Auton Robot (2019) manuscript No.
(will be inserted by the editor)

Simultaneous Planning of Sampling and Optimization

Study on Lazy Evaluation and Configuration Free Space Approximation for Optimal Motion Planning Algorithm

Donghyuk Kim[1] · **Sung-Eui Yoon**[1]

Received: date / Accepted: date

Abstract A recent trend in optimal motion planning has broadened the research area toward the hybridization of sampling, optimization, and grid-based approaches.

A synergy from such integrations can be expected to bring the overall performance improvement, but seamless integration and generalization is still an open problem. In this paper, we suggest a hybrid motion planning algorithm utilizing both sampling and optimization techniques, while simultaneously approximating a configuration-free space.

Unlike conventional optimization-based approaches, the proposed algorithm does not depend on a priori information or resolution-complete factors, e.g., a distance field. Ours instead learns spatial information on the fly by exploiting empirical collisions found during the execution, and decentralizes the information over the constructed graph for an efficient reference. With the help of the learned information, we associate the constructed search graph with the approximate configuration-free space so that our optimization-based local planner exploits the local area to identify the connectivity of free space without depending on the precomputed workspace information.

To show the novelty of the proposed algorithm, we apply the proposed idea to asymptotic optimal planners with lazy collision checking; lazy PRM* and Batch Informed Tree*, and evaluate them against other state-of-the-arts in both synthetic and practical benchmarks with varying degrees of freedom. We also discuss the performance analysis, properties of different algorithm frameworks of lazy collision checking and our approximation method.

Keywords Motion Planning · Lazy Collision Checking · Trajectory Optimization · Configuration-Free Space Approximation

Donghyuk Kim
E-mail: donghyuk.kim@kaist.ac.kr

Sung-Eui Yoon
E-mail: sungeui@kaist.edu, corresponding author

[1]Scalable Graphics, Vision and Robotics (SGVR) Lab, School of Computing, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea

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 (LaValle 1998) and PRM (Kavraki et al. 1996), which can be viewed as random geometric graph construction in the configuration-free space.

For the asymptotic optimal motion planning, Karaman and Frazzoli (2011) presented RRT*, PRM*, and RRG, 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 (Bialkowski et al. 2011; Jeon et al. 2011; Karaman et al. 2011).

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 (Kalakrishnan et al. 2011; Park et al. 2012; Zucker et al. 2013). 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 (Bialkowski et al. 2013a,b; Burns and Brock 2005a,b; Pan and Manocha 2016; Rickert et al. 2014; Shkolnik and Tedrake 2011).

Among them, Shkolnik and Tedrake (2011) represented the configuration-free space as a set of hyperspheres using empirical collision information for efficient biased sampling. Bialkowski et al. (2013a) took a similar approach but used explicit proximity computation to compute the boundary of the hyperspheres. In their following work, Bialkowski et al. (2013b) 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 (2011, 2012) suggested a methodology of mapping configuration-obstacle space to guide the sampling within free space. Denny et al. (2013) presented an extensive work in conjunction with lazy graph expansion for efficient graph expansion.

Kim et al. (2018a) suggested using configuration-free space approximation to predict regions that likely have no collisions. This approach is tailored to lazy collision checking to balance the overhead of collision checking and graph restructuring. Pan and Manocha (2016) proposed a probabilistic collision checking with free space approximation. 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. (2016) suggested *RABIT** (Regionally Accelerated Batched Informed Trees) which is the integration of a sampling-based algorithm BIT* (Batched Informed Tree, Gammell et al. 2015) and an optimization-based algorithm, CHOMP (Covariant Hamiltonian Optimization for Motion Planning, Zucker et al. 2013). 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 paper, 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 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. 2). Sec. 3 gives an overview of the proposed algorithm (Sec. 3.1), and how to approximate the configuration-free space with sampling-based motion planning (Sec. 3.2), followed by optimization based local planning with learned information (Sec. 3.3). We then demonstrate its benefits by experiments with various dimensions against other state-of-the-art planners (Sec. 4). Lastly, we discuss the properties of our method and the configuration-free space approximation in detail (Sec. 5).

