolnik and Tedrake represented the configuration-free space as a set of hyperspheres using empirical collision information for efficient biased sampling [31]. Bialkowski et al. took a similar approach but used explicit proximity computation to compute the boundary of the hyperspheres [46]. In their following work, they used a KD-tree based representation to represent the approximate configuration-free space with provable convergence to the ground-truth []. On the other hand, Denny and Amato suggested a methodology of mapping configuration-obstacle space to guide the sampling within free space [50, 51]. Furthermore, they presented an extensive work in conjunction with lazy graph expansion [52] for efficient graph expansion.

Kim et al. suggested using configuration-free space approximation to predict regions that likely have no collisions [49]. This approach is tailored to lazy collision checking to balance the overhead of collision checking and graph restructuring. Pan and Manocha proposed a probabilistic collision checking with free space approximation [48]. They introduced an environment learning phase to understand the given geometric structure and then exploited learned knowledge using spatial coherency for a probabilistic collision checking.

As a hybrid approach, Choudhury et al. suggested *RABIT** (Regionally Accelerated Batched Informed Trees [53]) which is the integration of a sampling-based algorithm BIT* (Batched Informed Tree [54]) and an optimization-based algorithm, CHOMP (Covariant Hamiltonian Optimization for Motion Planning, [40]). This hybrid algorithm considers the pros and cons of both approaches together to identify difficult-to-sample homotopy of the solution path efficiently, while preserving the asymptotic optimality.

It is, however, only workable under the assumption that a precomputed workspace information, e.g., a distance field, is given a priori or is analytically computable on demand, which can be a serious burden in practical problems.

In this work, we present a hybrid approach of sampling-based and optimization-based planning, in which the entire planning process is accomplished on the fly. The proposed algorithm uses empirical collision information to learn the configuration-free space during the execution. The optimization-based planner utilizes the learned information to guide trajectories toward the configuration-free space to establish more connections between sampled configurations. We also suggest an efficient decentralization of samples so that two different types of planners can

Algorithm 3: NAÏVE PRM*

```
1  V ← {q_init, q_goal}
2  E ← ∅
3  while Termination condition is not satisfied do
4      q_sample ← Sample()
5      if IsCollisionFree(q_sample) then
6          Insert q_sample to V
7          Q_near ← Near(q_sample)
8          foreach q_near ∈ Q_near do
9              if IsCollisionFree((q_near, q_sample)) then
10                 Insert (q_near, q_sample) to E
11     UpdateSolutionPath(G)
12 return SolutionPath(G)
```

work seamlessly with our approximate configuration-free space.

In the subsequent sections, we first explain the preliminaries of the sampling-based and optimization-based approach (Sec. 3.2). Sec. 3.3 gives an overview of the proposed algorithm (Sec. 3.3.1), and how to approximate the configuration-free space with sampling-based motion planning (Sec. 3.3.2), followed by optimization based local planning with learned information (Sec. 3.3.3). We then demonstrate its benefits by experiments with various dimensions against other state-of-the-art planners (Sec. 3.4). Lastly, we discuss the properties of our method and the configuration-free space approximation in detail (Sec. 3.5).

3.2 Background

This section reviews major previous studies and presents preliminaries and notations employed throughout the paper.

3.2.1 Sampling-based Motion Planning

In this work, we mainly consider the optimal motion planning problem, whose objective is to find a feasible and optimal trajectory ξ connecting two given end points q_{init} and q_{goal} satisfying $\xi(0) = q_{init}$ and $\xi(1) = q_{goal}$ in the configuration space $\mathbb{X}$, where a trajectory $\xi : [0,1] \to \mathbb{X}$. For feasibility, ξ should lie in the configuration free space $\mathbb{X}_{free}$, where $\mathbb{X}_{free} \subset \mathbb{X}$ and the configuration obstacle space $\mathbb{X}_{obs}$ is defined to be $\mathbb{X} \setminus \mathbb{X}_{free}$.

To explain how sampling-based approaches construct a search graph $G = (V, E) \in \mathbb{X}$, we briefly explain naïve PRM* shown in Alg. 3, which is one of the prominent sampling-based planners.

In each iteration of Alg. 3, PRM* samples a random configuration q_{sample} and checks its validity with $IsCollisionFree(\cdot)$ (Line: 5) which returns *true* if a given configuration q or an edge (q_i, q_j) is valid i.e. $\in \mathbb{X}_{free}$. For each valid configuration q_{sample}, r-nn (r-nearest neighbor) query of $Near(\cdot)$ finds near neighbors, Q_{near}, (Line: 7) within a ball of radius $\gamma\left(\frac{\log|V|}{|V|}\right)^{1/d}$ centered at q_{sample}, where d is the dimension of the problem, $|V|$ is the cardinality of V, and γ is a user defined constant greater than 1 ([6]). k-nn (k-nearest neighbor) query can also be an alternative to r-nn within $Near(\cdot)$; in that case, k is defined by $\lceil \gamma(\mathrm{e} + \frac{\mathrm{e}}{d}) \cdot log(|V|) \rceil$ ([6]).

Line 9 checks collision for every possible connection of ($q_{near} \in Q_{near}$, q_{sample}) to find a feasible connection between configurations in G; this is also known as *local planning*. $UpdateSolutionPath(\cdot)$ computes the shortest path from q_{init} to q_{goal} on the constructed graph G if the anytime property is required for the given problem.

Otherwise, the shortest path is computed (Line: 12) using a well-known A* or Dijkstra's algorithm at the end of execution.

In this study, the foundation of the proposed algorithm is based on sampling-based planning, mainly with PRM*. Furthermore, to see a wide applicability of our approach, we also test another sampling-based planning method BIT*. Irrespective of PRM* and BIT*, the role of the employed sampling-based planner within our method is to divide the entire motion planning problem into a set of smaller local planning problems, while approximating the configuration free space with empirical collisions. We discuss how the optimization-based local planning works with our spatial information in the subsequent section.

3.2.2 Optimization-based Motion Planning

Our local planner is based on a gradient optimization technique, CHOMP (Covariant Hamiltonian Optimization for Motion Planning, [40]). We first briefly review the concept of CHOMP and discuss the motivation with our observations.

The objective of CHOMP is to find a smooth, collision-free trajectory ξ, exactly like that of sampling-based planning. The objective function, $\mathcal{U}(\xi)$, is then formalized as the following:

$$\mathcal{U}(\xi) = f_{prior}(\xi) + \lambda \cdot f_{obs}(\xi), \tag{3.1}$$

where f_{prior} can be considered as a sum of squared derivatives for the trajectory ξ to satisfy local optimality and additional constraints, such as controlling smoothness or limiting the maximum acceleration. The obstacle cost function f_{obs} penalizes a configuration of a robot for being close to $\mathbb{X}_{obs}$ to avoid any collision. To be specific, f_{obs} and its gradient ∇f_{obs} are formalized in Eqs. 3.2 and 3.3, respectively:

$$f_{obs}(\xi) = \int_0^1 \int_B c(x(\xi(t),b)) \left\| \frac{d}{dt} x(\xi(t),b) \right\| db\, dt, \tag{3.2}$$

$$\nabla f_{obs}(\xi) = \int_B J^T \|x'\| \cdot [(I - \hat{x}'\hat{x}'^T)\nabla c - c\kappa]\, db. \tag{3.3}$$

Roughly speaking, Eq. 3.2 stands for a cost integration over the given trajectory ξ where an obstacle potential function, $c(\cdot)$ penalizes a configuration being near the obstacle space and the term inside $\|\cdot\|$ is a measurable length of a trajectory through $x(\xi(t),b)$ in the workspace. The notations used in both equations and throughout the paper are also summarized in Table 3.1.

The computation of the obstacle potential $c(\cdot)$ depends on a body simplification, and workspace information such as the distance field ([40]), which can be computed using Euclidean Distance Transforms (EDTs) in the workspace for instance.

The robot model B is generally simplified by a swept-sphere volume ([40, 42]). $x(\cdot)$ plays a mapping role from a configuration $\xi(t)$ defined in the configuration space to the workspace. As a result, the integration of whole body B results in the obstacle potential for a configuration $\xi(t)$.

Eq. 3.3 shows the gradient of Eq. 3.2, where $\nabla c(\cdot)$ is the gradient of obstacle potential $c(\cdot)$, and κ is the curvature of the trajectory. The objective function of CHOMP contains obstacle potential terms such as $c(\cdot)$ and ∇c, which are hard to compute in the configuration space. For this reason, the conversion from the configuration space to the workspace with $x(\cdot)$ and J is introduced to compute f_{obs} and ∇f_{obs} for a trajectory defined in the configuration space using workspace information.

When it comes to practical performance, the use of workspace obstacle potential makes the planning process less affected by the dimensionality of the configuration space. As a result, it generally provides a faster dynamic trajectory generation and the stable result compared to sampling-based planners.

Table 3.1: Notation summary table.

Notation	Description
$\xi(t)$	Configuration on the trajectory ξ at a time $t = [0,1]$.
B	Set of entire body points for a given robot model.
$x(\xi(t),b)$	Mapping of a body point $b \in B$ at a configuration $\xi(t)$ to the corresponding point in the workspace.
x'	Derivative of $x(\cdot)$.
$c(\cdot)$	Obstacle potential for being close or inside $\mathbb{X}_{obs}$.
J	Kinematic Jacobian, i.e., $\frac{d}{dq}x(q,b), q \in \mathbb{X}$ and $b \in B$.
V,E	Vertex and edge set of the search graph G.
V^*	Set of sample configurations observed during planning regardless of collision-freeness.

Fig. 3.1 shows a 2D manipulation planning problem with workspace obstacle potential in an illustrative way. In the left figure, a robot body is represented as a set of reference points B with swept-sphere volumes (red circles). The approximate volume is usually set conservatively to ensure that the entire robot body is lying inside. This approximation makes the collision checking process much lighter than checking the exact robot body with the signed distance field shown in the right figure.

A set of points in the workspace $x(\xi(t),b), \forall b \in B$ are computed from a configuration $\xi(t)$, and we then average them according to Eq. 3.2 and Eq. 3.3 with kinematic Jacobian J in order to be used in the configuration space where ξ is defined.

The limitation of the aforementioned process can be summarized as follows. First, CHOMP only guarantees the convergence to a local optimum since the optimization process exploits the local convexity of f_{obs}. Fortunately, the local optimality issue has been studied extensively with a stochastic sampling ([40, 41]) and a hybrid approach of sampling-based and optimization-based planning ([53, 55]). Second, CHOMP and its variants including other existing hybrid frameworks assume a discretized and approximate representation, e.g., distance fields in finite resolutions, in the workspace and the simplified robot model B. For these reasons, the resolution of the distance field and the simplified robot representation can affect the performance or further harm the completeness.

On the other hand, Fig. 3.2 shows what they look like in the configuration space, where obstacle potentials can be computed without the robot body simplification B, $x(\cdot)$ and Jacobian, J. It has been, however, known as a hard problem to compute such information and even represent it in a usable form, especially in a high-dimensional space. It will, thus, be the key of our problem, to represent and approximate such space for seamless integration of hybrid planners.

In the subsequent section, we describe how the proposed algorithm approximates such space without using discretized workspace information or expensive overheads related to proximity computation in the configuration space, while preserving the asymptotic optimality by integrating a sampling-based approach. We also suggest space information decentralization associated with the search graph for efficient referencing.

3.3 Algorithm

In this section, we describe the overview of our algorithm first and then elaborate each component in it with underlying theoretical meanings.

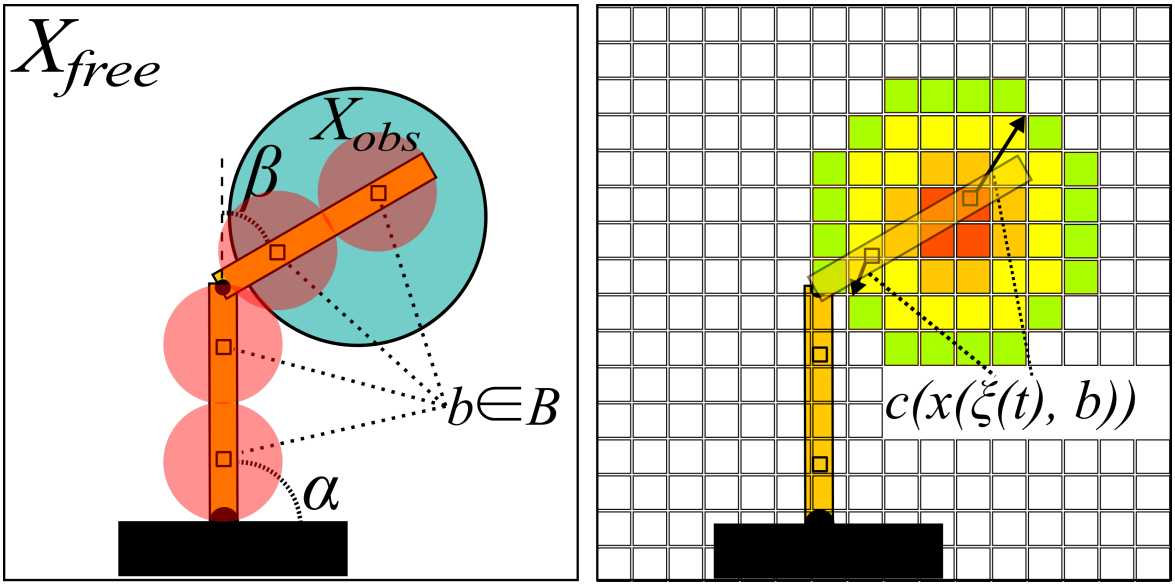


Figure 3.1: These figures show a 2D manipulation problem with two joints α and β in a planar space. The left figure shows how the robot is represented for working with workspace obstacle information (left) and the signed distance field of the environment (right). In the left, a robot arm is simplified by a set of body points $b \in B$ (small squares), each of which is a center of swept-sphere volume (red circles). In the right, potentials against the obstacle are computed by the proximity of those in the workspace (black arrow).

3.3.1 Overview

At a high level, the core of our idea is based on the integration of optimization-based and sampling-based planning without a priori knowledge of the given environment.

Fig. 3.4 shows an abstracted concept of the proposed algorithm. The main flow with the four solid boxes shows a single iteration of the generic sampling-based motion planning algorithm which also can correspond to PRM in Alg. 3. We first approximate the configuration-free space $\widetilde{\mathbb{X}}_{free}$ using empirical collisions found in both vertex and edge collision checking on the fly (Sec. 3.3.2). We also propagate the empirical collision information, and inherit from near neighbors on the implicit proximity graph (Sec. 3.3.2 and 3.5.3). Each edge is processed in local planning and rejected edges (e.g., due to the collisions), are handled by our local trajectory optimization as backup local planning. The optimizer works with our configuration-free space approximation; hence, the trajectory optimization is solely performed in the configuration space without any prior workspace information.

In order to leverage the efficiency of our local trajectory optimization, minimization of the number of local planning is necessary to concentrate on promising edges by skipping unnecessary ones, which can be achieved by lazy collision checking [54, 56, 57].

To show the novelty and applicability, we suggest two different asymptotic optimal planners, Dancing PRM* and BDT* (Batch Dancing Tree) named after lazy PRM* [56] and BIT* (Batch Informed Tree [54]), respectively. Lazy PRM* is based on DSPT (Dynamic Shortest Path Tree [58]) for lazy collision checking, while BIT* uses LPA* (Life-long Planning A* [59]) based approach. We also discuss how differently both algorithmic frameworks behave in terms of lazy graph expansion in Sec. 3.3.4.

In what follows, we first introduce the configuration-free space approximation and the local trajectory optimization with Dancing PRM* as a generalized form of a sampling-based motion planner.

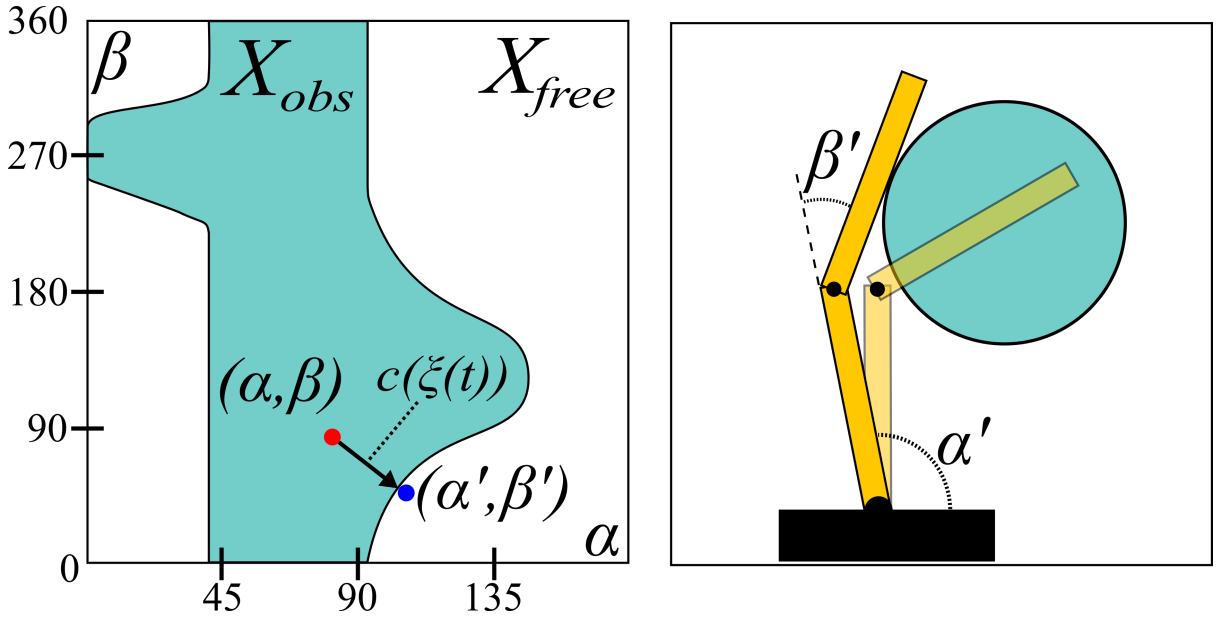


Figure 3.2: The left figure shows the configuration space of Fig. 3.1. If we can represent the configuration space in a usable form, we can directly compute the obstacle potential and its gradient without any other conversions between the workspace and configuration space. The right figure shows the corresponding movement in the workspace between two configurations (α,β) and (α', β').

3.3.2 Configuration Free Space Approximation

The main purpose of the approximate configuration-free space of $\mathbb{X}_{free}$, $\widetilde{\mathbb{X}}_{free}$ is to efficiently guide trajectories towards local configuration free space during the optimization.

We choose to represent $\widetilde{\mathbb{X}}_{free}$ as a set of *hyperspheres* motivated by previous studies ([31, 46, 49]) for scalability and light-weight proximity computation, e.g., the distance function. While the representation is analogously defined, our study differs in terms of the approximation procedure and accessing strategy for the efficient integration with optimization-based planning explained in a later paragraph.

Fig. 3.3(a) shows a conceptual image of $\widetilde{\mathbb{X}}_{free}$, which can be formalized as:

$$\widetilde{\mathbb{X}}_{free} = \{x \mid \|x - q_i\| < r_{q_i}\},\ q_i \in V, \tag{3.4}$$

where r_{q_i} is the approximate minimum distance to the closest obstacle in $\mathbb{X}_{obs}$.

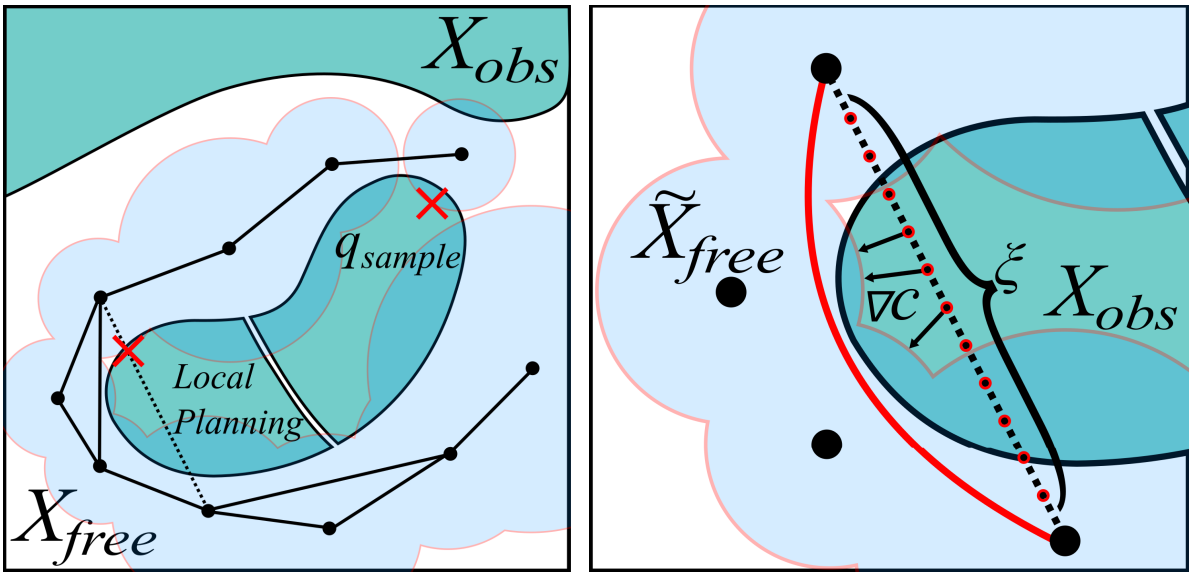
Intuitively, each configuration $q \in V$ is associated with a single hypersphere of which radius is the distance to the closest obstacle *witness*, $w_q \in \mathbb{X}_{obs}$ found during the execution, such that $\|q - w_q\| = r_q$.

A new witness w_q can be generated and replaced by the samples listed in List. 3.1 during the execution, as long as it results in a smaller radius.

Note that the intermediate configuration (2) can be easily computed during an edge collision checking with a discrete collision checker which is widely used in the most conventional sampling-based planners ([3]).

The above procedures are intended to exploit local empirical collisions found by collision checking; both vertex and edge collision checking, not to use any other additional proximity computation.

Alg. 4 shows Dancing PRM* which is integrated with the configuration-free space approximation procedure. The main flow is almost identical to that of PRM* in Alg. 3, and procedures newly added or that behave differently from the original are highlighted in the pseudocode. The lines related to the configuration-free space approximation are 15-16, 17 and 13 which correspond to the cases in Tab. 3.1. We discuss the details in the subsequent



(a) (b)

Figure 3.3: The left is a visualization of $\widetilde{\mathbb{X}}_{free}$, regions covered by a set of light blue circles in 2D; for simplicity, we show the merged region of circles, instead of visualizing each circle. Each configuration $q \in V$ is associated with an approximate collision-free hypersphere in $\mathbb{X}$. Their radii are trimmed by *witness* (red cross symbol) which is a configuration in $\mathbb{X}_{obs}$ found during a local planning (dotted black segment on the left side) or a sample q_{sample} (the right side in the same figure). The right figure shows an example of local optimization for a trajectory ξ. Black arrows show the gradient of obstacle potential computed with our approximate configuration space and the red curved segment shows an optimized trajectory. Each red dot indicates an intermediate configuration $\xi(t)$ on the discretized ξ.

sections.

Decentralized storage for observed collision states. Since $\widetilde{\mathbb{X}}_{free}$ is associated with the search graph G, we need an appropriate method to use $\widetilde{\mathbb{X}}_{free}$ to optimize an edge; an example is shown in Fig. 3.3(b).

As an efficient handling for our approximate configuration free space, we adopt a decentralized storage strategy for $\widetilde{\mathbb{X}}_{free}$, where each $q \in V$ maintains a subset $V_q \subset V$. V_q is updated with its near neighbors (Line: 6, 8, 11 in Alg. 4).

As a result, our search graph G can be considered as a proximity graph ([60]), and also similar to the neighborhood set of RRT^X ([61]). The primary benefit of this kind of structure is that each configuration $q \in V$ can retrieve a local subset of $\widetilde{\mathbb{X}}_{free}$ centered at q without performing any additional NN query.

When a vertex is inserted into V_q, the maximum distance from q to the newly added vertex is bounded by $\gamma\left(\frac{\log|V|}{|V|}\right)^{1/d}$, since the bound value is identical to the search radius of $Near(\cdot)$ in the nature of sampling-based motion planning. Therefore, for an edge $(q \in V, w \in V)$, we can guarantee that a subset of $\widetilde{\mathbb{X}}_{free}$ covered by V_q, and V_w entirely surrounds the edge.

Witness propagation step. To acquire more accurate $\widetilde{\mathbb{X}}_{free}$ in practice, we apply a witness propagation step (Line: 13 in Alg. 4). Our key insight for the witness propagation is to treat finding the closest configuration obstacle for $q \in \mathbb{X}_{free}$ as a connectivity problem on random geometric graphs ([6]). This suggests that for an arbitrary configuration $q \in V$, all witnesses of V located within the ball of radius $\gamma\left(\frac{\log|V|}{|V|}\right)^{1/d}$ centered at q serve as candidates for w_q.

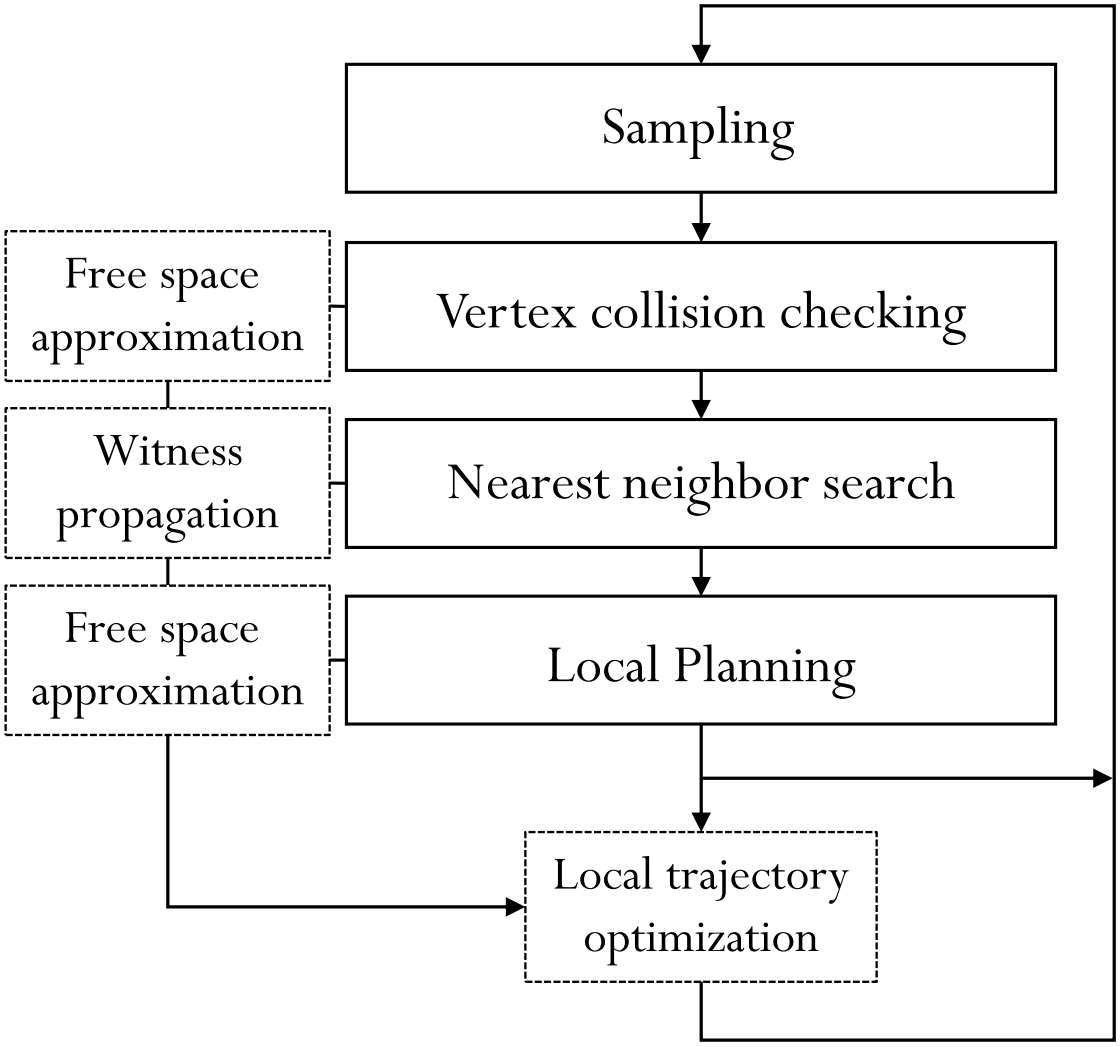


Figure 3.4: An abstraction of the proposed algorithm. The dotted boxes indicate proposed components in conjunction with a generic sampling-based motion planning algorithm with the solid boxes.

For this purpose, we define $PropagateCFreeSpace(\cdot)$ (Alg. 5), which initializes a new sample configuration $r_{q_{sample}}$ using witnesses of its near neighbors Q_{near}, and also propagates its witness $w_{q_{sample}}$ to Q_{near} at the same time. Fig. 3.5 shows an exemplar sequence of witness propagation for a sample configuration q_{sample} depicted in Alg. 5. This process enhances the association between local near neighbors and the closest witness to reduce the error of our configuration free space approximation. Likewise, $UpdateCFreeSpace(\cdot)$ (Line: 16 in Alg. 4) updates the radii of cumulative neighbors of $q_{nearest}$, $V_{q_{nearest}}$ with $q_{sample} \in \mathbb{X}_{obs}$.

It is, however, inevitable to over-estimate the collision-free radii with a limited number of samples. For this reason, we propose a statistical technique to reduce the error further, named *radius compensation*, in the subsequent section, and also analyze the approximation error in Sec. 3.5.3.

3.3.3 Local trajectory optimization in Configuration Space

Our CHOMP-based optimizer is performed lazily in where explicit edge collision checkings are invoked. In Dancing PRM*, it is $CheckSolutionPath(\cdot)$ as depicted in Alg. 6. Compared to lazy PRM* ([56]), we additionally activate our local trajectory optimizer, $Optimize(\cdot)$ only after a collision checking for an edge $(v \in V, w \in V)$ is

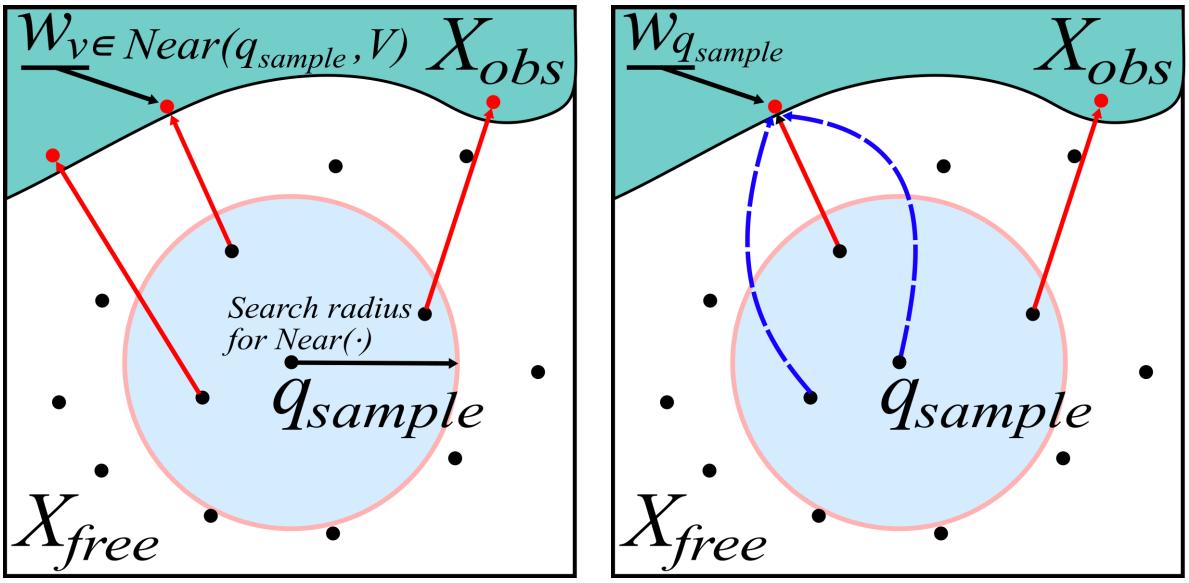


Figure 3.5: A sequence of witness propagation for a sample configuration q_{sample} and its near neighbors within the light-blue circle with a radius of $\gamma\left(\frac{\log|V|}{|V|}\right)^{1/d}$. Pairs of configuration and its former witness are connected by red segments, and blue dotted lines indicate witness newly updated. q_{sample} inherits its witness $w_{q_{sample}}$ first (left) from its near neighbors. It is then propagated to other neighbors in the circle to ensure that near neighbors have a witness as the closest empirical collision.

List 3.1: List of samples causing the update of $\widetilde{\mathbb{X}}_{free}$.

1. A sampled configuration $q_{sample} \in \mathbb{X}_{obs}$ (Line: 5 in Alg 3).
2. An intermediate configuration that turned out to be in $\mathbb{X}_{obs}$ during an explicit collision checking (Line: 9 and 11 in Alg. 3).
3. A witness propagated from near neighbors.

failed, it can be thus considered as a backup local planner.

At the beginning of each iteration in *CheckSolutionPath*$(\cdot)$ (Alg. 6), we retrieve a provisional solution path $E_{solution} \subset E$ (Line: 3), which possibly contains infeasible edges due to the lazy collision checking. When the solution path validation fails by an intermediate configuration $q_{obs} \in \mathbb{X}_{obs}$ along the edge e_i (Line: 5), we remove e_i from E (Line: 6). We then update the local subset of $\widetilde{\mathbb{X}}_{free}$, i.e., radii of configurations in $V_v \cup V_w$ for an edge $e_i = (v, w)$ (Line: 7).

Our optimization-based local planner then attempts to optimize the invalid edge e_i (Line: 8) to find a new trajectory bypassing local $\mathbb{X}_{obs}$ using $\widetilde{\mathbb{X}}_{free}$. Successful optimization yields a non-linear trajectory σ_{opt}, which increases the chance of reducing the cost of the solution path or finding a better homotopy.

The local planner explicitly takes into account the obstacle potential computation with our approximate configuration free space $\widetilde{\mathbb{X}}_{free}$, which learns the given arbitrary environment dynamically. This generality separates our work from (RABIT*, [53]) which assumes the obstacle potential, f_{obs} and ∇f_{obs}, to be given a priori or analytically computable.

When it comes to the optimization process, we replace the obstacle potential computations (Eqs. 3.2, 3.3)

Algorithm 4: DANCING PRM*

$V \leftarrow \{q_{init}, q_{goal}\}$; $E \leftarrow \emptyset$
while *Termination condition is not satisfied* **do**
  $q_{sample} \leftarrow Sample()$
  if $IsCollisionFree(q_{sample})$ **then**
    Insert q_{sample} to V
    $V_{q_{sample}} \leftarrow \emptyset$
    $Q_{near} \leftarrow Near(q_{sample}, V)$
    Insert Q_{near} to $V_{q_{sample}}$
    foreach $q_{near} \in Q_{near}$ **do**
      Insert (q_{near}, q_{sample}) to E
      Insert q_{sample} to $V_{q_{near}}$
    $UpdateShortestPathTree(\cdot)$
    $PropagateCFreeSpace(Q_{near}, q_{sample})$
  else
    $q_{nearest} \leftarrow Nearest(q_{sample}, V)$
    $UpdateCFreeSpace(q_{sample})$
  $CheckSolutionPath(G)$
return $SolutionPath(G)$

with more simpler forms (Eqs. 3.8, 3.5) by removing the conversion between the workspace and configuration space.

In the objective function of Eq. 3.1, f_{prior} is generally assumed to be independent of the environment ([40]) and computed in the configuration space. We, therefore, only deal with f_{obs} depicted in the following equation:

$$f_{obs}(\xi) = \int_0^1 c(\xi(t)) \parallel \frac{d}{dt}\xi(t) \parallel dt. \tag{3.5}$$

Unlike the original form in Eq. 3.2, we can naturally eliminate the following things for our obstacle cost function:

1. The use of resolution-complete workspace obstacle information.
2. The workspace-configuration mapping function $x(\cdot)$.
3. The integration over the simplified body model B.

These are possible because we can directly compute the obstacle potential $c(\xi(t))$ with the following equation as formalized in [40]:

$$c(\xi(t)) = \begin{cases} -\mathfrak{D}(\xi(t)) + \frac{1}{2}\varepsilon, & \mathfrak{D}(\xi(t)) < 0 \\ \frac{1}{2\varepsilon}(\mathfrak{D}(\xi(t)) - \varepsilon)^2, & 0 < \mathfrak{D}(\xi(t)) \leq \varepsilon \\ 0, & otherwise, \end{cases} \tag{3.6}$$

where ε is the clearance threshold and $\mathfrak{D}(\xi(t))$ is the distance field value computed from our approximate configuration free space $\widetilde{\mathbb{X}}_{free}$:

$$\mathfrak{D}(\xi(t)) = -\min_{\forall q \in V}(\|\xi(t) - q\| - \omega(|V^*|) \cdot r_q). \tag{3.7}$$

Algorithm 5: *PropagateCFreeSpace*

Input: q_{sample}, a sample configuration,
Q_{near}, a set of near neighbor of q_{sample}

$r_{q_{sample}} \leftarrow \infty$; $w_{q_{sample}} \leftarrow \emptyset$
foreach $q_{near} \in Q_{near}$ **do**
  if $(w_{q_{near}} \neq \varnothing) \wedge (\|w_{q_{near}} - q_{sample}\| < r_{q_{sample}})$ **then**
    $r_{q_{sample}} \leftarrow \|w_{q_{near}} - q_{sample}\|$
    $w_{q_{sample}} \leftarrow w_{q_{near}}$
foreach $q_{near} \in Q_{near}$ **do**
  if $(w_{q_{sample}} \neq \varnothing) \wedge (\|w_{q_{sample}} - q_{near}\| < r_{q_{near}})$ **then**
    $r_{q_{near}} \leftarrow \|w_{q_{sample}} - q_{near}\|$
    $w_{q_{near}} \leftarrow w_{q_{sample}}$

According to the general definition of a signed distance field, it gives a negated distance to the closest $\widetilde{\mathbb{X}}_{free}$ for a configuration outside $\widetilde{\mathbb{X}}_{free}$, and we discuss the definition of $\omega(\cdot)$ in the subsequent section.

Radius Compensation. The concept of $\omega(\cdot)$ in Eq. 3.7 is to compensate the overestimation of r_q. Since our approximation entirely depends on the random sampling procedure, we can expect an overestimation of collision-free radius r, i.e., $r \geq r^*$, where r^* is the ground-truth distance to the closest obstacle space. To accommodate this approximation error, we apply a *radius compensation* procedure, defined by $\omega(n)$, a parameter function of n, to be $max(1-\zeta \cdot \delta(n), 0)$. $\omega(n)$ is designed to converge toward 1 as $n \to \infty$, which is considered as a sparsity of V^* in our work. We discuss our approximation process and radius compensation in more detail and experimentally show its impact in terms of the accuracy later in Sec. 3.5.

Fig. 3.3(b) visualizes an example of a gradient of obstacle potentials denoted by black arrows in the configuration space. By applying compensation $\omega(\cdot)$, the boundary of $\widetilde{\mathbb{X}}_{free}$ can shrink toward V and the optimizer works more conservatively. We can also compute ∇f_{obs}, as follows:

$$\nabla f_{obs}(\xi) = \|\xi'\| \cdot [(I - \hat{\xi}' \cdot \hat{\xi}'^{T})\nabla c - c\kappa], \tag{3.8}$$

where ∇c for a specific $\xi(t)$ is a normalized d-dimensional direction vector toward the closest $\widetilde{\mathbb{X}}_{free}$, and can be computed as $\frac{p-\xi(t)}{\|p-\xi(t)\|}$, where

$$p = \arg\min_{q \in V} (\|\xi(t) - q\| - \omega(|V^*|) \cdot r_q) \tag{3.9}$$

It is, however, computationally expensive to compute the obstacle potentials $c(\xi(t))$ and $\nabla c(\xi(t))$ if we consider the entire V, which must be evaluated repeatedly during the optimization. To this end, we restrict the configuration space used for a local optimization of ξ (q_{from}, q_{to}) with $V_{q_{from}} \cup V_{q_{to}}$ only. First, this choice is made mainly because

(1) $V_{q \in V}$ can be constructed with $Near(\cdot)$ in the nature of PRM*-like planner without having additional NN-query and

(2) maintaining more samples in $V_{q_{from}} \cup V_{q_{to}}$ can give rise to a substantial overhead.

The reduction of this approach in terms of overheads is also discussed in Sec. 3.5.2.

Nonetheless, this approach may result in a sub-optimal result, but this happens in a low probability. Furthermore, this sub-optimal result will be improved as we have more sampling iterations. Under these circumstances,

Algorithm 6: *CheckSolutionPath*

Input: G, a search graph

1 $E_{solution} \leftarrow \emptyset$
2 **repeat**
3 $\quad E_{solution} \leftarrow ProvisionalSolutionPath(G)$
4 $\quad$ **foreach** $e_i \in E_{solution}$ **do**
5 $\quad\quad$ **if** $\neg IsCollisionFree(e_i)$ **then**
6 $\quad\quad\quad E = E \setminus \{e_i\}$
7 $\quad\quad\quad UpdateCFreeSpace(e_i)$
8 $\quad\quad\quad \sigma_{opt} \leftarrow Optimize(e_i)$
9 $\quad\quad\quad$ **if** $IsCollisionFree(\sigma_{opt})$ **then**
10 $\quad\quad\quad\quad E = E \cup \{\sigma_{opt}\}$
11 $\quad\quad\quad$ **else**
12 $\quad\quad\quad\quad UpdateCFreeSpace(\sigma_{opt})$
13 $\quad\quad\quad UpdateShortestPathTree(G)$
14 $\quad\quad\quad Break$
15 **until** $e_i \in \mathbb{X}_{free}, \forall e_i \in E_{solution}$

the optimizer makes the given initial trajectory to converge within the configuration space covered by $V_{q_{from}} \cup V_{q_{to}}$, which is found to be reasonable choice for faster optimization in practice.

Back to the pesudocodes of Dancing PRM*, $UpdateShortestPathTree(\cdot)$ (Line: 12 in Alg. 4 and line: 13 in Alg. 6) is invoked when a graph restructuring such as edge insertion or deletion occurs in order to maintain the shortest solution path from q_{init} to q_{goal} over the search graph G ([56]) which is necessary for anytime lazy collision checking in PRM*.

Finally, $CheckSolutionPath(\cdot)$ is terminated (Line: 15) when it yields a feasible solution path after validating the best-so-far provisional solution path by lazy collision checking.

3.3.4 Batch Dancing Tree*

The integration of BIT* with our proposed algorithm can be viewed as a variation of RABIT* ([53]), where the original local optimizer is replaced by ours with the configuration free space approximation. The benefit of BIT* is to use batches of samples and Lifelong Planning A* -based graph search ([54]), which results in an efficient lazy graph expansion.

In this algorithm, we consider BT* (Batch Tree) as a simplified version of BIT* ([62]) without the informed sampler to focus on the sampling-invariant analysis.

The pseudocode of the proposed algorithm, BDT* (Batch Dancing Tree), is depicted in Alg. 7, where the highlighted lines indicate modified ones from RABIT*. BDT* maintains two disjoint vertex sets: V, the actual vertex set in graph $G = \{V, E\}$, which is a tree rooted at q_{init}. Also, $X_{samples}$ is a set of samples not connected to G, which contains samples to be inserted into G.

Unlike $Sample(\cdot)$ in Alg. 4, $SampleCollision\,Free(m_{batch})$ randomly samples $m_{batch>0}$ of collision-free configurations at once. BDT* then incrementally expands its search graph by LPA* (Line: 9-10) in increasing order of a heuristic cost, $g_G(v) + \hat{h}(v)$ for a vertex $v \in Q_V$ and $g_G(v) + (\widehat{|(v,x)|}) + \hat{h}(x)$ for an edge $(v,x) \in Q_E$.