Algorithm 1: 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)
```

This paper is an extended version of Dancing PRM* (Kim et al. 2018b) to shed light on the applicability and novelty of Dancing PRM*. It shares preliminaries (Sec. 1, 2) and the central theme (Sec. 3) with the original work. Some of experiment results and analysis on the asymptotic complexity are reused combined with our new results. On top of that, we provide additional materials in this paper, as follows:

1. Provide an abstraction of our approach over a generic sampling-based motion planning algorithm (Sec. 3).
2. Present BDT* (Batch Dancing Tree), which is the integration of the local trajectory optimization with configuration-free space approximation of Dancing PRM* and BIT* (Batch Informed Trees Gammell et al. 2015), to show the generality of our idea (Sec. 3.4).
3. Report additional experiment results to show the performance of BDT* with analysis of the different behavior between BDT* and Dancing PRM* in terms of a lazy graph expansion (Sec. 4).
4. Discuss how each component in our configuration-free space approximation, i.e., witness propagation and radius compensation, contributes to the error reduction (Sec. 5.3 and 5.4).

2 Background

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

2.1 Sampling-based Motion Planning

In this paper, 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 $\xi : [0,1] \rightarrow \mathbb{X}$. 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. 1, which is one of the prominent sampling-based planners.

In each iteration of Alg. 1, 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 (Karaman and Frazzoli 2011). k-nn (k-nearest neighbor) query can also be an alternative to r-nn within $Near(\cdot)$; in that case, k is defined by $\lceil \gamma(\mathrm{e} + \frac{\mathrm{e}}{d}) \cdot log(|V|) \rceil$ (Karaman and Frazzoli 2011).

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 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.

2.2 Optimization-based Motion Planning

Our local planner is based on a gradient optimization technique, CHOMP (Covariant Hamiltonian Optimization for Motion Planning, Zucker et al. 2013). We first briefly review the concept of CHOMP and discuss the motivation with our observations.

Table 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.

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

$$\mathscr{U}(\xi) = f_{prior}(\xi) + \lambda \cdot f_{obs}(\xi), \tag{1}$$

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. 2 and 3, respectively:

$$f_{obs}(\xi) = \int_0^1 \int_B c(x(\xi(t),b)) \left\| \frac{d}{dt} x(\xi(t),b) \right\| db\, dt, \tag{2}$$

$$\nabla f_{obs}(\xi) = \int_B J^T \|x'\| \cdot [(I - \hat{x}'\hat{x}'^T)\nabla c - c\kappa]\, db. \tag{3}$$

Roughly speaking, Eq. 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 1.

The computation of the obstacle potential $c(\cdot)$ depends on a body simplification, and workspace information such as the distance field (Zucker et al. 2013), which can be computed using Euclidean Distance Transforms (EDT) in the workspace, for instance.

The robot model B is generally simplified by a swept-sphere volume (Park et al. 2012; Zucker et al. 2013). $x(\cdot)$ 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 $\xi(t)$.

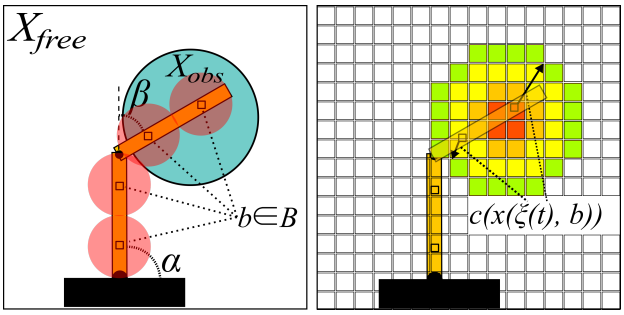


Fig. 1 These figures show a 2D manipulation problem with two joints α and β in a planar space. The left figure shows how the robot is represented for working with workspace obstacle information (left) and the signed distance field of the environment (right). In the left, a robot arm is simplified by 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).

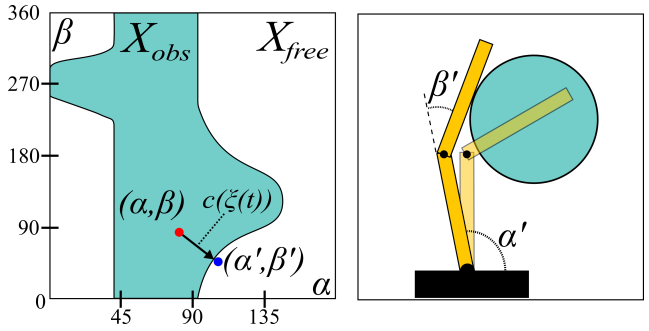


Fig. 2 The left figure shows the configuration space of Fig. 1. If we can represent the configuration space in a usable form, we can directly compute the obstacle potential and its gradient without any other conversions between the workspace and configuration space. The right figure shows the corresponding movement in the workspace between two configurations (α,β) and (α', β').