In the above equation, $g_G(v)$ represents a cost-to-come to v over the search graph G, which is the cost of the shortest path in G connecting q_{init} to v. $|(v,x)|$ is a positive value of motion cost from v to x, and it has ∞ if (v,x)

Algorithm 7: BATCH DANCING TREE*

1 $V \leftarrow \{q_{init}, q_{goal}\}$; $E \leftarrow \emptyset$
2 $X_{samples} \leftarrow \{q_{goal}\}$;
3 $Q_V \leftarrow \emptyset$; $Q_E \leftarrow \{(q_{init}, q_{goal})\}$
4 **while** *Termination condition is not satisfied* **do**
5 **if** $Q_V \equiv and\ Q_E \equiv \emptyset$ **then**
6 $X_{samples} \leftarrow SampleCollisionFree(m_{batch})$
7 $V_{\forall x \in X_{samples}} \leftarrow \emptyset$
8 $Q_V \leftarrow V$
9 **while** $BestQueueValue(Q_V) \leq BestQueueValue(Q_E)$ **do**
10 $v_{best} \leftarrow BestInQueue(Q_V)$
11 $Q_{near} \leftarrow Near(v_{best}, V)$; $X_{near} \leftarrow Near(v_{best}, X_{samples})$
12 **if** $IsNew(v_{best})$ **then**
13 Insert Q_{near} to $V_{v_{best}}$, and X_{near} to $V_{v_{best}}$
14 Insert v_{best} to $V_{q \in Q_{near}}$, and v_{best} to $V_{x \in X_{near}}$
15 $ExpandVertex(v_{best}, Q_{near}, X_{near})$
16 $ProcessEdge(BestInQueue(Q_E))$
17 **return** $SolutionPath(G)$

contains any collision. $\hat{h}(v)$ stands for an admissible estimation of cost-to-go from v to q_{goal} and $(\widehat{|(v,x)|})$ is that of the motion cost value.

$ProcessEdge(\cdot)$ in Alg. 8 considers three constraints (Line: 2, 7, 8) to minimize the number of edge collision checking and the cardinality of the edge set E. To be specific, it determines whether to invoke an explicit collision checking followed by our local optimization for an edge $e \in Q_E$ or reject.

In these inequality conditions, $\hat{g}(v)$ stands for an admissible estimation of cost-to-come from q_{init} to v. These constraints are checked to prune edges that cannot improve the current solution even further.

For the construction of $\mathbb{X}_{free}$, three subroutines; $Sample(\cdot)$, $ProcessEdge(\cdot)$ and $ExpandVertex(\cdot)$ related to collision checking are changed internally according to List. 3.1 so that any empirical collision found by an explicit collision checking is propagated to near configurations in V.

To summarize the significant differences between Dancing PRM* and BDT*, the first thing is the way to sample configurations (1 sample vs. m_{batch} per batch), and how to compute a path in G (DSPT vs. LPA*). The other processes such as the configuration-free space approximation and optimization are applied identically. We also expect that the other planners sharing the basic skeleton of a generalized sampling-based planner can be integrated with our *Dancing* part, i.e., local optimization with the configuration-free space approximation.

We discuss how the graph expansion procedure in BDT* differs from that in lazy PRM* in Sec. 3.4.4 to analyze their properties.

Algorithm 8: PROCESSEDGE ($(v,x) \in Q_E$), $v \in V, x \in X_{samples}$.

Pop (v,x) from Q_E
if $g_G(v) + (\widehat{|(v,x)|}) + \hat{h}(x) < g_G(q_{goal})$ **then**
  if $IsCollisionFree((v,x))$ **then**
    $\sigma(v,x) = (v,x)$
  else
    $\sigma(v,x) = Optimize((v,x))$
  if $IsCollisionFree((v,x)) \wedge \hat{g}(v) + |\sigma(v,x)| + \hat{h}(x) < g_G(q_{goal})$ **then**
    if $g_G(v) + |\sigma(v,x)| < g_G(x)$ **then**
      $E \leftarrow E \cup \sigma(v,x)$
      Update $V, X_{samples}$ and Q_E
else
  $Q_V \leftarrow \emptyset$; $Q_E \leftarrow \emptyset$

3.4 Experiments

3.4.1 Experiment setup

For fair comparison, all the tested methods are built upon the same proximity subroutines such as discrete collision detection and the nearest neighbor search available in OMPL (Open Motion Planning Library, [63]).

For visualization and framework integration, we use V-REP simulator ([64]). For near neighbor search, $Near(\cdot)$, k-nn is used with a parameter $\gamma = 1.1$ and the CHOMP-based optimizer works with parameters of $\lambda = 1$, $\varepsilon = 10^{-3}$, $\mu = 2.0$, $z = 10$, $\zeta = 0.3$ and $i_{max} = 10$, which are applied identically to RABIT* and our methods. The reported results are averaged over 30 trials.

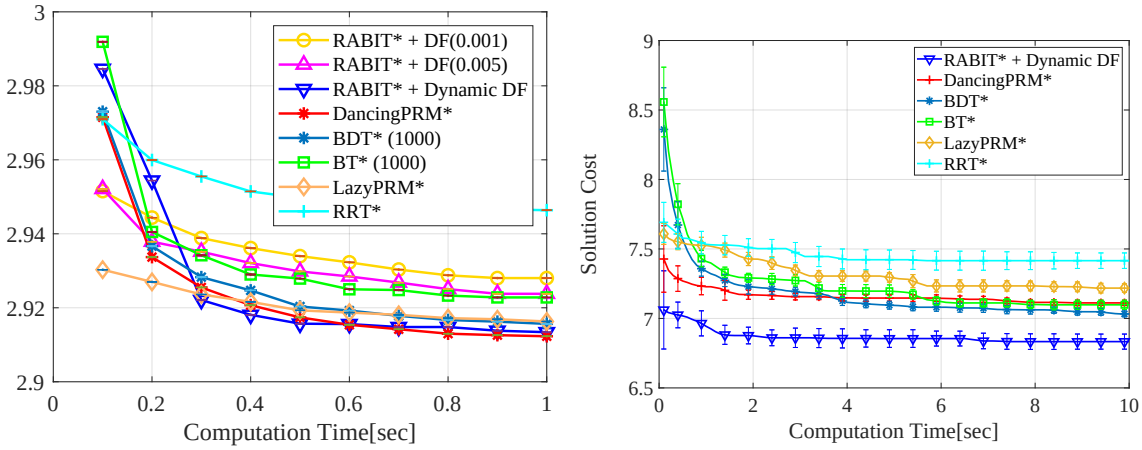
3.4.2 Comparison against RABIT*

We first compare the performance of DancingPRM* and BDT* against RABIT*, which is applicable only when the explicit representation of $\mathbb{X}_{free}$ is available, i.e., the analytic computation of obstacle potential is available.

Since our approaches do not assume such an environment, we cannot say that the direct comparison in this setting is completely fair. Nonetheless, we would like to show how our method works, even in these cases, to show the difference against RABIT*.

For this test, our evaluation benchmark is constructed by following the ones used in the original paper of RABIT*. Specifically, we consider two synthetic scenes, which are $\mathbb{R}^2$ and $\mathbb{R}^8$ configuration spaces where a wall with 10 narrow passages are located at the center in a d-dimensional hypercube of a width 2, i.e. $[-1,1]^d$. The boundary of the configuration obstacle space is created to be axis-aligned for easy computation of $c(\cdot)$ and $\nabla c(\cdot)$. A pair of input configurations (q_{init}, q_{goal}) are set to ($[-1,...,-1]$, $[1,...,1]$). This benchmark is, therefore, designed to have multiple difficult-to-sample homotopies of solution paths. We consider a point robot as an agent, its workspace and the configuration space are therefore identical.

Fig. 3.6 shows our experimental results of the solution cost as a function of computation time. The value in the parentheses of BDT* means the number of samples per batch and the value is chosen by experiments with a range of 1 to 1000 batch sizes. We also discuss a performance comparison of BT* and BDT* with different batch sizes in Sec. 3.4.4. Under our experimental setting, "RABIT* + DF(α)" uses a d-dimension distance field with



(a) $\mathbb{R}^2$ (b) $\mathbb{R}^8$

Figure 3.6: Performance comparison over computation time for different algorithms in synthetic benchmarks. Results are averaged over 30 trials. In $\mathbb{R}^8$, RABIT + DF (Distance Field)s are not reported since the construction of the distance field with a reasonable resolution is intractable in a high-dimensional space.

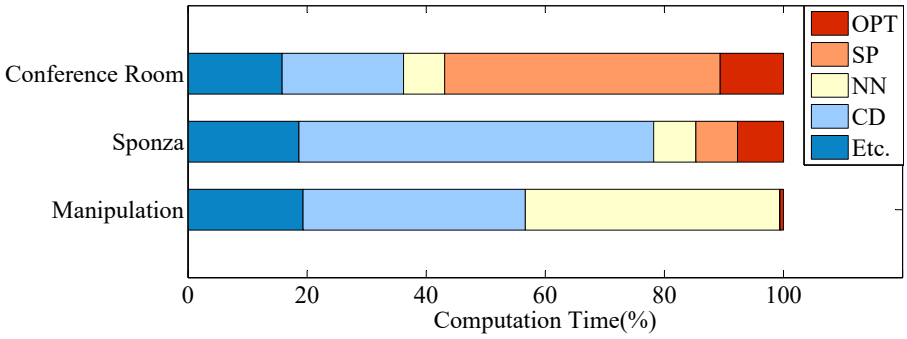


Figure 3.7: Computation time breakdown of the proposed algorithm DancingPRM* measured in our benchmarks (Fig. 3.8). Each abbreviation in legend stands for OPTimization (OPT), Shortest Path tree update (SP), Nearest Neighbor search (NN), and Collision Detection (CD). Note that the computation time for $\widetilde{\mathbb{X}}_{free}$ construction is negligible ($< 1\%$) for all of three benchmarks.

a resolution of α, given as a *priori* and "RABIT* + Dynamic DF" computes obstacle potentials analytically on the fly. The runtime computation of obstacle potential is possible because, for a configuration x_{obs} in $\mathbb{X}_{obs}$, the minimum distance to $\mathbb{X}_{free}$ from x_{obs} is a distance to the closest face of the obstacle containing x_{obs}, which is even computationally lighter with axis-aligned obstacles.

Fig. 3.6(a) shows that RABIT* with a dynamic distance field outperforms than those with precomputed distance fields because the latter with a fixed resolution makes the optimization process more conservatively, i.e., yielding longer trajectories that are farther from the boundary of $\mathbb{X}_{obs}$ on a given α. Even in this case, our method shows comparable performance to "RABIT* + Dynamic DF" due to the local optimization with our configuration free space approximation. Although the performance gap between the best and the worst result seems to be only $1-2\%$. However, when it comes to the convergence speed toward the optimum, we can notice the visible improvement. For instance, the solution cost of Lazy PRM* at 1s can be achievable by Dancing PRM* much earlier; around at 0.5 sec, which is two times faster in terms of the convergence speed.

In $\mathbb{R}^8$, it takes a huge amount of time and memory to construct a distance field with a reasonable resolution.

We therefore only report "RABIT* + Dynamic DF" and other planners. In $\mathbb{R}^8$ case, we can observe that our approaches, both DancingPRM* and BDT* show lower performance than RABIT* + Dynamic DF.

When we look at the performance improvement between our methods (Dancing PRM* and BDT*) and their base methods (lazy PRM* and BT*), we can see that our approaches show meaningful improvement. For the high-dimensional case, DancingPRM* shows up to 10x faster convergence speed over lazy PRM* in the given time budget. At the same setting, BDT* shows 2x improvement over BT*. Our local trajectory optimization can be considered a *biased trajectory sampling* guided by our configuration-free space approximation. We can thus find the benefit from the fact that the local optimizations are only performed at where the conventional local planner failed to generate a feasible one, which improves the probability to find a feasible one in a difficult-to-sample region.

This experimental result, however, can be seen negative for the proposed algorithms; especially, $\mathbb{R}^8$ seems to show our limitation in a high-dimensional problem. In practice, however, we do not know the exact configuration free space, and the obstacle potential computation in runtime, requires heavy computational overhead as well.

Regardless of the computational overhead, RABIT + Dynamic DF is supposed to get exact obstacle potential values, which can be computed as a vector from a configuration $v \in \mathbb{X}_{obs}$ to the closest face of the obstacle containing v. Meanwhile, ours only depends on the samples to approximate arbitrary configuration-free spaces.

For these reasons, beyond the theoretical aspect, we further evaluate the proposed algorithms with more general benchmarks, which RABIT* could not handle directly due to the complexity of the configuration space in the next subsection.

3.4.3 Comparison in practical benchmarks

In this experiment we compare DancingPRM* and BDT* against other asymptotic optimal planners, lazyPRM*, BT* and RRT*.

Fig. 3.8(a), 3.8(c), 3.8(e) show comparison results tested in our benchmarks illustrated in Fig. 3.8(b), 3.8(d) and 3.8(f), where the configuration space are $\mathbb{R}^2$, $SE(3)$ and $\mathbb{S}^6$, respectively. Fig. 3.7 shows a computation time breakdown of DancingPRM* measured in our benchmark.

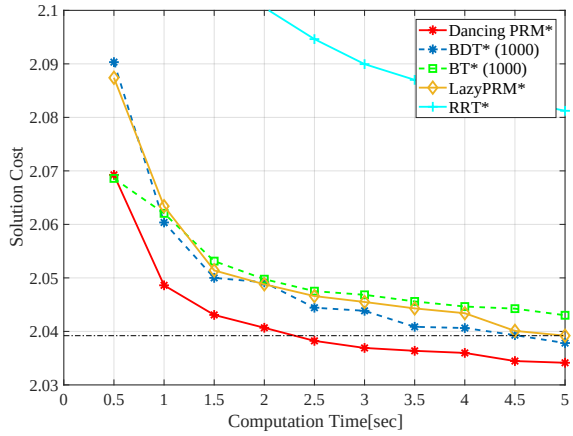
The benchmark set contains both easy-to-find homotopies and difficult-to-sample optimal homotopy of a solution path. Throughout all of three benchmarks, our approaches, i.e., local trajectory optimization with configuration free space approximation, not only improve the performance by optimizing a solution path in a specific homotopy but also help to identify a better homotopy earlier than other tested planners. Moreover, the light computation of our $\widetilde{\mathbb{X}}_{free}$ makes the proposed algorithm accomplish the entire process in runtime without priori space information, while providing a better result against other tested algorithms.

The 6-DOF of Figs. 3.8(d) and 3.8(f) has a relatively higher computation cost for collision checking as observed in Fig. 3.7. Especially, RRT* without lazy collision checking shows inferior performance in such cases, and we thus only report the results of planners with a lazy collision checking.

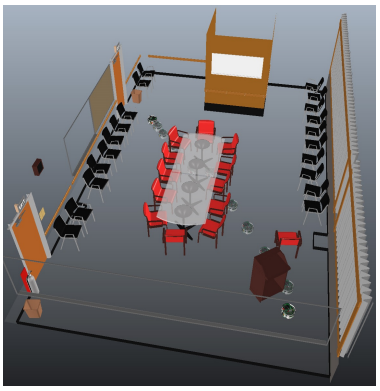
The proposed algorithms, both Dancing PRM* and BDT* show better performance over the other tested methods across different benchmarks as shown in the experiment results, even with the overhead of free space approximation and optimization-based local planning.

On top of that, the proposed algorithms particularly tend to have a lower variance compared to their original forms, BT* and Lazy PRM* in Fig. 3.8(c) and 3.8(c).

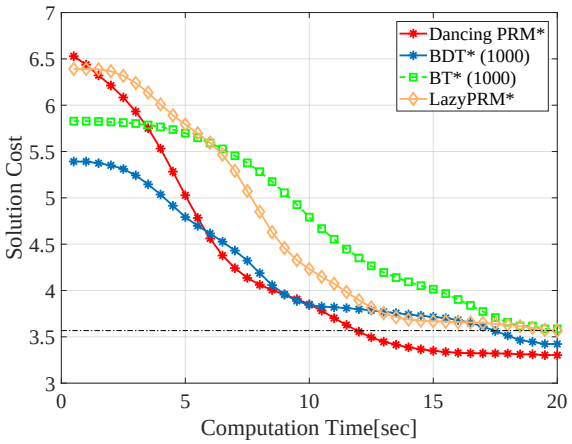
This is mainly attributed by the enhanced exploitation in a difficult-to-sample area by our local trajectory optimization. The proposed algorithms are capable of generating more connections between configurations; as a result, the connectivity which can improve not only the convergence speed but also stability of the solution cost



(a) Performance vs. time

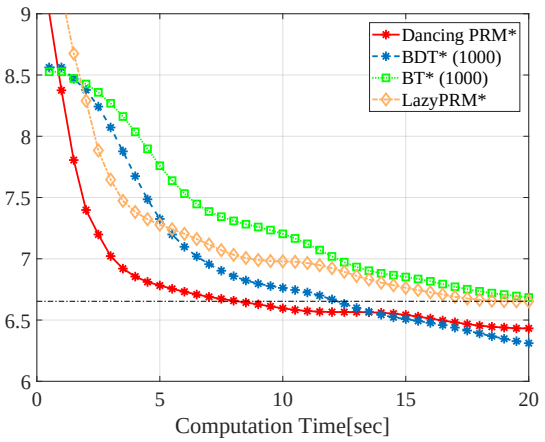


(b) Conference Room : $\mathbb{R}^2$

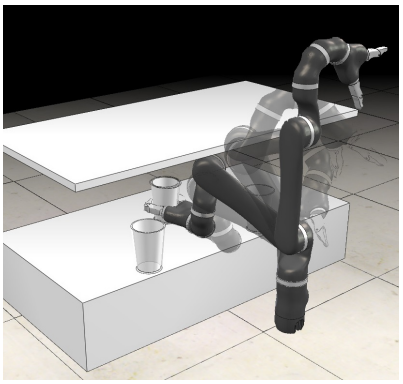


(c) Performance vs. time

(d) Sponza : $SE(3)$



(e) Performance vs. time



(f) Manipulation : $\mathbb{S}^6$

Figure 3.8: Performance comparison over computation time (left) and the given environment (right). The plots show the performance of asymptotic optimal planners tested in our benchmark. The horizontal black dotted line shows the best solution cost achieved by algorithms without our local trajectory optimization. Figures on the right side are visualization of benchmarks. Results are averaged over 30 attempts, and RRT* is not reported for Fig. 3.8(c) and 3.8(e) due to a high performance gap.

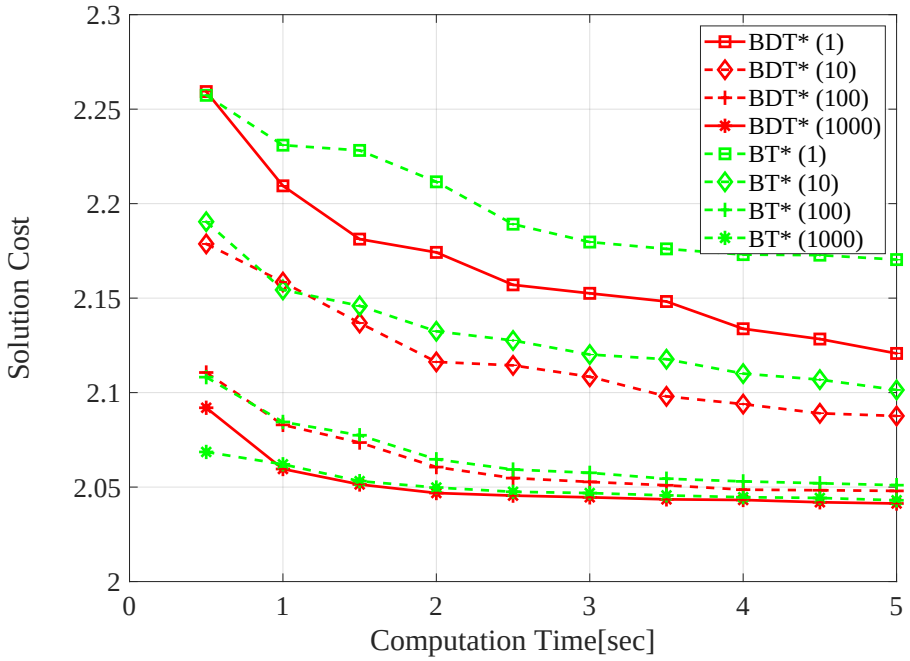


Figure 3.9: Performance comparison with varying the number of samples per batch in BDT*. The results are averaged over 30 attempts and tested in Fig. 3.8(b). A higher number of samples per batch provides a better convergence speed at the expense of anytime property.

with a lower variance.

Last but not least, there are no substantial overheads on our configuration-free space approximation in the nature of sampling-based planning is another benefit of our algorithm.

3.4.4 Comparisons with varying batch sizes

In this work, we presented BDT*, which is the integration between the core of Dancing PRM* and LPA*-based graph expansion with batch sampling, originally presented in [54]. For better understanding of BDT*, we discuss how BDT* and Dancing PRM* work differently and the effect of batch sizes in this subsection.

In Fig. 3.9, we measure the performance with varying batch sizes from 1 to 1000 in the $\mathbb{R}^2$ benchmark (Fig. 3.8(b)). As we can observe, there is a performance improvement as the number of samples per batch increases. Also, the core of Dancing PRM*, i.e., local trajectory optimization with configuration free space approximation, successfully improves the convergence speed toward the optimum even with BT*. We can observe the computation time breakdown over varying batch sizes in Fig. 3.10. Compared to Fig. 3.7, the difference can be summarized by follows:

(1) In Dancing PRM*, so-called *optimistic thrashing* ([56, 49]) poses an additional overhead on shortest path computation (SP). Its effect is maximized in the 2D benchmark (Fig. 3.8(b)) due to the cluttered environment with narrow passages around the optimal homotopy.

(2) BDT* spends more time on collision checking, most of which are on edge collision checking.

(3) In BDT*, the local trajectory optimization (*OPT*) becomes the majority as the batch size increases. This is because LPA*-based expansion in BDT* requires more edges to be checked for collision, and the compu-

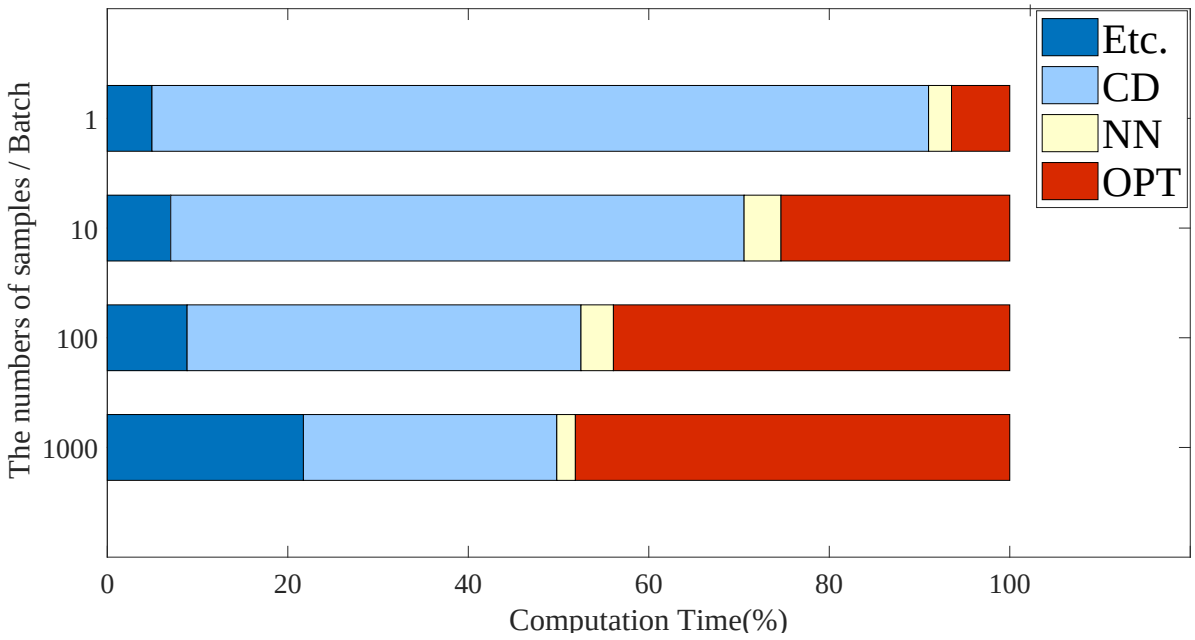


Figure 3.10: Computation time breakdown of Batch Dancing Tree* tested in Fig. 3.8(b). The result shows the distribution of computation time spent on each component over varying batch size. Note that the majority of *Etc.* is a graph expansion based on LPA*, which corresponds to *SP* in Fig. 3.7. The overhead for the configuration free space approximation is measured $< 3\%$ in this experiment.

tation overhead of local trajectory optimization is identical regardless of the length of the trajectory unlike collision checking.

The most important point is that the use of DSPT (Dynamic Shortest Path Tree) gives rise to the optimistic thrashing problem, causing a more overhead on the shortest path computation. Meanwhile, LPA*-based expansion requires more edge collision checking than the DSPT-based approach, because the edges satisfying the constraints at line 2 in Alg. 8, i.e., $g_G(v) + \widehat{|(v,x)|} + \hat{h}(x) < g_G(q_{goal})$ are checked for collision, where $\hat{}$ stands for an admissible estimation.

On the other hand, DSPT-based lazy collision checking is performed only for the edges on the solution path, thus it has a much stronger condition, i.e., $g_G(v) + |(v,x)| + h_G(x) = g_G(q_{goal})$, where $h_G(x)$ is a cost-to-go from x to q_{goal} over G.

Since both $\widehat{|(v,x)|} \leq |(v,x)|$ and $\hat{h}(x) \leq h_G(x)$ hold, DSPT-based approach is designed to require a less number of edges for collision at the expense of optimistic thrashing. Likewise, as the more edges reach at line 6 in Alg. 8, the portion of optimization overwhelms those of other components in terms of computation time.

Note that while we report the case of $\mathbb{R}^2$ only here, its tendency is similar throughout all of the three difference benchmarks.

3.5 Analysis

In this section, we discuss various properties of the proposed methods. We first discuss the asymptotic optimality and then the time complexity of the computational overhead induced by the configuration free space approximation and optimization-based local planning.

3.5.1 Almost-sure Asymptotic Optimality

Let $E_{proposed}$ and $E_{lazyPRM^*}$ refer to the valid edges in a graph constructed by the proposed DancingPRM* and lazy PRM* ([56]), respectively; two vertex sets of $V_{lazyPRM^*}$ and $V_{proposed}$ are defined in a similar way. Without loss of generality, we assume that a sequence of random samples and subroutines in both planners are identical.

As described with Alg. 6, the proposed method never rejects any edge $e \in E_{lazyPRM^*}$, because the path validation in the proposed algorithm is identical to that in lazy PRM* except for the additional trajectory optimization. Instead, the proposed algorithm optimized and refined an edge that was initially identified to have collisions. Accordingly, $E_{proposed}$ could rather contain more number of edges than $E_{lazyPRM^*}$ (Line: 10 in Alg. 6). Therefore, $E_{lazyPRM^*} \subseteq E_{proposed}$ holds.

On the other hand, the vertex set is identical, because no modification is applied to the sampling and collision checking on q_{sample} as shown in Algs. 3 and 4, therefore, $V_{lazyPRM^*} = V_{proposed}$. Consequently, if we compute a solution path on $G_{proposed} = \{V_{proposed}, E_{proposed}\}$, the optimality of the proposed algorithm follows that of lazy PRM*, which was proven almost-sure asymptotically optimal in [56].

For BDT*, it considers more edges due to the weak condition for local planning as discussed in Sec. 3.4.4, thus $E_{lazyPRM^*} \subseteq E_{BDT^*}$ hold. As a result, it also follows the asymptotic optimality of lazy PRM*. While both of our methods, Dancing PRM* and BDT* generate more edges compared to Lazy PRM* and BT*, respectively, our methods have shown faster convergence speed. This is because that our local trajectory optimization integrated with sampling-based planning, efficiently exploits difficult-to-sample homotopies. Our configuration free space approximation, which is learned in the nature of sampling-based planning, also guides the optimization process, therefore the resulting algorithms are free from additional heavy computational proximity calculation and a precomputed knowledge on the environment.

3.5.2 Computational Complexity

The complexity and analysis of primitive operations used for our method follow the discussion in [65]. We thus deal with overheads introduced mainly by the proposed algorithms in this section.

Time Complexity In Alg. 4, we added various steps for $\widetilde{\mathbb{X}}_{free}$ construction. First of all, updates of V_q for the configuration free space approximation linearly increase as the number of elements to be inserted increases, because it does not have to be an ordered structure. We also have $O(|Q_{near}|)$ of iterations witness propagation (Sec. 3.3.2) ; thus the time complexity of the entire while loop in Alg. 4 and the while loop for $ExpandVertex(\cdot)$ in Alg. 7 are dominated by that of $Near(\cdot)$, i.e., $O(\gamma^d \cdot 2^d \cdot log(|V|))$, which is identical to the expected cardinality of Q_{near}.

We perform an additional nearest neighbor search for $q_{sample} \in \mathbb{X}_{obs}$ (Line: 15 in Alg. 4 and 6 in Alg. 7), which is proportional to $log(|V|) \cdot \mathfrak{L}(\mathbb{X}_{obs})$, where $\mathfrak{L}(\cdot)$ is a Lebesgue measure, i.e., the hypervolume of $\mathbb{X}_{obs}$ It is an additional overhead compared to lazy PRM* depending on the volume of the configuration obstacle space, since conventional PRM* and lazy PRM* reject the sample configuration without nearest neighbor search.

Note that RRG also performs a nearest neighbor search for every sample configuration; Dancing PRM* has no additional overhead in sampling phase compared to RRG ([6]).

The lazy collision checking is performed in both Dancing PRM* and BDT* in Alg. 6 and 8, respectively. On top of the procedures for lazy collision checking in these functions, we additionally perform $Optimize(\cdot)$ for optimization-based local planning. Its computational overhead with modified obstacle potentials defined in Eqs. 3.5 and 3.8 can be expressed as $i_{max} \cdot z \cdot C(\mathfrak{D}(\cdot))$, where i_{max} is the maximum iteration of optimization, z the number of discretized intermediate nodes, and $C(\mathfrak{D})$ the computational complexity for $\mathfrak{D}(\cdot)$ calculation. To

compute $\mathfrak{D}(\cdot)$, subsets of V_u and V_v associated with the two end points of ξ for an edge (u,v) should be considered. Its computational cost can be bounded by $O(\gamma^d \cdot 2^d \cdot log(|V|))$, the expected cardinality of Q_{near}. The number of $optimize(\cdot)$ calls is, however, remarkably reduced by lazy collision checking in practice.

According to the above analysis, we can conclude that there is no substantial overhead in terms of the time complexity, since the overhead for the configuration free space approximation is dominated by that of $Near(\cdot)$, thanks to our decentralized storage in Sec. 3.3.2.

The analysis for the local trajectory optimization is somewhat tricky since the number of local planning affects the complexity of optimization, as shown in Fig. 3.7 and 3.10. For this reason, its complexity heavily depends on the cost of the solution path, $g_G(q_{goal})$, during the execution, which is the bounds for the edge expansion as discussed in Sec. 3.4.4.

Memory Complexity The proposed algorithm additionally maintains the near neighbor set, $V_v, \forall v \in V$ in Sec. 3.3.2, whose cardinality gradually increases as the number of samples goes higher. The memory overhead can be estimated as the sum of cardinality, $\sum_{v \in V} |V_v|$, which is $\Omega(\gamma^d \cdot 2^d \cdot log(|V|) \cdot |V|)$ since each V_v contains a cumulative near neighbor set over the iterations. A possible way to alleviate the complexity i.e., reduction of $\Omega(\cdot)$ to $O(\cdot)$ is introduced in [61] where the planner culls out neighbors outside the ball of radius r centered at v, where r for r-nn search (Sec. 3.2.1).

3.5.3 Configuration free space approximation

Our configuration free space approximation $\widetilde{\mathbb{X}}_{free}$ is constructed entirely during the sampling process in an efficient manner. Nonetheless, minimizing the approximation error as low as possible is also important for the better optimization performance.

In this subsection, we show the benefit of our witness propagation and the radius compensation method explained in Sec. 3.3.2 by measuring the error of our approximate free space $\widetilde{\mathbb{X}}_{free}$ against the inexact version of the ball-tree algorithm ([31]).

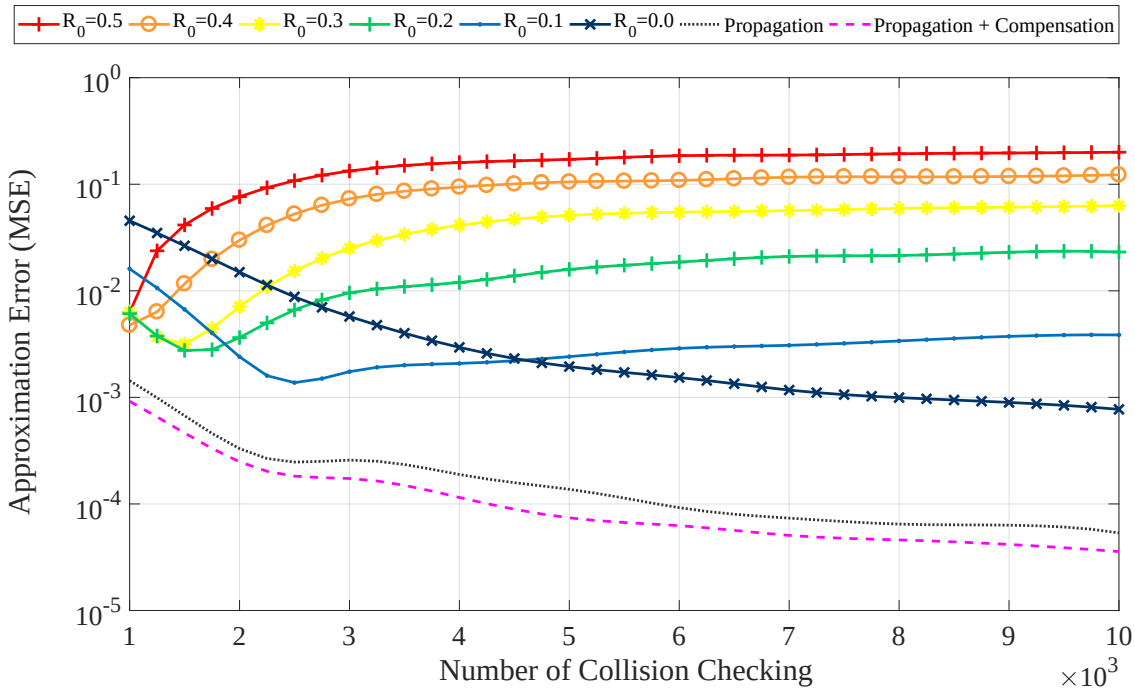
Both algorithms share the representation and the skeleton of approximation procedures, but the ball-tree algorithm initializes the radius of a new sample configuration with a large value, R_0.

For the ease of the experiment setup, we use the synthetic scenes previously tested in the performance comparison against RABIT* (Sec. 3.4). We compare the mean squared error of the distance value computed with $\widetilde{\mathbb{X}}_{free}$, i.e., $\mathfrak{D}(x)$ giving the minimum distance to the closest $\widetilde{\mathbb{X}}_{free}$ from a configuration x, which is defined in Sec. 3.3.3. Note that both *Propagation* and *Propagation + Compensation* methods are run on Dancing PRM*, and the errors are measured at the intermediate configurations during the local trajectory optimization.

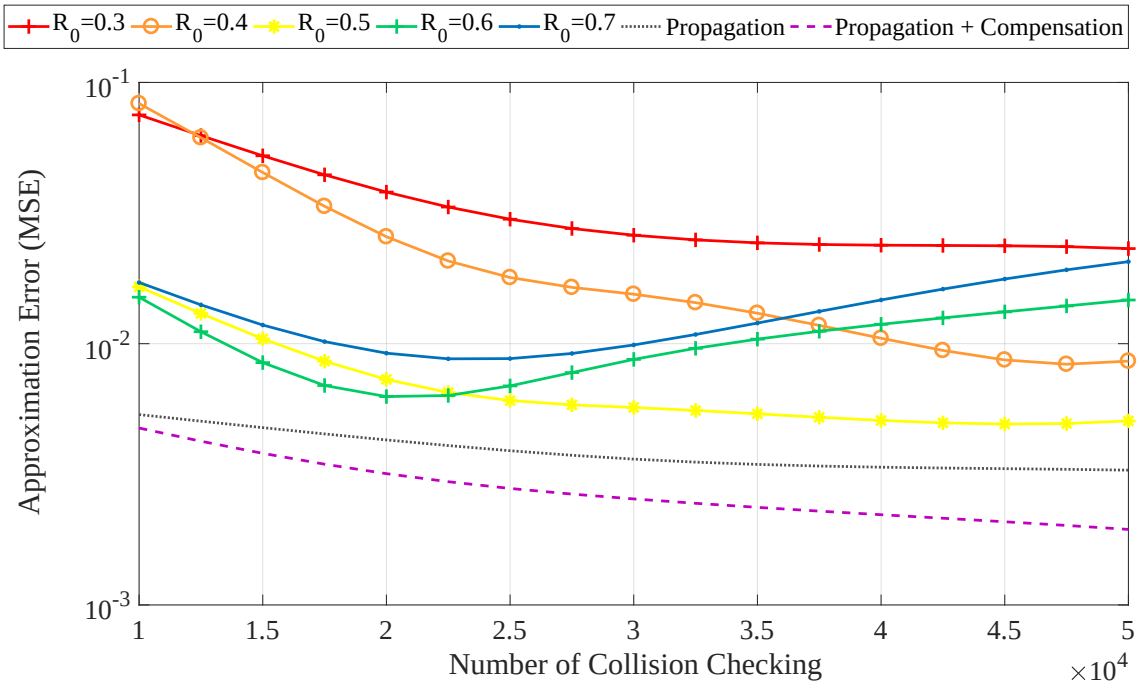
Fig. 3.11 shows the approximation error as a function of the number of collision checking with more iterations. In the plot legends, as defined in the original ball-tree algorithm, different methods of $R_0 = r_{init}$ set their initial radius r_x of an approximate collision free hypersphere centered at a new sample configuration x with r_{init}.

There is a noticeable observation that $Optimize(\cdot)$ in Alg. 6 makes the majority of x lie in $\mathbb{X}_{obs}$, especially on the boundary of $\mathbb{X}_{obs}$. In this sense, this test is *biased* in terms of the sample distribution, although it apparently reflects configurations that matter most in the nature of Dancing PRM*. Accordingly, Fig. 3.11 should not be misinterpreted to indicate the convergence of $\widetilde{\mathbb{X}}_{free}$ toward $\mathbb{X}_{free}$.

Back to our experiment results, we can observe that prior methods initialized with different fixed values suffer from higher errors. This is mainly because newly sampled configurations have relatively insufficient information on the given environment, and they are initialized regardless of the empirical collision information accumulated, especially its near neighbors that can be accessible without having any additional overhead in the nature of sampling-based motion planner.



(a) 2D



(b) 8D

Figure 3.11: Mean squared errors measured at intermediate configurations during the local optimization by both our configuration free space approximation and the ball-tree algorithm with different initial R_0 values. Ours shows a monotonic decreasing result, while the ball-tree with fixed initial radii tends to converge at a particular value depending on the initial value. These plots use a logarithmic scale for the y-axis and a linear scale for the x-axis.

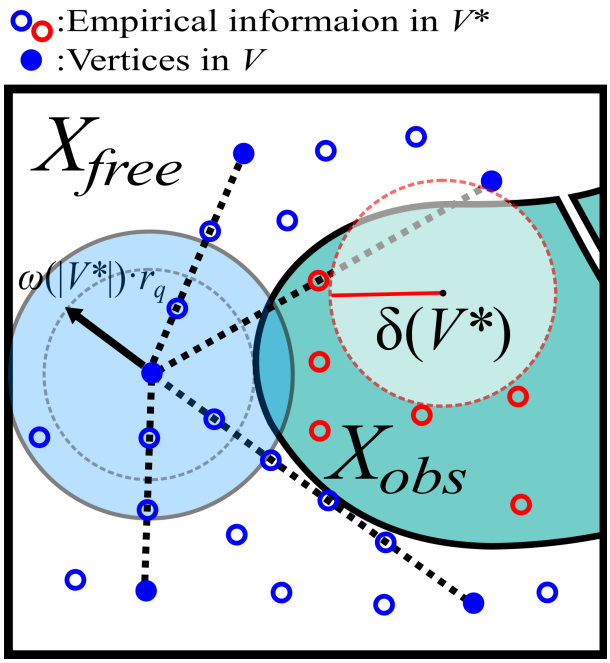


Figure 3.12: A concept of dispersion for the sample set V^* and the radius compensation. V^* consists of all sampled configurations in both $\mathbb{X}_{free}$ (blue) and $\mathbb{X}_{obs}$ (red) regardless of feasibility, and including samples for edge collision checkings (dotted segments). The radius of the red dotted circle on the right side stands for a dispersion of V^*, which is used as an approximate sparsity of V^*. The radius of the left circle represents a compensated radius (grey-dotted smaller circle), i.e., $\omega(|V^*|)\cdot r_q$, which is shrunk from the original radius r_q (grey-solid).

In the 2-dimensional case, we can guess the best R_0 would be 0 in terms of the asymptotic error as shown in the Fig. 3.11(a). This is because $\mathfrak{D}(x)$ for an arbitrary configuration $x \in \mathbb{X}_{obs}$ asymptotically converges to zero. We also cannot say that any specific value of R_0 outperforms every other options through the entire execution. For instance, the reasonable choice can be far from $R_0 = 0$ depending on a given environment in practice.

In contrast, the result of R_0 in the 8-dimensional case shows that we achieve the lowest approximation error at $R_0 = 0.5$, which makes the parameter tuning complicated in practice. Meanwhile, the combination of our witness propagation and radius compensation provides better accuracy. In addition to that, radius compensation reduces the error consistently throughout the execution of planning process, demonstrating the robustness experimentally.

3.5.4 Radius compensation and dispersion

The purpose of radius compensation (Sec. 3.3.2) is to improve our approximation model further with a limited amount of empirical collision information. We have shown its benefit experimentally in the previous subsection, and we discuss it more deeply for better understanding of our approximation process.

Radius compensation is based on the concept of *dispersion*, which is also related to the optimal threshold for nearest neighbor search in sampling-based motion planners ([66]). Dispersion of a finite sample set P in a metric space $(\mathbb{X}, \rho)$ for a sample space $\mathbb{X}$ and a metric ρ, is defined as the following equation:

$$\delta(P) = \sup_{x \in X} \{\min_{p \in P} \{\rho(x, p)\}\}. \tag{3.10}$$

The above equation can be interpreted as an expected radius of the largest empty hypersphere in the given space. For the sake of simplicity, we assume that the sampling distribution is uniform, even though the actual sampling

distribution for the empirical collision is non-uniform due to the the behavior of collision detector (e.g., bisection or linear) and especially the lazy collision checking.

The asymptotic bound of dispersion, i.e., $\delta(|P|) = O(\frac{log(|P|)}{|P|}^{1/d})$, was originally introduced by [67] and [68]. It also has been studied in the perspective of sampling-based motion planning ([69, 6]) for the threshold of near neighbor search to guarantee the almost-sure asymptotic optimality. In our work, we consider the dispersion to be a sparsity or an expected approximation error bound of V^*, a set of samples checked for collision in the given space X. As explained in Sec. 3.3.3, we presented a radius compensation for the local trajectory optimization, i.e.,

$$\omega(|V^*|) \cdot r_q, \text{ where } \omega(n) = 1.0 - \zeta \cdot \delta(n). \tag{3.11}$$

The multiplication between $\omega(\cdot)$ and the collision-free radius r_q therefore becomes the asymptotic error bound of our approximation, since the dispersion is the expected maximal gap between samples in V^*. Fig. 3.12 shows the concept of dispersion in an illustrative way. We use $\zeta = 0.3$ as explained in Sec. 3.4, which is chosen experimentally and provides the best performance regardless of the dimension of given problems.