Eq. 3 shows the gradient of Eq. 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.

Fig. 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. 2 and Eq. 3 with kinematic Jacobian J in order to be used in the configuration space where ξ is defined.

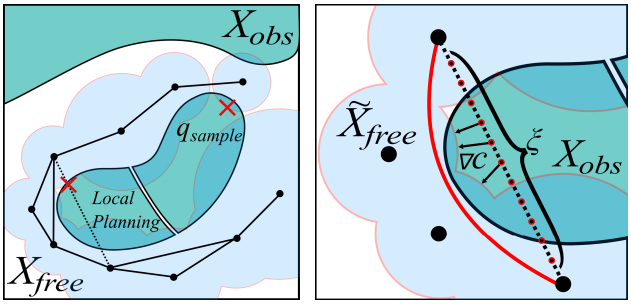
The limitation of the aforementioned process can be summarized as follows. First, CHOMP only guarantees the convergence to a local optimum since the optimization process exploits the local convexity of f_{obs}. Fortunately, the local optimality issue has been studied extensively with a stochastic sampling (Kalakrishnan et al. 2011; Zucker et al. 2013) and a hybrid approach of sampling-based and optimization-based planning (Choudhury et al. 2016; Kuntz et al. 2017). 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. 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 for a high-dimensional space. It will, thus, be the key to our problem for representing and approximating 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 Algorithm

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



(a) (b)

Fig. 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 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.

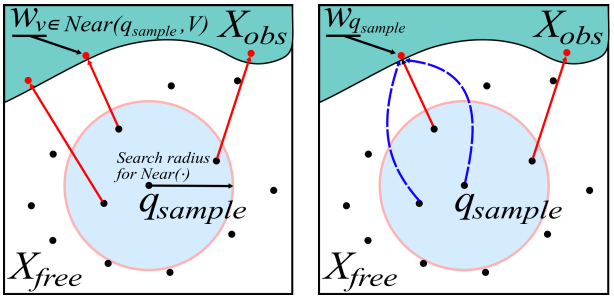


Fig. 4 A sequence of witness propagation for a sample configuration q_{sample} and its near neighbors within the light-blue circle with a radius of $\gamma\left(\frac{\log|V|}{|V|}\right)^{1/d}$. Pairs of configuration and its former witness are connected by red segments, and blue dotted lines indicate 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.

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. 5 shows an abstraction of the proposed algorithm. The main flow with the four solid boxes shows a single iteration of a generic sampling-based motion planning algorithm, which also can correspond to PRM in Alg. 1. 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. 3.2). We also propagate the empirical collision information, and inherit from near neighbors on the implicit proximity graph (Sec. 3.2 and 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 (Hauser 2015, Gammell et al. 2015, Haghtalab et al. 2018).

To show the novelty and applicability, we suggest two different asymptotic optimal planners, Dancing PRM* and BDT* (Batch Dancing Tree) named after lazy PRM* (Hauser 2015) and BIT* (Batch Informed Tree, Gammell et al. 2015), respectively. Lazy PRM* is based on DSPT (Dynamic Shortest Path Tree, Frigioni et al. 2000) for lazy collision checking, while BIT* uses LPA* (Life-long Planning A*, Koenig et al. 2004) based approach. We also discuss how differently both algorithmic frameworks behave in terms of lazy graph expansion in Sec. 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.

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 (Bialkowski et al. 2013a; Kim et al. 2018a; Shkolnik and Tedrake 2011)

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 efficient integration with optimization-based planning explained in a later paragraph.

Fig. 3(a) shows a conceptual image of $\widetilde{\mathbb{X}}_{free}$, which can be formalized as:

$$\widetilde{\mathbb{X}}_{free} = \{x \mid \|x - q_i\| < r_{q_i}\},\ q_i \in V, \tag{4}$$

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 $\|q - w_q\| = r_q$.

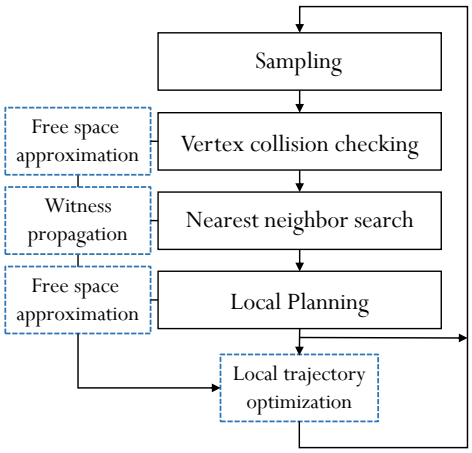


Fig. 5 An abstraction of the proposed algorithm. The blue dotted boxes indicate proposed components in conjunction with a generic sampling-based motion planning algorithm represented by the solid boxes.

A new witness w_q can be generated and replaced by the samples listed in List. 1 during the execution, as long as it results in a smaller radius.

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

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 (LaValle 2006).

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. 2 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. 1, 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. 1. We discuss the details in the subsequent 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(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

Algorithm 2: DANCING PRM*