Finally, the result of Fig. 3.11 provably shows that the radius compensation improves the accuracy of our configuration free space approximation with a negligible overhead as revealed in Fig. 3.7 and 3.10.

Chapter 4. Volumetric Tree*: Adaptive Sparse Graph for Effective Exploration of Homotopy Classes

4.1 Introduction

Path and motion planning problem is a fundamental research area in robotics and has been widely studied for autonomous systems with mobility and manipulability. Among various categories of planning algorithms, sampling-based approaches have attracted considerable attention thanks to its probabilistic completeness [70, 71] and almost-sure asymptotic optimality [6]. Their key concept is to construct a random geometric graph or tree to identify the connection of feasible motions in the configuration-free (C-free) space.

It has been well-known that when applied to practical problems, sampling-based motion planners require a considerable amount of computation cost to check collision for vertices and edges, especially for computing the optimal solution in high dimensions [56, 72]. To this end, there have been a plethora of researches to alleviate the overhead of collision checking. Hauser [56] proposed a lazy collision checking with DSPT (Dynamic Shortest Path Tree [58]) to delay the explicit checking until it is necessary, while preserving the asymptotic optimality and anytime properties. Bialkowski et al. [72] presented a graph associated with safety certificates, i.e., a set of collision-free balls to reduce the amortized complexity of collision checking. Gammell et al. [54] combined sampling-based motion planning and Lifelong Planning A* (LPA*) with a batch sampling technique. Their approach expands the graph using LPA*-style incremental search techniques, performing graph expansion and collision checking on a partial subset of graph components.

Meanwhile, optimization-based planners and the hybridization of sampling and optimization also has been suggested to get synergy in various manner. CHOMP (Covariant Hamiltonian Optimization for Motion Planning) [40], its variants [41, 42] and TrajOpt [73] formulate the trajectory optimization as a convex problem to minimize the local approximation of the objective function sequentially.

As a hybridization of sampling and optimization, Choudhury et al. [53] suggested RABIT* (Regionally Accelerated Batch Informed Trees), where a sampling-based planner runs as a global planner, and an optimization-based planner takes over the local planning. Kim et al. [74] proposed a similar hybridization, Dancing PRM*, whose obstacle potential computation is solely done in the configuration space. It directly approximates the C-free space by utilizing the empirical collisions found during the execution. Kuntz et al. [55] presented another hybrid algorithm, which locally refines solution paths using lazy interior point optimization to compute high-quality solutions quickly.

When it comes to a scalability issue, recent researches have also pointed out the significance of nearest-neighbor search [65, 75]. Kleinbort et al. [65] analyzed the asymptotic computational bottleneck in sampling-based motion planning. To be specific, the complexity of collision detection is proportional to that of the given workspace, while nearest-neighbor search asymptotically dominates the entire computation time as the number of samples goes to infinity. Varricchio and Frazzoli [75] proposed pruning techniques for the k-d tree to reduce the computational cost of nearest neighbor search.

Main Contributions. In this work, we present Volumetric Tree*, a hybridization of sampling-based and optimization-based motion planning for effectively exploring the homotopy class of solution paths and identifying local optimums in each homotopy class (Sec. 4.3.1). In volumetric tree*, the role of a sampling-based motion planner is used for a homotopy exploration. To be specific, it is designed to construct an adaptive sparse graph, where wide-

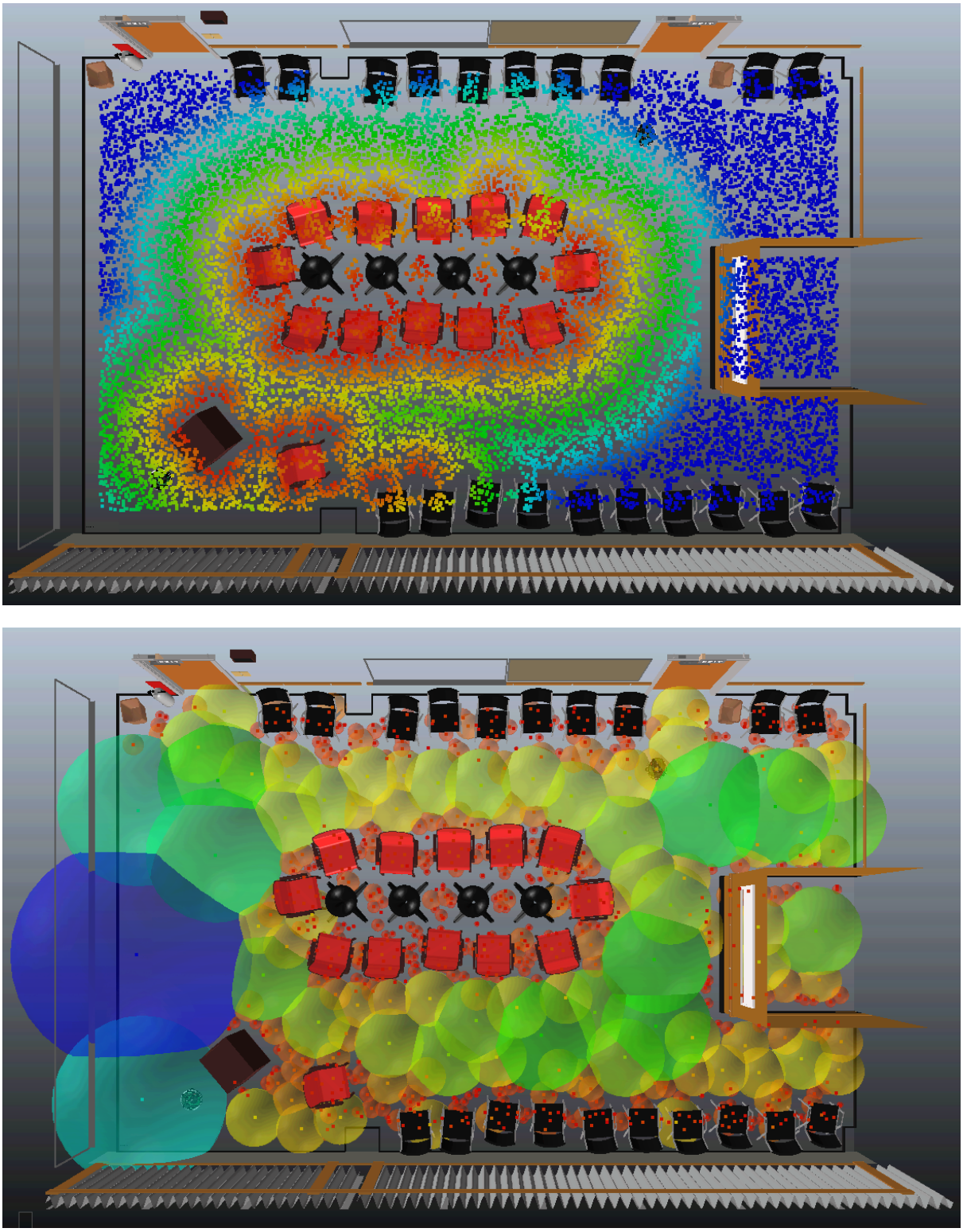


Figure 4.1: A heatmap-style visualization of the vertex set V, constructed by a conventional planner (top, $|V| = 14384$) and that of volumetric tree* (bottom, $|V| = 540$) in the same time budget. We can observe the volumetric tree* constructs a sparse graph, while capturing the samples around narrow passages or boundaries. The vertices close to the obstacles are encoded red; otherwise blue.

open areas are covered by a fewer number of volumetric vertices (hyper-spheres) while maintaining fine-grained vertices around a boundary of free space or narrow passage (Sec. 4.3.2), instead of constructing a dense graph. On top of that, we can represent a set of paths homotopic to each other into a sequence of volumetric vertices as a compact representation. To complement the coarse-grained paths computed on the sparse graph, we combine an optimization-based motion planner (Sec. 4.3.3) to refine the solution paths with a dedicated shortest path computation technique, dropout (Sec. 4.3.4). As a result, volumetric tree* efficiently identifies initial paths in multiple homotopy classes in a sampling-based manner for the optimization-based planning by complementing each other.

According to conducted experiments, we observe up to 3 times speedup over other tested methods in terms of convergence to the optimum on rigid body and manipulation planning problems (Sec.4.4). These results are mainly obtained by the adaptive sparse graph integrated with the optimization-based planning, which allows volumetric tree* to explore the configuration space efficiently.

4.2 Algorithmic Background

We first formulate the problem we would like to address and review preliminaries of sampling-based motion planning.

4.2.1 Problem Definition

Let $\mathcal{X} = \mathbb{R}^d$ be the configuration space, where d is the dimension of a given problem. Let the configuration-obstacle (C-obs) space be $\mathcal{X}_{obs}$, which is a set of states in collision with obstacles. The complement of $\mathcal{X}_{obs}$, $\mathcal{X}_{free}$ = $\mathcal{X} \setminus \mathcal{X}_{free}$, becomes the configuration-free (C-free) space. For a given pair of an initial and goal configurations, x_{init} and $x_{goal} \in \mathcal{X}_{free}$, respectively, let $\sigma \in \Sigma : [0,1] \rightarrow \mathcal{X}_{free}$ be a feasible (e.g., collision-free) path, where Σ is a set of all possible paths satisfying $\sigma(0) = x_{init}$ and $\sigma(1) = x_{goal}$.

The optimal motion planning problem then can be formulated as:

$$\sigma^* = \arg\min_{\sigma \in \Sigma} (c(\sigma)), \tag{4.1}$$

where $c(\cdot)$ is a cost function such that $c : \Sigma \rightarrow \mathbb{R}_{\geq 0}$.

For a sampling-based planner, it is said to be almost-sure asymptotic optimal if the probability that a path computed by the planner at an iteration number i, σ_i, converges to the optimal path is 1, as the number of iteration goes to infinity:

$$\mathbb{P}\Big[\lim_{i \rightarrow \infty} \big(c(\sigma_i) = c(\sigma^*)\big)\Big] = 1 \tag{4.2}$$

4.2.2 Sampling-based Motion Planning

There have been proposed various optimal sampling-based motion planning algorithm such as RRT*, PRM* [6], FMT* [76] and BIT* [54]. They sample a set of configurations, $\mathcal{X}_{sample} \in \mathcal{X}$, to construct a search graph $G = \{V, E\}$, where each vertex $v \in V$ represents a collision-free configuration and $e = (v, w) \in E$ is a collision-free motion for a pair of configurations v and $w \in V$.

V is initialized with $\{x_{init}, x_{goal}\}$ and E is set to empty, then we sample a random configuration at each iteration until the termination condition is satisfied, e.g., time limit or a maximum number of iterations.

For a sample configuration x_{sample}, the edge connection process is done with its near neighbors found by $Near(\cdot)$. $Near(x)$ returns a set of near vertices in V such that those vertices lie inside of an implicit ball centered at x with a connection radius r. This type of NN search is also called ε-NN search. k-nearest neighbor search can be

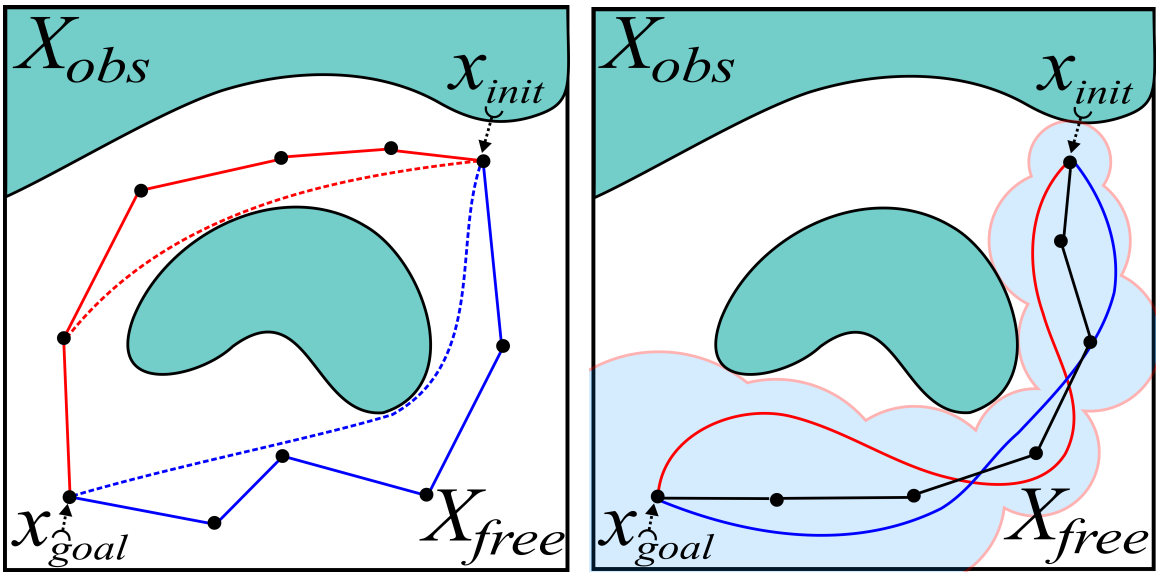


Figure 4.2: The left figure illustrates two solution paths, shown red and blue solid lines, in two different homotopy classes with their local optimal paths (dotted). The right figure shows a solution path (solid black) covered by a sequence of collision-free balls and paths homotopic to the solution path (red and blue). These observations suggest that such a coarse-grained graph can be a sufficient representation if we can optimize each path toward the local optimum.

used alternatively; in either case, both connection radius r and k are determined proportionally to the cardinality of V to achieve the almost-sure asymptotic optimality [6]. Each vertex and edge is then checked for a collision, and only valid components are inserted into G for the graph expansion. Finally, a path connecting x_{init} and x_{goal} becomes a discrete solution path $\sigma \in \Sigma$, and its cost $c(\sigma)$ can be computed by summing up the cost of all edges over the path.

4.3 Algorithm

4.3.1 Motivations

We would like to explain motivations of volumetric tree*, a hybridization of sampling and optimization-based planning, with the following two aspects. Firstly, Fig. 4.2(a) shows two exemplar paths shown in the solid lines that connect x_{init} and x_{goal} in different homotopy classes. Once a solution path is computed, we can apply a local optimal planner such as an iterative optimization-based planners [40] to refine the path toward the local optimal path (dotted lines in Fig. 4.2(a)) in the same homotopy class. Based on this observation, we suggest using a sampling-based motion planner, which guarantees probabilistic completeness, for a homotopy exploration to identify solution paths in all possible homotopy classes. On top of that, an optimization-based planner then locally optimizes the initial paths computed by the sampling-based planner toward the local optimum.

Once we realize the hybridization of sampling-based and optimization-based approaches, it is unnecessary to construct a dense graph. This is mainly because the cost of an initial path is not that important for the optimization-based planning. Instead, it is more important to find different homotopic classes of initial paths. Moreover, constructing such a dense graph would require a massive computational cost due to graph manipulation operations such as NN-search. As a result, the sparse representation of the given space can be a reasonable choice for our

Algorithm 9: VOLUMETRIC TREE*

$V \leftarrow \{x_{init}, x_{goal}\}; \mathcal{T} \leftarrow \{x_{init}\}; E \leftarrow \emptyset$
while *Termination condition is not satisfied* **do**
  $x_{sample} \leftarrow Sample()$
  if $IsCollisionFree(x_{sample})$ **then**
    $V_{near} \leftarrow Near(x_{sample}, V)$
    if $\neg IsInside(x_{sample}, V_{near})$ **then**
      Insert x_{sample} to V
      foreach $v_{near} \in V_{near}$ **do**
        Insert (v_{near}, x_{sample}) to E
      $PropagateCFreeSpace(x_{sample}, V_{near})$
      $\sigma_{new} \leftarrow UpdateDSPT(x_{sample}, \mathcal{T})$
  else
    $V_{near} \leftarrow Near(x_{sample}, V)$
    $UpdateCFreeSpace(V_{near}, x_{sample})$
  if $BetterPathFound(\sigma_{new})$ **then**
    $OptimizePath(\mathcal{T}, \mathcal{G})$
return $SolutionPath(\mathcal{T})$

objective.

In particular, we associate each vertex of a graph with a collision-free hyper-sphere volume for locally representing the C-free space, while rejecting samples inside volumes, resulting in a sparse graph. The idea of using the hyper-sphere volume is inspired by the previous literature [72, 31, 77, 74]. Fig. 4.2(b) shows an exemplar solution path over a sparse graph constructed in the proposed manner, where a set of collision-free hyper-spheres are associated with vertices.

We then utilize the following observation. For a collision-free hyper-sphere S, we can assume that any two given configurations located within S can be connected directly without having any collision. More importantly, it implies that for a given discrete path $\sigma = \{v_0, ..., v_{n-1}\} \in \Sigma$, associated with collision-free hyper-spheres S_{v_i}, centered at v_i, we can then expect any path through the same sequence of vertices located in the hyper-spheres, $\sigma' = \{v'_0 \in S_{v_0}, ..., v'_{n-1} \in S_{v_{n-1}}\}$ is homotopic to σ. We can hence represent plenty of paths homotopic to each other into a single path over the adaptive sparse graph. Our volumetric tree* utilizes this observation for creating a sparse graph and computes the optimal paths with the graph. Fig. 4.1 shows two different types of graphs, a dense graph computed by the conventional approach and ours in an example scene.

4.3.2 Adaptive Sparse Tree Construction

Volumetric tree* constructs a random geometric graph $G = \{V, E\}$, where a vertex $v \in V$ also encodes a collision-free configuration; depending on the context, we can just use v to denote its configuration, and at that case, $v \in \mathcal{X}_{free}$. An edge $e = (v \in V, w \in V) \in E$ represents a continuous motion connecting two configurations. We also define $S_v (\forall v \in V)$ to represent a hyper-sphere centered at a configuration v, associated with a radius of r_v, a distance to the closest empirical collision $o_v \in \mathcal{X}_{obs}$. They are additionally stored in each v.

Our hyper-sphere based representation is designed for approximately encoding $\mathcal{X}_{free}$, and its construction

is inspired by the approximate C-free representation proposed in [74]. The main idea of C-free approximation is to associate each vertex with the closest empirical collision found during the execution, resulting in a set of approximate collision-free hyper-spheres, reducing the approximation error over iterations probabilistically.

For $Near(\cdot)$, we use a distance function specialized for considering radii of vertices to measure a distance between two hyper-spheres associated with those two vertices:

$$dist_{NN}(v \in V, w \in V) = \|v - w\| - r_v - r_w, \tag{4.3}$$

where $\|\cdot\|$ is the Euclidean norm of a vector. The reason why we use a specialized $dist_{NN}(\cdot)$ is to enable a vertex with a large radius to be better connected to other samples for $Near(\cdot)$, either r-NN or k-NN, during the graph expansion. Note that the cost of an edge $e = (v, w)$ is still defined as conventional, i.e., $c(e) = \|v - w\|$.

We can recognize that $dist_{NN}(\cdot)$ violates non-negativity and triangle inequality, which should be held for the metric space. For this reason, the conventional k-d tree or G-NAT [78] can show sub-optimal performance for near neighbor search in volumetric tree*. We hence use NMSlib [79], a proximity-graph based approximate near neighbor search library for generic non-metric spaces for efficient performance.

Fig. 4.1 shows an example of the constructed graph. We can observe that the volumetric tree* covers $\mathcal{X}_{free}$ adaptively with a fewer number of vertices, while the conventional planner maintains a set of tremendous vertices in the same time budget. We further discuss the comparison of the cardinality of graph components in Sec. 4.4 with Table 4.4.

We also maintain a subset of G, $\mathcal{T} = \{V^{\mathcal{T}}, E^{\mathcal{T}} \subset E\}$ for the shortest path computation, where $e \in E^{\mathcal{T}}$ is a set of edges on the shortest paths to all $v \in V^{\mathcal{T}}$ from x_{init} that is constructed by DSPT (Dynmic Shortest Path Tree) [58]. The search tree $\mathcal{T}$ consisting of volumetric vertices $V^{\mathcal{T}}$ in the end contains the solution path we are seeking, and is also used for the homotopy exploration with the dropout technique explained in Sec. 4.3.4.

Overall process. The overall process is depicted in Alg. 9. The proposed volumetric tree* is based on Dancing PRM* [74] in terms of the graph construction and configuration-free space approximation. To be specific, volumetric tree* inherits the lazy collision checking with DSPT and witness propagation with radius compensation from Dancing PRM* for the precomputation-free approximation. Accordingly, we allow G to have edges in collision to reduce unnecessary edge collision checkings and check lazily if they are necessary.

During the iteration in Alg. 9, a collision-free x_{sample} is checked for a inclusion test with its near neighbor set V_{near}, not to fall inside of the hyper-sphere volumes associated with V (Alg. 9, Line: 6). On the other hand, for a $x_{sample} \in \mathcal{X}_{obs}$, we exploit the sample to improve the configuration-free approximation (Alg. 9, Line: 13-14). In $UpdatecFreeSpace$, we update $r_{v_{near}}$ if $\left\|v_{near} - x_{sample}\right\|$ is smaller than $r_{v_{near}}$ to trim the volumes by updating $o_{x_{sample}}$. Likewise, $PropagateCFreeSpace(\cdot)$ initializes $r_{x_{sample}}$ to be $\arg\min_{v \in V_{near}} (\left\|o_v - x_{sample}\right\|)$ and then updates r_v to be $\min(\left\|o_{x_{sample}} - v_{near}\right\|, r_{v_{near}})$ for all $v \in V_{near}$ to propagate the empirical collision information locally. For the details, refer to the original work [74].

$UpdateShortestPath(\cdot)$ updates $\mathcal{T}$ to maintain shortest paths for all possible destinations, i.e., $(v \in V^{\mathcal{T}}, x_{goal})$ dynamically. Finally, *BetterPathFound* (Alg. 9, Line: 15) checks whether the cost of the best-so-far path $c(\sigma_{new})$ is updated in $UpdateDSPT(\cdot)$ at this iteration. If so, we lazily check the feasibility of σ_{new} and then $OptimizePath(\cdot)$ (Alg 9, Line: 16) refines the σ_{new} towards the local optimal path, which is discussed in the subsequent subsection.

4.3.3 Path Optimization

Our path optimization is based on CHOMP (Covariant Hamiltonian Optimization for Motion Planning) [40], which uses gradient descent techniques to optimize the motion iteratively.

The original objective function related to the obstacle cost, f_{obs}, contains so-called obstacle potential terms, which can be expressed as a vector to the closest obstacle or the obstacle proximity, usually obtained by the Euclidean distance transformation in the workspace. It however requires additional precomputation and model simplification such as swept-sphere technique and kinematic Jacobian [40]. For this reason, our objective function is designed to be free from such dependencies to achieve a higher applicability and seamless integration with existing sampling-based planners.

At a high level, our objective function used for volumetric tree* is identical to the original form [40]:

$$\mathcal{U}(\xi) = f_{prior}(\xi) + f_{obs}(\xi), \tag{4.4}$$

where $\xi : [1, n \in \mathbb{N}] \to \mathcal{X}$ is a discrete path consisting of n equidistant intermediate configurations along σ_{new}, i.e., $\xi \in \mathbb{R}^{n \times d}$, and f_{prior} is a sum of squared derivatives, i.e., $\frac{1}{2}\xi^T A\xi + \xi^T b$. $A \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^{n \times d}$ are available in [40].

The obstacle cost f_{obs} for our work is formulated as follows:

$$f_{obs}(\xi) = \sum_{i=1}^{n} \omega(\|\xi^*(i) - \xi(i)\|) \left\| \frac{d}{dt}\xi(i) \right\|, \tag{4.5}$$

where $\omega(\cdot)$ is a user-defined weighting function; we simply use the identify function $\omega(x) = x$ and $\xi^*(t)$ is the last configuration of $\xi(i)$ without collision. Unlike the problem CHOMP and its variants aim to solve, we can assume that the given initial path σ_{new} is collision-free, and thus that $\xi^*(i)$ exists by initializing ξ^* as σ_{new} during the iteration; therefore, $\xi^*(i)$ can be considered as the closest empirical collision-free state of $\xi(i)$.

The update rule follows the iterative quasi-Newton approach like CHOMP, which can be written as:

$$\xi_{i+1} = \xi_i - \nabla\mathcal{U}(\xi_i), \tag{4.6}$$

where $\nabla\mathcal{U}(\cdot) \in \mathbb{R}^{n \times d}$ is:

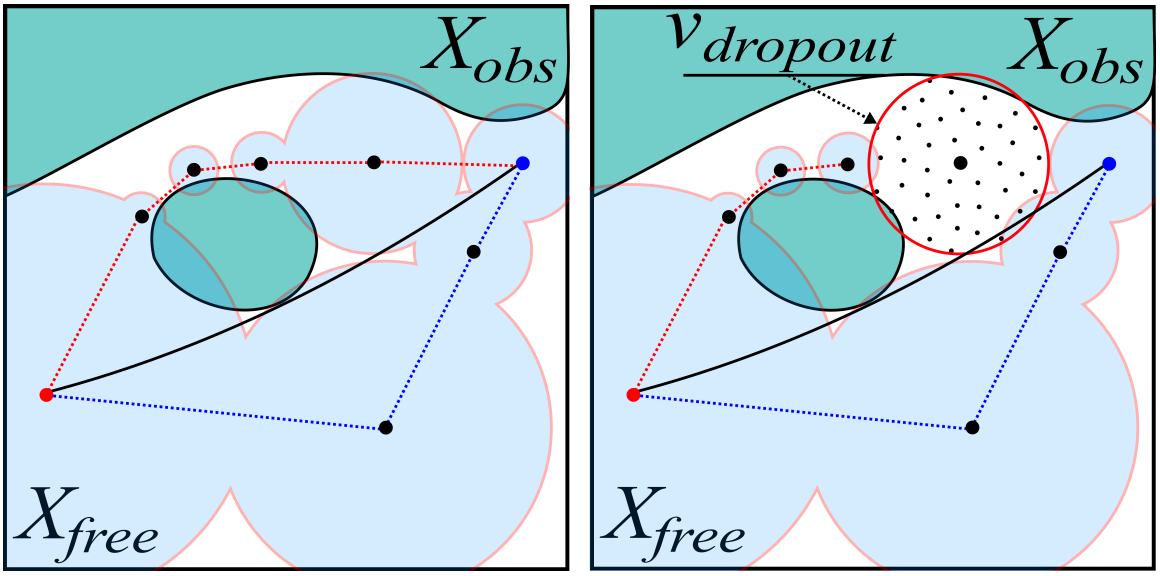
$$\nabla\mathcal{U}(\xi) = \eta A^{-1}(\lambda(A\xi + B) + \nabla f_{obs}(\xi)). \tag{4.7}$$

In the above equation, η is a user-defined convergence rate, set to be 1, $(A\xi + B)$ is ∇f_{prior} in the expanded form, and λ is a trade-off parameter for the smoothness against the obstacle avoidance, set to be $\frac{n}{2}$, where $n = 50$ is the number of intermediate nodes. The number of iterative optimization of Eq. 4.6 is fixed to 50 in our implementation. Lastly, ∇f_{obs} is formulated as:

$$\begin{aligned} \nabla f_{obs}(\xi(i)) = \left\|\xi(i)'\right\| (I - \widehat{\xi(i)}'\widehat{\xi(i)}'^T)(\xi(\widehat{i)^* - \xi(i)}) \\ - \|\xi(i)^* - \xi(i)\| \kappa, \end{aligned} \tag{4.8}$$

where $\xi(i)'$ is the derivative of $\xi(i)$, $\xi\hat{(i)}$ stands for the normalizing function, i.e., $\hat{x} = \frac{\xi(i)}{\|\xi(i)\|}$, and κ is the curvature of ξ at $\xi(i)$. We also replace the obstacle potential terms used for CHOMP with a function of $\xi(i)^* - \xi(i)$, accordingly to Eq. 4.5. The underlying meaning of Eq. 4.8 is that when $\xi(i) \in \mathcal{X}_{obs}$, we push $\xi(i)$ with collision toward its last empirical C-free state $\xi(i)^*$ as a roll-back. In CHOMP, this type of approach is not applicable since a given initial path is assumed in collision, which demands a local workspace obstacle analysis in advance, e.g., the signed distance field, to compute the obstacle potentials efficiently. Our approach, therefore, gets rid of such dependencies and makes the optimization process even intuitive, while combining sampling-based and optimization-based planning seamlessly. Note that $\xi(i) \in \mathcal{X}_{free}$ makes $\nabla f_{obs} = 0$.

For feasibility of the updated path, it is necessary to check ξ for collision and update ξ^* prior to performing the gradient update in Eq. 4.6 at each iteration. For each $\xi(i) \in \mathcal{X}_{obs}$ found during the optimization process is also



(a) (b)

Figure 4.3: Naïve shortest path computation can miss to explore a solution path even homotopic to the optimal solution path (the blue dotted one), due to a sparse graph structure. We address this issue by using the dropout of vertices in solution paths observed during the execution. The dotted-filled ball in the right figure stands for an excluded vertex, $v_{dropout}$. The search tree constructed without $V_{dropout}$ allows our approach to find other solution paths (the blue dotted one) that can be homotopic to the optimal solution, the solid black line.

used to improve our configuration-free space approximation (Sec. 4.3.2). For a set of vertices along the path, i.e., $V_\sigma = \{v | v \in \sigma\}$, we perform $UpdateCFreeSpace(V_\sigma, \xi(i))$ (Alg. 9, Line: 14), which assures that each $o_{v \in \sigma}$ is set to the closest empirical collision found locally.

4.3.4 Shortest Path Computation with Dropout

As the number of iteration increases, volumetric tree* attempts to optimize multiple best-so-far paths as this method is also based on the sampling-based approach. Nonetheless, we found a technical challenge that arises due to the sparse graph structure with hyper-sphere volumes.

The left figure in Fig. 4.3 shows two different paths; one containing a vertex associated with a large volume, σ_{blue} (dotted blue), and the other, σ_{red} (dotted red), with a relatively lower cost, i.e., $c(\sigma_{red}) < c(\sigma_{blue})$. If the σ_{red} is found prior to the σ_{blue}, $UpdateDSPT(\cdot)$ (Alg. 9, Line: 14) can fail to return σ_{blue}, which is homotopic to the optimal solution path (the solid black line).

To ensure that volumetric tree* finds all possible homotopy classes of solution paths given the aforementioned challenge, we propose to use a path optimization with *dropout*, which is a randomized vertex exclusion procedure. The core concept is motivated by the Yen's algorithm [80] designed for finding the loop-less k-th shortest path as a variation of Dijkstra's algorithm.

Our dropout technique is summarized as follows:

1. Whenever a solution path σ_{new} is to be optimized in $OptimizePath(\cdot)$ (Alg. 9, Line: 16), insert σ_{new} to Σ_{new}, a set of all solution paths found so far, and insert all vertices in σ_{new} to $V_{\Sigma_{new}}$, a set of all vertices in Σ_{new}.

2. In $UpdateDSPT(\cdot)$ (Alg. 9, Line: 11), compute a set of excluded vertices, $V_{dropout} \subset V$, at a probability (Eq. 4.9) from V prior to updating $\mathcal{T}$.

3. Update $\mathcal{T}$ with $V \setminus V_{dropout}$ to compute a new shortest path, σ'_{new} from x_{init} to x_{goal}.

4. With dropout, $BetterPathFound(\cdot)$ (Alg. 9, Line: 15) behaves differently; it returns $true$ if σ'_{new} has not been observed previously, i.e., $\sigma'_{new} \notin \Sigma_{new}$.

5. Empty $V_{dropout}$ and repeat the iteration.

Note that our approach is to find a solution paths in unrevealed homotopy classes, while the original Yen's algorithm is for finding the next, i.e., $k+1$-th shortest path by excluding an edge in k-th shortest path.

Fig. 4.3(b) shows an example of the shortest path computation with dropout. Suppose that a vertex associated with the dotted-filled ball is excluded out by dropout, which allows the planner to find σ_{new} as the blue dotted path. Consequently, the dropout enables our approach to find a solution path homotopic to the optimum even with the coarse-grained search graph G constructed by volumetric tree*.

To realize the dropout approach, we record vertices that have been involved in any σ_{new} previously found, and then randomly exclude recorded ones, followed by updating our search tree $\mathcal{T}$ with remaining vertices, The dropout probability for a vertex v can be defined as follows:

$$\mathbb{P}[v \in V_{dropout}] = \begin{cases} c_{dropout} \frac{1}{|V_{\Sigma_{new}}|} & \text{if } v \in V_{\Sigma_{new}}, \\ 0 & \text{otherwise}, \end{cases} \tag{4.9}$$

where $c_{dropout}$ is a user-tuned dropping-out parameter and set to be 1.

As an alternative to our dropout approach, one can consider prioritizing a candidate path σ over G. However, this alternative is difficult to be realized, since for an arbitrary vertex $v \in V$, both cost-to-come from x_{init} and cost-to-go to x_{goal} are unknown in advance. Moreover, using an admissible estimator, e.g., the Euclidean distance in the simplest form, can require an excessive amount of computations due to its weak bound.

4.4 Experiment

We compare the performance of volumetric tree* against the other almost-sure asymptotic optimal sampling-based planners: RRT* [70], Lazy PRM* [56], BIT* [54], and Dancing PRM* [74]. Volumetric tree* is implemented using OMPL (Open Motion Planning Library) [63] including CHOMP-based path optimizer and DSPT (Dynamic Shortest Path Tree). Volumetric tree* uses NMSlib [79] for nearest neighbor search (Sec. 4.3.2). Other planners use G-NAT [78], which is available in OMPL.

We test our method with a 2-DoF rigid body planning (Fig. 4.4) and 6-DoF manipulation problem (Fig. 4.5) using V-REP simulator [64]. We also perform an evaluation in two synthetic environments ($\mathbb{R}^2$ and $\mathbb{R}^8$) shown in Fig. 4.7 and a real-robot experiment using Hubo 4.6(b) to illustrate different behaviors clearly. Our results are averaged over 30 trials and the time budgets are set to 5, 30, 10, 1 and 10 seconds for 2-DoF rigid body, 6-DoF manipulation problem, 7-DoF Hubo, $\mathbb{R}^2$, and $\mathbb{R}^8$, respectively. The detail of the problem settings can be seen in the attached video.

Fig. 4.4 and 4.5 show 2-DoF rigid body and 6-DoF manipulation planning problems, respectively. In these scenes, planners spend much time, 20% to 80% of the total computation, on collision checking. For our method, collision checking takes about 30% to 50% of the total computation time. For the 2-DoF rigid body problem, the computation time on optimization, $T(OPT)$, becomes the major computation bottleneck in volumetric tree* due to a number of explicit collision checkings during the optimizations. On the other hand, for the 6-DoF manipulation problem, collision checking, $T(CC)$, becomes the main computation bottleneck for all the planners, which is caused by the higher complexity of workspace, i.e., the number of triangles.

Volumetric tree*, nevertheless, outperforms the other planners by reducing the number of vertices ($|V|$) by an order of magnitude. It can be interpreted as that our adaptive sparse graph efficiently captures the homotopy of solution paths, while achieving a better quality of solution paths by the integration with optimization-based planning.

In 6-DoF manipulation planning (Fig. 4.5), the computational portion of $T(NN)$ becomes comparable to $T(CC)$ in both Lazy PRM* and Dancing PRM*, which also use lazy collision checking. On the other hand, volumetric tree* achieves a noticeable improvement with the fewer number of vertices and the reduced overhead of NN, despite its higher $T(OPT)$. It is because that volumetric tree* efficiently exploits the solution path using optimization-based planning rather than constructing a dense graph to explore the configuration space. Its benefit is expected to go higher as we have a higher dimensional problem since it is required to have denser graphs in that higher space for other methods.

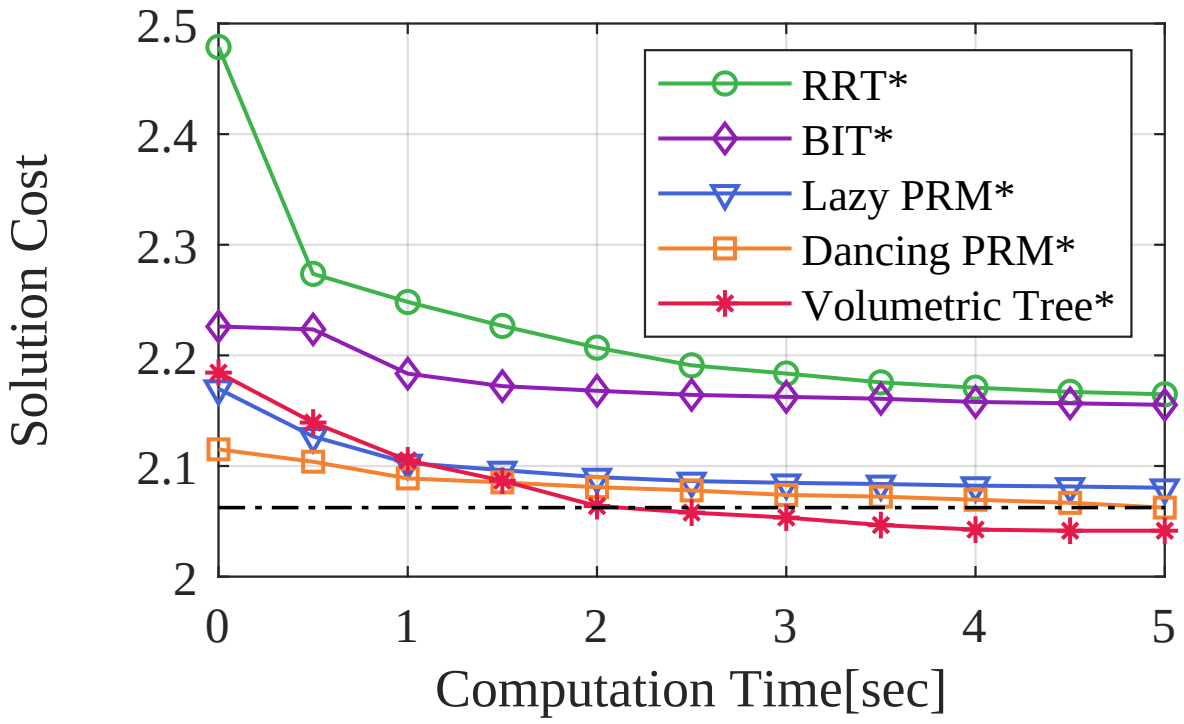
Discussion. We also provide results with synthetic scenes for in-depth analysis. The synthetic benchmarks are designed to have the narrow passage with multiple homotopy classes of the solution path. It can also be considered *NN-sensitve* [65]; the computational cost of nearest neighbor search is not negligible, since that of collision checking is relatively lighter due to the simple geometry. In both $\mathbb{R}^2$ and $\mathbb{R}^8$, x_{init} and x_{goal} are set to $[-1,...-1]^d$ and $[1,...,1]^d$, respectively, in a hyper-cube of width 2, and the narrow gaps are equidistantly located in the middle wall; the gap has a height of $\frac{1}{6}$ and i-th obstacle is defined by two diagonal points, $[-0.15, i(gap + \frac{gap}{10}), -1, \ldots, -1]$ and $[0.15, i \times (gap + \frac{gap}{10}, 1, \ldots, 1)] \in \mathbb{R}^d$.

Fig. 4.8 shows the solution cost as a function of computation time measured in synthetic benchmarks and the corresponding statistics are organized in Table 4.4. In $\mathbb{R}^2$, we can observe some of the other planners outperform volumetric tree*. While volumetric tree* identifies the homotopy class of the optimal solution path and optimizes solution paths toward a local optimum within the finite number of optimization iterations, it may result in near-optimal paths depending on the optimizer parameters, yet the performance gap is $< 1\%$ in terms of the final solution cost. This issue can show a slowdown for our method, especially for simple, lower dimensional problems. Adopting advanced optimization techniques or automatically tuning the parameters depending on the problem are left for future work.

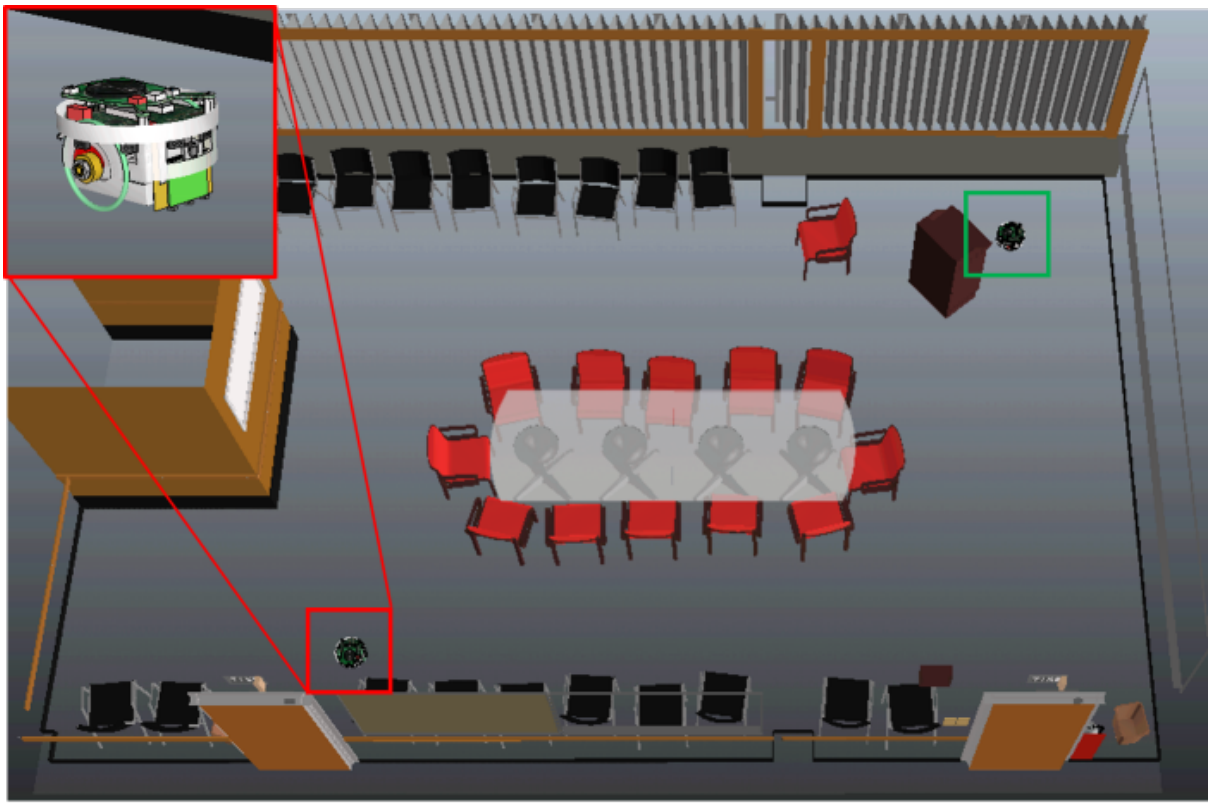
For the case of $\mathbb{R}^8$, volumetric tree* shows exceptional performance improvement. As we can observe in Table 4.4, volumetric tree* checks more vertices ($|VC|$), which can provide a better understanding of the given space, while keeping a fewer number of vertices thanks to our adaptive sparse graph construction with C-free approximation. Furthermore, the hybridization with optimization-based planning allows volumetric tree* not to rely only on sampling for the convergence of solution paths, resulting in a better performance, especially in a higher dimensional problem.

Table 4.1: Statistics of experiment results. $|V|$ and $|E|$ are the cardinality of the vertex set V and edge set E, respectively. $|VC|$ and $|EC|$ stand for the number of vertex and edge collision checking, and $T(\cdot)$ is a computation time ratio of each component: CC = Collision Checking, NN = Nearest Neighbor search, SP = Shortest Path computation in DSPT and OPT = OPTimization. Note that the collision checking time during the optimization is also included in $T(OPT)$ and the number of NN-query is identical to $|VC|$ in volumetric tree*. In 6D manipulation and 7D Hubo arm planning problem, the result of RRT* is not reported due to a high ratio ($>90\%$) of unsuccessful attempts.

Planner	Cost	$\|V\|$	$\|E\|$	$\|VC\|$	$\|EC\|$	$T(CC)$	$T(NN)$	$T(SP)$	$T(OPT)$
2D Rigid body (5sec)									
RRT*	2.164	5363	185276	7239	22163	97%	<1%	N/A	N/A
BIT*	2.156	10102	34220	12604	4506	47%	52%	N/A	N/A
Lazy PRM*	2.081	16274	621267	20297	3825	24%	8%	55%	N/A
Dancing PRM*	2.062	14834	560427	18553	3562	20%	7%	46%	11%
Volumetric Tree*	**2.040**	540	11808	13688	1195	29%	2%	<1%	67%
6D Manipulation (30sec)									
RRT*	-	1316	29051	2543	5314	99%	<1%	N/A	N/A
BIT*	16.799	33588	6336	53064	1179	83%	16%	N/A	N/A
Lazy PRM*	16.315	24259	771850	38309	265	50%	48%	<1%	N/A
Dancing PRM*	16.058	22401	706865	35400	335	47%	39%	<1%	1%
Volumetric Tree*	**13.646**	6242	169534	40788	306	48%	27%	<1%	22%
2D Hyper-cube (1sec)									
RRT*	2.914	44493	1958254	51846	144975	13%	72%	N/A	N/A
BIT*	2.922	4102	23656	4926	7531	2%	82%	N/A	<1%
Lazy PRM*	2.916	16550	634554	19200	1885	1%	27%	56%	N/A
Dancing PRM*	**2.913**	11057	404458	12810	1618	2%	15%	50%	11%
Volumetric Tree*	2.917	353	6862	62406	935	3%	86%	<1%	3%
8D Hyper-cube (10sec)									
RRT*	7.525	29536	938437	35548	132745	4%	95%	N/A	N/A
BIT*	7.750	10791	64296	12502	16637	<1%	97%	N/A	N/A
Lazy PRM*	7.326	25227	779203	29262	1404	<1%	97%	<1%	N/A
Dancing PRM*	7.156	24737	762916	28670	801	<1%	86%	<1%	3%
Volumetric Tree*	**6.243**	5050	128114	61409	1547	4%	90%	<1%	<1%
7D Hubo arm (10sec)									
RRT*	-	548	10251	929	3391	99%	<1%	N/A	N/A
BIT*	3.8423	4727	2197	6516	2216	81%	19%	N/A	N/A
Lazy PRM*	4.4452	8140	224252	12258	179	88%	11%	<1%	N/A
Dancing PRM*	3.9814	8017	220494	12070	187	88%	11%	<1%	<1%
Volumetric Tree*	**2.9767**	1250	26501	9934	197	71%	2%	<1%	27%

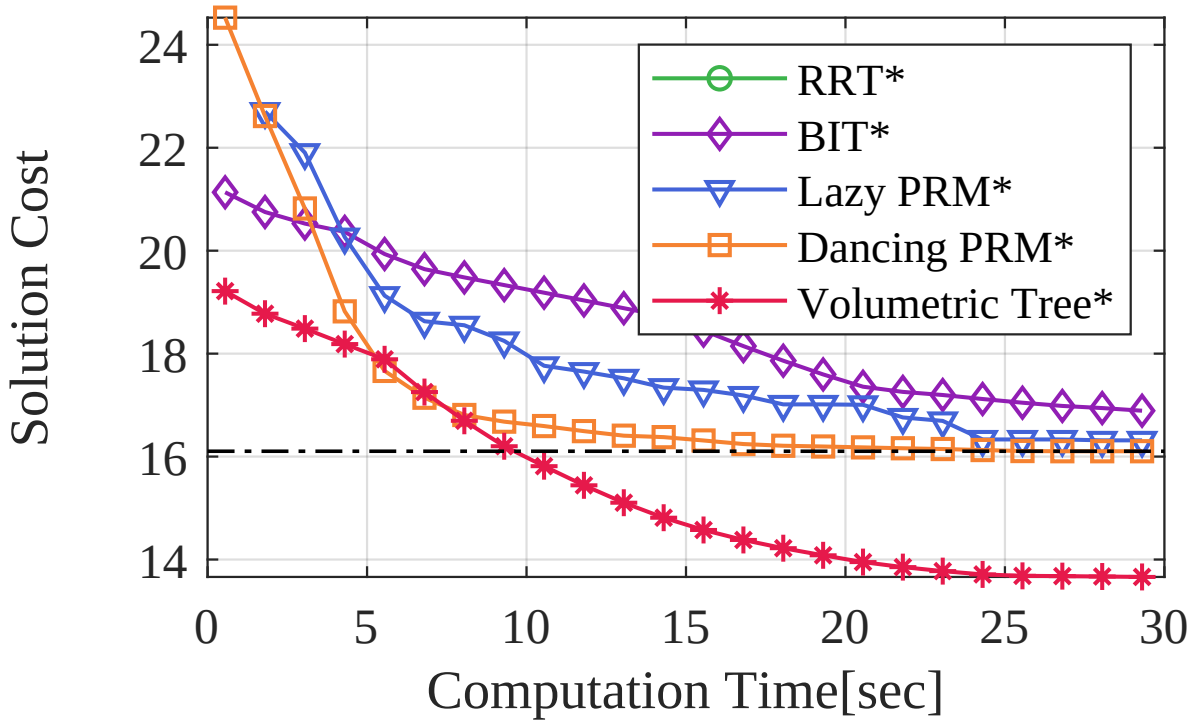


(a)

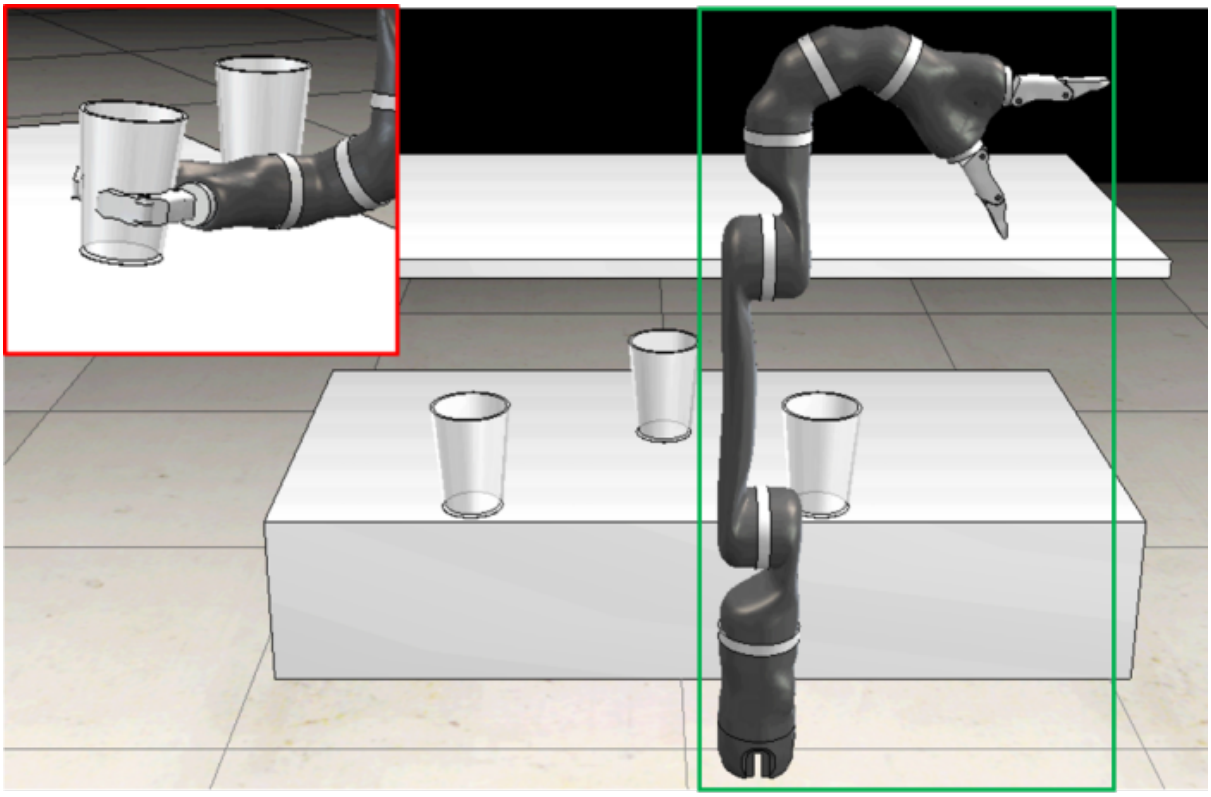


(b)

Figure 4.4: A 2-DoF mobile robot planning problem in a conference room with narrow passages under the chairs, resulting in difficult-to-sample homotopies and surrounding wide-open areas. x_{init} and x_{goal} are depicted in the green and red boxes, respectively.

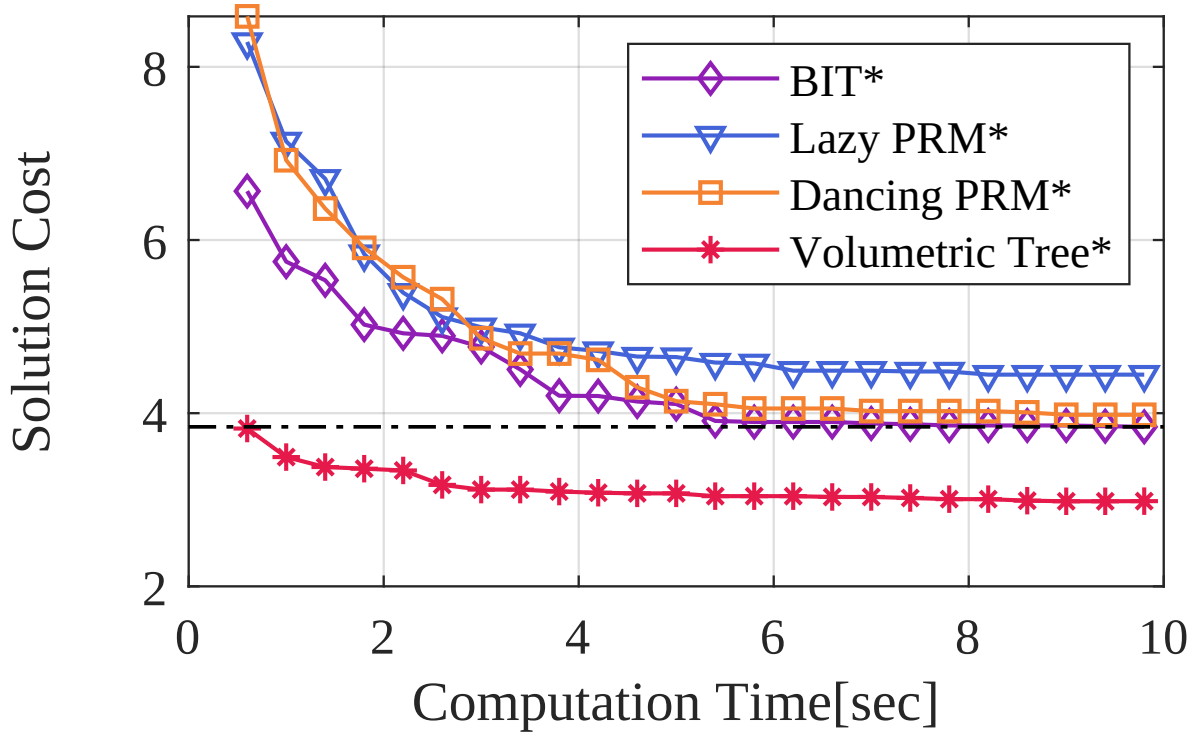


(a)

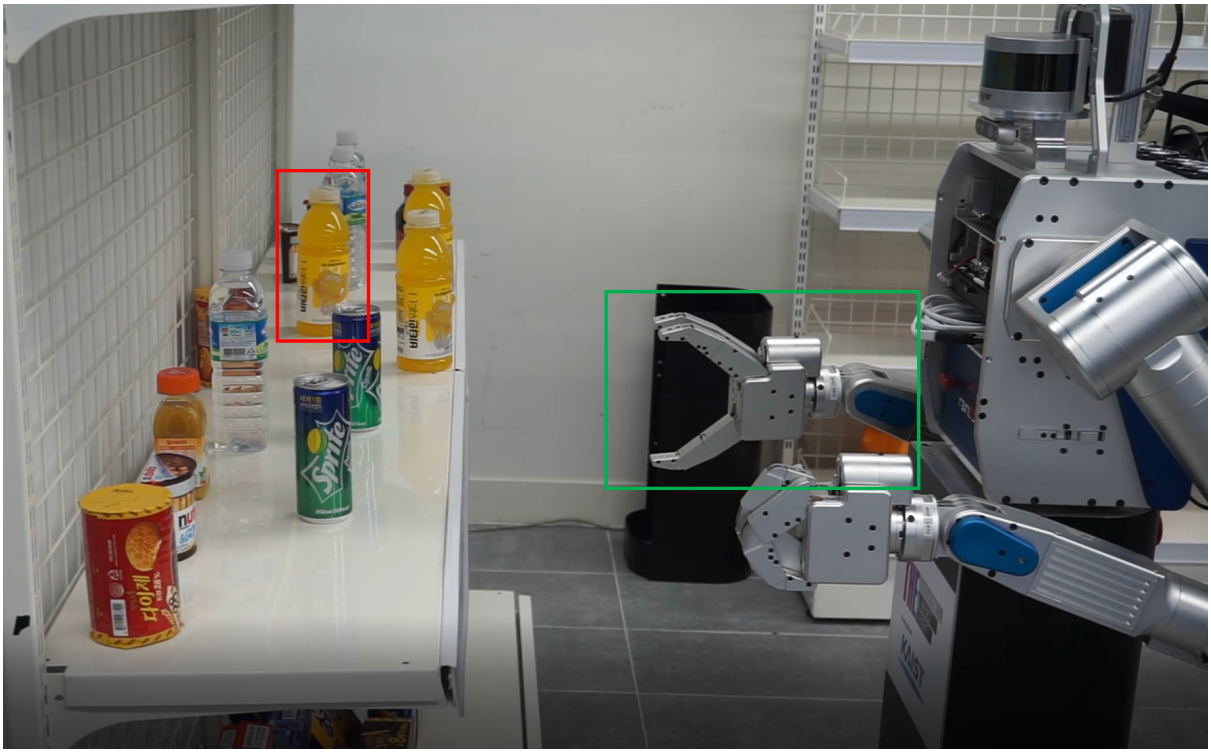


(b)

Figure 4.5: A 6-DoF manipulation planning problem. x_{goal} (in the red box) is a pose of grasping the cup in the middle of the table, avoiding the other cups and the ceiling from x_{init} (in the green box).



(a)



(b)

Figure 4.6: A 7-DoF Hubo robot arm planning problem. The goal is to grasp the target bottle (inside the red rectangle) in the middle of other objects on the shelf.

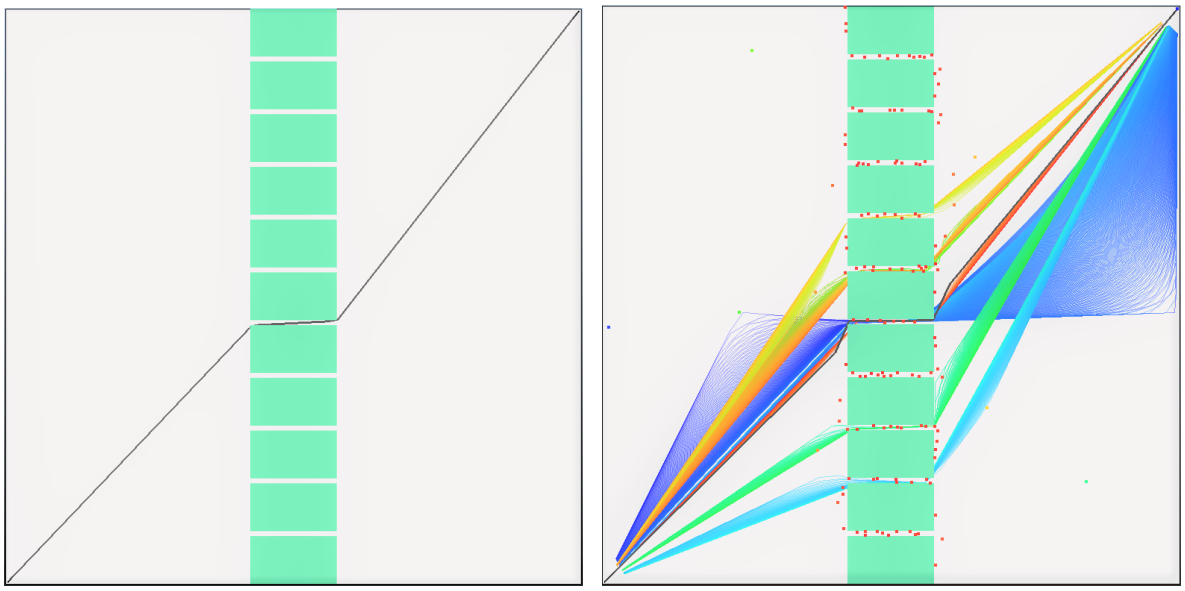
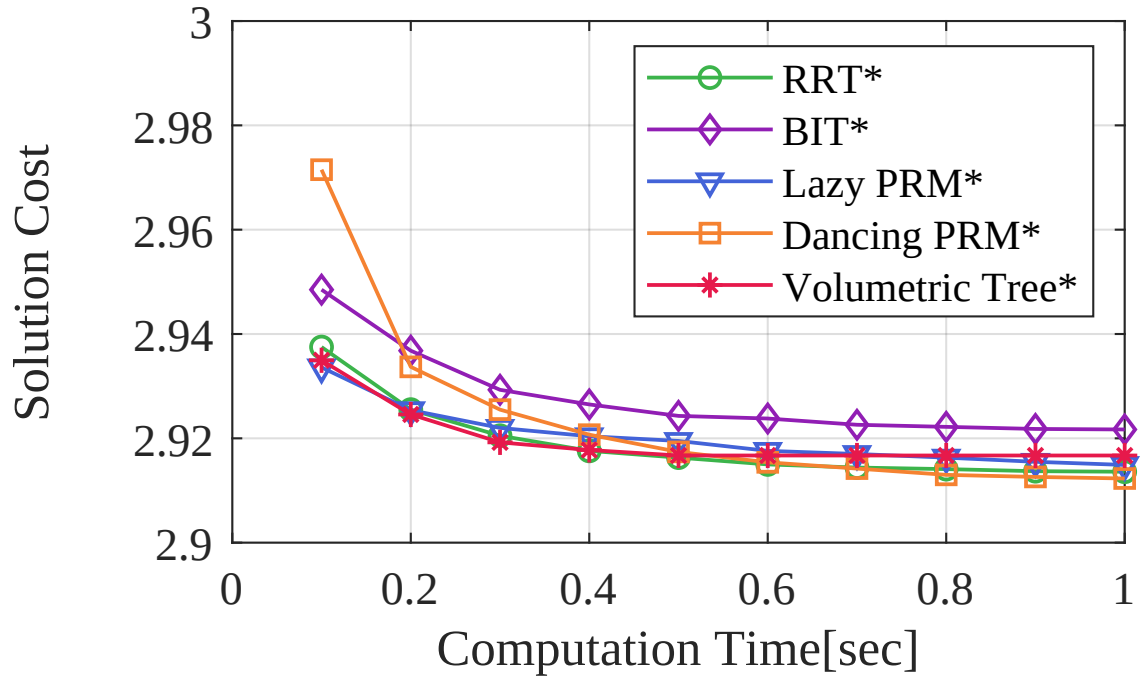
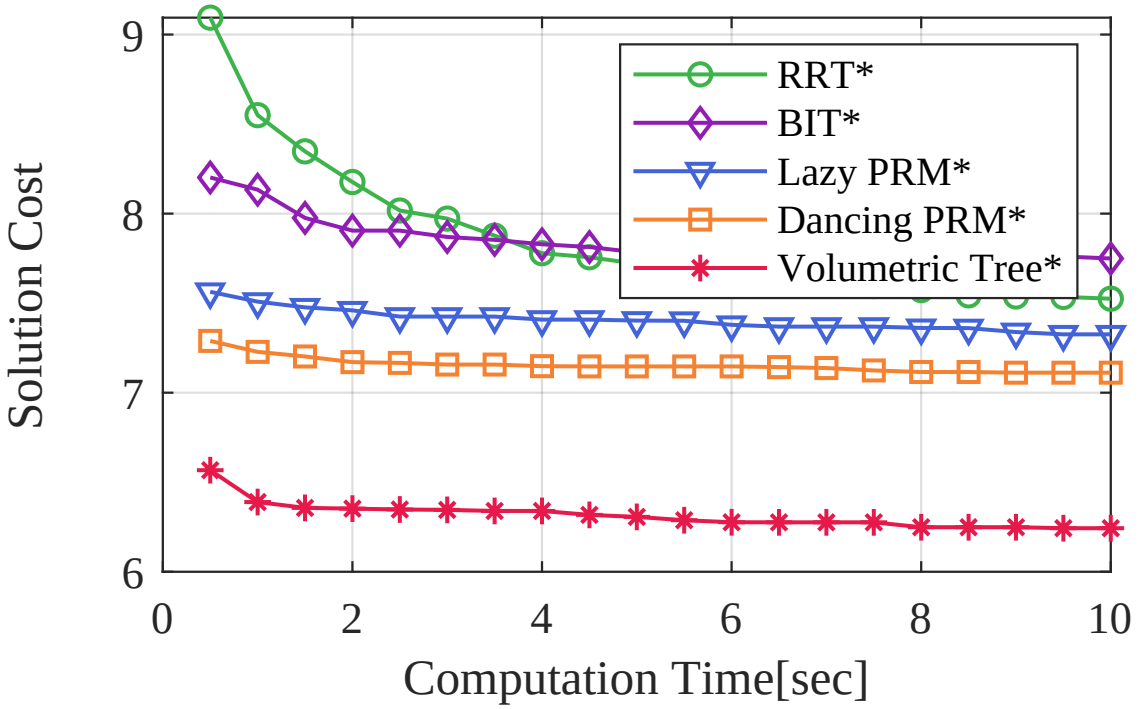


Figure 4.7: 2D synthetic benchmark with 10 narrow gaps located in the middle of the hyper-cube. In the left figure, green squares indicate the obstacles and the black lines are the optimal solution path. The right figure shows intermediate configurations during (colored differently) the optimization iterations. The dots indicate vertices of our sparse graph, colored according to their radii (red for smaller values).



(a) $\mathbb{R}^2$



(b) $\mathbb{R}^8$

Figure 4.8: Performance comparison over computation time for different algorithms in synthetic benchmarks.

Chapter 5. Summary and Conclusion

In this dissertation, we have proposed three different hybridization of motion planning algorithms and three different spherical representations.

In chapter 2, we have proposed Cloud RRT* which utilizes sampling cloud, a set of sampling spheres guided by Generalized Voronoi Graph. Our update method for the sampling cloud is based on newly identified configurations from the solution history and we generate additional spheres to locally exploit such regions. Furthermore, our update method maintains the global sampling distribution pattern for sampling less explored homotopy classes, by locally modifying sampling patterns related only to milestones. We also have proposed a simple pruning method for spheres to cull out unpromising spheres.

In chapter 3, we have presented the local trajectory optimization with the configuration free space approximation algorithm for optimal motion planing algorithms. We have applied the proposed idea to the state-of-the-art motion planning algorithms with lazy collision checking, named as Dancing PRM* and BDT*. They are almost-sure asymptotic optimal hybrid planners, which explore the configuration space with a sampling-based approach while approximating the configuration free space as a set of hyperspheres. The local trajectory optimizer exploits the learned spatial information to optimize local trajectories around narrow passages or around the boundary space to enhance the connections between configurations. Our approximate configuration-free space allows the planner not to depend on the precomputed workspace information or expensive proximity computation for obstacle potentials for a seamless integration. We instead approximate the configuration free space on the fly by empirical collisions found during the execution and decentralize the spatial information over the search graph for efficient access.

In the last chapter 4, we have presented a hybridization of sampling-based and optimization-based planning named volumetric tree*. Our approach constructs a random geometric graph, where each vertex is associated with a hyper-sphere volume with the C-free approximation, while rejecting other samples falling into the space occupied by the existing volumes, resulting in a sparse graph. Volumetric tree* identifies all possible homotopy classes of solution paths with the dropout technique and refines solution paths found during the execution toward the local optimum using optimization-based planning. Our experiment results have shown meaningful performance improvement in most tested environments, showing higher robustness compared to the other tested methods.

Even though we have shown a noticeable performance improvement through synthetic, realistic and real-robot experiments with our approaches, the motion planning algorithm can have the true meaning only when combined with various components such as sensing, controller, and all the other algorithms to make an autonomous system in the real world. Consequently, the ultimate future work would be a hybridization between two different fields, i.e., a collaboration with other researchers in different fields to chase the same objective together.

Bibliography

[1] Jacob T Schwartz and Micha Sharir, "On the piano movers' problem i. the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers", *Communications on pure and applied mathematics*, vol. 36, no. 3, pp. 345–398, 1983.

[2] Lydia E Kavraki and Steven M LaValle, "Motion planning", *Springer Handbook of Robotics*, pp. 109–131, 2008.

[3] Steven M LaValle, *Planning algorithms*, Cambridge university press, 2006.

[4] Steven M LaValle, "Rapidly-exploring random trees a new tool for path planning", Tech. Rep. 98-11, Iowa State University, 1998.

[5] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces", *IEEE Transactions on Robotics and Automation (TRA)*, vol. 12, no. 4, pp. 566–580, 1996.

[6] Sertac Karaman and Emilio Frazzoli, "Sampling-based algorithms for optimal motion planning", *Int'l. Journal of Robotics Research (IJRR)*, vol. 30, no. 7, pp. 846–894, 2011.

[7] Jeong Hwan Jeon, Sertac Karaman, and Emilio Frazzoli, "Anytime computation of time-optimal off-road vehicle maneuvers using the RRT*", in *IEEE Conf. on Decision and Control and Eupopean Control Conference (CDC-ECC)*, 2011, pp. 3276–3282.

[8] Oren Salzman and Dan Halperin, "Asymptotically near-optimal RRT for fast, high-quality motion planning", *IEEE Transactions on Robotics (TRO)*, vol. 32, no. 3, pp. 473–483, 2016.

[9] Alejandro Perez, Robert Platt, George Konidaris, Leslie Kaelbling, and Tomas Lozano-Perez, "LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics", in *IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, 2012, pp. 2537–2542.

[10] Dustin J Webb and Jur van den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics", in *IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, 2013, pp. 5054–5061.

[11] B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions", in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems (IROS)*, 2011, pp. 2640–2645.

[12] Donghyuk Kim, Junghwan Lee, and Sung-eui Yoon, "Cloud RRT*: Sampling cloud based RRT*", in *IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, 2014, pp. 2519–2526.

[13] Jonathan D Gammell, Timothy D Barfoot, and Siddhartha S Srinivasa, "Informed sampling for asymptotically optimal path planning", *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 966–984, 2018.

[14] Jauwairia Nasir, Fahad Islam, Usman Malik, Yasar Ayaz, Osman Hasan, Mushtaq Khan, and Mannan Saeed Muhammad, "RRT*-SMART: A rapid convergence implementation of RRT", *International Journal of Advanced Robotic Systems*, vol. 10, no. 7, pp. 299, 2013.

[15] R. Luna, I. Sucan, M. Moll, and L. Kavraki, "Anytime solution optimization for sampling-based motion planning", in *IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, 2013.

[16] Roland Geraerts, "Planning short paths with clearance using explicit corridors", in *IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, 2010, pp. 1997–2004.

[17] M. Foskey, M. Garber, M. Lin, and D. Manocha, "A voronoi-based hybrid motion planner", in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems (IROS)*, 2001, vol. 1, pp. 55–60.

[18] Jur P van den Berg and Mark H Overmars, "Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners", *Int'l. Journal of Robotics Research (IJRR)*, vol. 24, no. 12, pp. 1055–1071, 2005.

[19] V. Boor, M.H. Overmars, and A.F. van der Stappen, "The gaussian sampling strategy for probabilistic roadmap planners", in *IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, 1999, pp. 1018–1023.

[20] T. Simeon, J. P. Laumond, and C. Nissoux, "Visibility based probabilistic roadmaps for motion planning", *Advanced Robotics Journal*, vol. 14, no. 6, 2000.

[21] D. Hsu, Tingting Jiang, J. Reif, and Zheng Sun, "The bridge test for sampling narrow passages with probabilistic roadmap planners", in *IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, 2003, pp. 4420–4426.

[22] S Dalibard and J.P. Laumond, "Linear dimensionality reduction in random motion planning", *Int'l. Journal of Robotics Research (IJRR)*, 2011.

[23] Junghwan Lee, OSung Kwon, Liangjun Zhang, and Sung-eui Yoon, "SR-RRT: Selective retraction-based RRT planner", in *IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, 2012, pp. 2543–2550.

[24] Ioan A Şucan and Lydia E Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration", in *Algorithmic Foundation of Robotics VIII*, pp. 449–464. Springer, 2009.

[25] O. Brock and L. Kavraki, "Decomposition-based motion planning: A framework for real-time motion planning in high-dimensional configuration spaces", in *IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, 2001, vol. 2, pp. 1469–1474.

[26] Y. Yang and O. Brock, "Efficient motion planning based on disassembly", in *Robotics: Science and Systems (RSS)*, 2005, vol. 1, p. 97.

[27] Yuandong Yang and Oliver Brock, "Adapting the sampling distribution in prm planners based on an approximated medial axis", in *IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, 2004, vol. 5, pp. 4405–4410.

[28] Fahad Islam, Jauwairia Nasir, Usman Malik, Yasar Ayaz, and Osman Hasan, "Rrt*-smart: Rapid convergence implementation of rrt* towards optimal solution", in *Mechatronics and Automation (ICMA), 2012 Int'l. Conf. on*, 2012, pp. 1651–1656.

[29] Zvi Shiller and Sanjeev Sharma, "On-line obstacle avoidance at high speeds", *Int'l. Journal of Robotics Research (IJRR)*, vol. 32, pp. 1030–1047, 2013.

[30] M. Rickert, O. Brock, and A. Knoll, "Balancing exploration and exploitation in motion planning", in *IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, 2008, pp. 2812–2817.

[31] A. Shkolnik and R. Tedrake, "Sample-based planning with volumes in configuration space", *arXiv preprint arXiv:1109.3145*, 2011.

[32] H. Choset and J. Burdick, "Sensor based planning: The hierarchical generalized voronoi graph", *Int'l. Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 1996.

[33] Steven Fortune, "A sweepline algorithm for voronoi diagrams", *Algorithmica*, vol. 2, no. 1-4, pp. 153–174, 1987.

[34] K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha, "Fast computation of generalized voronoi diagrams using graphics hardware", *Proceedings of ACM SIGGRAPH 1999*, pp. 277–286, 1999.

[35] "Boost polygon library (ver. 1.53)", *www.boost.org/libs/polygon*, Aug 2003.

[36] Lester E Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents", *American Journal of mathematics*, vol. 79, 1957.

[37] Sertac Karaman and Emilio Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods", in *IEEE Conf. on Decision and Control*, 2010, pp. 7681–7687.

[38] S. Karaman, M. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT*", in *IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, 2011, pp. 1478–1483.

[39] Joshua Bialkowski, Sertac Karaman, and Emilio Frazzoli, "Massively parallelizing the RRT and the RRT", in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems (IROS)*, 2011, pp. 3513–3518.

[40] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa, "CHOMP: Covariant hamiltonian optimization for motion planning", *Int'l. Journal of Robotics Research (IJRR)*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[41] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal, "STOMP: Stochastic trajectory optimization for motion planning", in *IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, 2011, pp. 4569–4574.

[42] Chonhyon Park, Jia Pan, and Dinesh Manocha, "ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments.", in *Int'l. Conf. on Automated Planning and Scheduling*, 2012.

[43] Brendan Burns and Oliver Brock, "Sampling-based motion planning using predictive models", in *IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, 2005, pp. 3120–3125.

[44] Brendan Burns and Oliver Brock, "Toward optimal configuration space sampling", in *RSS*, Cambridge, USA, June 2005.

[45] Markus Rickert, Arne Sieverling, and Oliver Brock, "Balancing exploration and exploitation in sampling-based motion planning", *IEEE Transactions on Robotics (TRO)*, vol. 30, no. 6, pp. 1305–1317, 2014.

[46] Joshua Bialkowski, Sertac Karaman, Michael Otte, and Emilio Frazzoli, "Efficient collision checking in sampling-based motion planning", in *Int'l. Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2013, pp. 365–380.

[47] Joshua Bialkowski, Michael Otte, and Emilio Frazzoli, "Free-configuration biased sampling for motion planning", in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems (IROS)*, 2013, pp. 1272–1279.

[48] Jia Pan and Dinesh Manocha, "Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing", *Int'l. Journal of Robotics Research (IJRR)*, vol. 35, no. 12, pp. 1477–1496, 2016.

[49] Donghyuk Kim, Youngsun Kwon, and Sung-eui Yoon, "Adaptive lazy collision checking for optimal sampling-based motion planning", in *Int'l Conference on Ubiquitous Robots (UR)*, 2018, pp. 320–327.

[50] Jory Denny and Nancy M. Amato, "Toggle prm: Simultaneous mapping of c-free and c-obstacle - a study in 2d -", in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems (IROS)*, 2011, pp. 2632–2639.

[51] Jory Denny and Nancy M. Amato, "The toggle local planner for sampling-based motion planning.", in *IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, 2012, pp. 1779–1786.

[52] Jory Denny, Kensen Shi, and Nancy M Amato, "Lazy toggle prm: a single-query approach to motion planning", in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 2407–2414.

[53] Sanjiban Choudhury, Jonathan D Gammell, Timothy D Barfoot, Siddhartha S Srinivasa, and Sebastian Scherer, "Regionally accelerated batch informed trees (RABIT*): A framework to integrate local information into optimal path planning", in *IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, 2016, pp. 4207–4214.

[54] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot, "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs", in *IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, 2015, pp. 3067–3074.

[55] Alan Kuntz, Chris Bowen, and Ron Alterovitz, "Fast anytime motion planning in point clouds by interleaving sampling and interior point optimization", *Int'l. Symposium on Robotics Research*, 2017.

[56] Kris Hauser, "Lazy collision checking in asymptotically-optimal motion planning", in *IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, 2015.

[57] Nika Haghtalab, Simon Mackenzie, Ariel D Procaccia, Oren Salzman, and Siddhartha S Srinivasa, "The provable virtue of laziness in motion planning", in *Int'l. Conf. on Automated Planning and Scheduling*, 2018.

[58] Daniele Frigioni, Alberto Marchetti-Spaccamela, and Umberto Nanni, "Fully dynamic algorithms for maintaining shortest paths trees", *Journal of Algorithms*, vol. 34, no. 2, pp. 251–281, 2000.

[59] Sven Koenig, Maxim Likhachev, and David Furcy, "Lifelong planning A*", *Artificial Intelligence*, vol. 155, no. 1-2, pp. 93–146, 2004.

[60] Jerzy W Jaromczyk and Godfried T Toussaint, "Relative neighborhood graphs and their relatives", *Proceedings of the IEEE*, vol. 80, no. 9, pp. 1502–1517, 1992.

[61] Michael Otte and Emilio Frazzoli, "Rrt-x: Real-time motion planning/replanning for environments with unpredictable obstacles", in *Algorithmic Foundations of Robotics XI*, pp. 461–478. Springer, 2015.

[62] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic", in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems (IROS)*, 2014, pp. 2997–3004.

[63] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki, "The Open Motion Planning Library", *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, http://ompl.kavrakilab.org.

[64] M. Freese E. Rohmer, S. P. N. Singh, "V-REP: a versatile and scalable robot simulation framework", in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems (IROS)*, 2013.

[65] Michal Kleinbort, Oren Salzman, and Dan Halperin, "Collision detection or nearest-neighbor search? on the computational bottleneck in sampling-based motion planning", *Int'l. Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2016.

[66] Sertac Karaman and Emilio Frazzoli, "Incremental sampling-based algorithms for optimal motion planning", *arXiv preprint arXiv:1005.0416*, 2010.

[67] Paul Deheuvels, "Strong bounds for multidimensional spacings", *Probability Theory and Related Fields*, vol. 64, no. 4, pp. 411–424, 1983.

[68] Harald Niederreiter, *Random number generation and quasi-Monte Carlo methods*, vol. 63, SIAM, 1992.

[69] Stephen R Lindemann and Steven M LaValle, "Current issues in sampling-based motion planning", in *Robotics Research. The Eleventh International Symposium*. Springer, 2005, pp. 36–54.

[70] J. Kuffner and S.M. LaValle, "RRT-connect: An efficient approach to single-query path planning", in *IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, 2000, pp. 995–1001.

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

[72] Joshua Bialkowski, Michael Otte, Sertac Karaman, and Emilio Frazzoli, "Efficient collision checking in sampling-based motion planning via safety certificates", *Int'l. Journal of Robotics Research (IJRR)*, vol. 35, no. 7, pp. 767–796, 2016.

[73] John Schulman, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization.", in *Robotics: science and systems*, 2013.

[74] Donghyuk Kim, Youngsun Kwon, and Sung-Eui Yoon, "Dancing PRM*: Simultaneous planning of sampling and optimization with configuration free space approximation", in *IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, 2018, pp. 7071–7078.

[75] Valerio Varricchio and Emilio Frazzoli, "Asymptotically optimal pruning for nonholonomic nearest-neighbor search", in *IEEE Conf. on Decision and Control*. IEEE, 2018, pp. 4459–4466.

[76] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions", *The International journal of robotics research*, vol. 34, no. 7, pp. 883–921, 2015.

[77] Sean Quinlan and Oussama Khatib, "Elastic bands: Connecting path planning and control", in *Proceedings IEEE International Conference on Robotics and Automation*. IEEE, 1993, pp. 802–807.

[78] Sergey Brin, "Near neighbor search in large metric spaces", in *International Conference on Very Large Data Bases*, 1995.

[79] Leonid Boytsov and Bilegsaikhan Naidan, "Engineering efficient and effective non-metric space library", in *Int'l Conf. on Similarity Search and Applications*. Springer, 2013, pp. 280–293.

[80] Jin Y Yen, "Finding the k shortest loopless paths in a network", *management Science*, vol. 17, no. 11, pp. 712–716, 1971.

Acknowledgments in Korean

2010년도, 학부를 졸업할 당시 제 이름이 담긴 논문을 하나 갖고 싶다는 마음 만으로 시작한 대학원 생활은 생각보다 긴 여정을 거쳐 오늘에야 마무리 되었습니다. 1년간의 석사 도중 하차, 1년간의 인턴과 스타트업 그리고 현재의 연구실에서 새로운 석사 2년 및 박사 5년 반의 여정을 거쳤습니다. 윤성의 교수님과 이제는 졸업하여 사회로 돌아간 선후배들을 만나 적지않은 시간을 보내며 마침내, 다소 초라하지만 소중한 박사학위 논문을 완성하여 기쁘기도 하고 다소 부족한 자신에게 씁쓸함도 있습니다. 무엇보다도 시작부터 쉽지 않은 입학과 7년 반 간의 부족한 저에게 가르침을 전해주신 윤성의 교수님께 더할 나위 없는 감사의 뜻을 표하고 싶습니다. 로봇팀 1호 박사님인 이정환형에게 받은 가르침에도 심심치 않은 감사의 말을 올립니다.

앞으로 교수님과 함께 연구실을 이끌어 갈 김수민, 권용선, 김태영, 강민철, 안인규, 임우빈, 신희찬, 김재윤, 박훈민, 김재윤, 조윤기군에게는 좋은 나날들이 함께하길 바랍니다. 연구실을 조만간 떠날 송치완, 최홍선군도 새로운 곳에서 좋은 사람들을 만나기 바랍니다. 저의 삶에 무한한 지원을 해주신 부모님과 형에게도 잊지 않고 감사의 뜻을 올립니다. 더불어 10년만의 재회를 감상할 수 있게 한 안유리에게도 고맙다고 말하고 싶습니다.

제가 읽었던 수 많은 논문과 책들의 저자는 물론이고 듣고 보고 느꼈던 모든 이들에 감사합니다. 잠시 몸담았던 스타트업에서 감사의 편지를 쓰는 시간에, 제가 화장실 라디에이터 (겨울에 화장실이 상당히 추웠었음)에게 쓴 감사의 편지를 기억하는 모든 사람들에게 감사합니다. Red가 마지막으로 저에게 해주었던 ”자신의 인생에 중요한 선택의 순간이 있었음을 기억해달라”는 말을 저는 아직도 기억하고 감사하게 생각하고 있습니다. 아주 어린 시절 즐겼던 데모 게임에 대한 흐릿한 저의 설명을 1년간 잊지 않고, 마침내 찾아주신 모 커뮤니티 유저 ’니아옹’님에게도 감사의 뜻을 표합니다. 하물며 어린 시절 아파트 계단에 떨어져 있었던 아이스크림 껍데기에게도 감사의 뜻을 보냅니다. 그 후로 저는 평생 쓰레기를 길에 버리지 않기로 했기 때문입니다. 제 기억이 맞다면 그것은 스크류바 껍데기였을 겁니다.

그리고 마지막으로 이 여정을 즐겁게 마쳐낸 제 자신에게 그 누구보다도 더 감사합니다.

Curriculum Vitae in Korean

이　　름: 김 동 혁

생 년 월 일: 1987년 06월 26일

학　　력

2004. 3. – 2006. 2.　용남고등학교

2006. 3. – 2010. 2.　서강대학교 컴퓨터공학과 (학사)

2012. 3. – 2014. 2.　한국과학기술원 전산학부 (석사)

2014. 2. – 2019. 7.　한국과학기술원 전산학부 (박사)

경　　력

2011. 4. – 2011. 7.　구글 코리아 소프트웨어 엔지니어링 인턴

2011. 7. – 2012. 2.　Adbyme(현 Zoyi Inc.) 소프트웨어 엔지니어

연 구 업 적

1. **Donghyuk Kim** and Sung-eui Yoon, “Simultaneous Planning of Sampling and Optimization: Study on Lazy Evaluation and Configuration Free Space Approximation for Optimal Motion Planning Algorithm”, Autonomous Robots (AURO), (Conditionally accepted)
2. **Donghyuk Kim** and Sung-eui Yoon, “Volumetric Tree*: Adaptive Sparse Graph for Effective Exploration of Homotopy Classes, (Under review)
3. Youngsun Kwon, **Donghyuk Kim** and Sung-eui Yoon, “Super Rays and Culling Region for Real-Time Updates on Grid-based Occupancy Maps”, IEEE Transactions on Robotics (T-RO), 2019
4. Heechan Shin, **Donghyuk Kim** and Sung-eui Yoon, “Kinodynamic Comfort Trajectory Planning for Car-like Robots”, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018
5. **Donghyuk Kim**, Youngsun Kown and Sung-eui Yoon, “Adaptive lazy Collision Checking for Optimal Sampling-based Motion Planning”, International Conference on Ubiquitous Robots (UR), 2018
6. **Donghyuk Kim**, Youngsun Kown and Sung-eui Yoon, “Dancing PRM*: Simultaneous Planning of Sampling and Optimization with Configuration Free Space Approximation”, IEEE International Conference on Robotics and Automation (ICRA), 2018
7. Youngsun Kwon, **Donghyuk Kim** and Sung-eui Yoon, “Super Ray based Updates for Occupancy Maps”, IEEE International Conference on Robotics and Automation (ICRA), 2016
8. **Donghyuk Kim**, Youngsun Kwon and Sung-eui Yoon, “Fully sample-based configuration free space approximation for optimal motion planning”, Robotics: Science and Systems (RSS) Workshop on Recent Advances in Planning and Manipulation for Industrial Robots, 2016

9. **Donghyuk Kim**, Junghwan Lee and Sung-eui Yoon, "Cloud RRT*: Sampling Cloud based RRT*", IEEE International Conference on Robotics and Automation (ICRA), 2014