```
1  V ← {q_init, q_goal}; E ← ∅
2  while Termination condition is not satisfied do
3      q_sample ← Sample()
4      if IsCollisionFree(q_sample) then
5          Insert q_sample to V
6          V_{q_sample} ← ∅
7          Q_near ← Near(q_sample, V)
8          Insert Q_near to V_{q_sample}
9          foreach q_near ∈ Q_near do
10             Insert (q_near, q_sample) to E
11             Insert q_sample to V_{q_near}
12         UpdateShortestPathTree(·)
13         PropagateCFreeSpace(Q_near, q_sample)
14     else
15         q_nearest ← Nearest(q_sample, V)
16         UpdateCFreeSpace(q_sample)
17     CheckSolutionPath(G)
18 return SolutionPath(G)
```

$V_q \subset V$. V_q is updated with its near neighbors (Line: 6, 8, 11 in Alg. 2).

As a result, our search graph G can be considered as a proximity graph (Jaromczyk and Toussaint 1992), and also similar to the neighborhood set of RRT^X (Otte and Frazzoli 2015). 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. 2). 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 (Karaman and Frazzoli 2011). 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.

For this purpose, we define $PropagateCFreeSpace(\cdot)$ (Alg. 3), 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. 4 shows an exemplar sequence of witness propagation for a sample configuration q_{sample} depicted in Alg. 3. This process enhances the association between local near neighbors and the closest witness to reduce the

Algorithm 3: *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**
    $r_{q_{near}} \leftarrow \|w_{q_{sample}} - q_{near}\|$
    $w_{q_{near}} \leftarrow w_{q_{sample}}$

error of our configuration-free space approximation. Likewise, $UpdateCFreeSpace(\cdot)$ (Line: 16 in Alg. 2) 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. 5.3.

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. 4. Compared to lazy PRM* (Hauser 2015), we additionally activate our local trajectory optimizer, $Optimize(\cdot)$ only after a collision checking for an edge $(v \in V, w \in V)$ is failed, it can be thus considered as a backup local planner.

At the beginning of each iteration in $CheckSolutionPath(\cdot)$ (Alg. 4), 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*, Choudhury et al. 2016) which assumes the obstacle potential, f_{obs} and ∇f_{obs}, to be given a priori or analytically computable.

Algorithm 4: *CheckSolutionPath*

Input: G, a search graph

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

When it comes to the optimization process, we replace the obstacle potential computations (Eqs. 2, 3) with more simpler forms (Eqs. 8, 5) by removing the conversion between the workspace and configuration space.

In the objective function of Eq. 1, f_{prior} is generally assumed to be independent of the environment (Zucker et al. 2013) and computed in the configuration space.

We, therefore, only deal with f_{obs} depicted in the following equation:

$$f_{obs}(\xi) = \int_0^1 c(\xi(t)) \parallel \frac{d}{dt}\xi(t) \parallel dt. \tag{5}$$

Unlike the original form in Eq. 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 Zucker et al. (2013):

$$c(\xi(t)) = \begin{cases} -\mathfrak{D}(\xi(t)) + \frac{1}{2}\varepsilon, & \mathfrak{D}(\xi(t)) < 0 \\ \frac{1}{2\varepsilon}(\mathfrak{D}(\xi(t)) - \varepsilon)^2, & 0 < \mathfrak{D}(\xi(t)) \leq \varepsilon \\ 0, & otherwise, \end{cases} \tag{6}$$

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

$$\mathfrak{D}(\xi(t)) = -\min_{\forall q \in V}(\|\xi(t) - q\| - \omega(|V^*|) \cdot r_q). \tag{7}$$

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. 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. 5.

Fig. 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:

$$\nabla f_{obs}(\xi) = \|\xi'\|\cdot[(I-\hat{\xi}'\cdot\hat{\xi}'^{T})\nabla c - c\kappa], \tag{8}$$

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

$$p = \arg\min_{q\in V}(\|\xi(t)-q\| - \omega(|V^*|)\cdot r_q) \tag{9}$$

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. 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, 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 a reasonable choice for faster optimization in practice.

Algorithm 5: BATCH DANCING TREE*

```
1  V ← {q_init, q_goal}; E ← ∅
2  X_samples ← {q_goal};
3  Q_V ← ∅; Q_E ← {(q_init, q_goal)}
4  while Termination condition is not satisfied do
5      if Q_V ≡ and Q_E ≡ ∅ then
6          X_samples ← SampleCollisionFree(m_batch)
7          V_x ← ∅, ∀x ∈ X_samples
8          Q_V ← V
9      while BestQueueValue(Q_V) ≤ BestQueueValue(Q_E) do
10         v_best ← BestInQueue(Q_V)
11         Q_near ← Near(v_best, V); X_near ← 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∈Q_near}, and v_best to V_{x∈X_near}
15         ExpandVertex(v_best, Q_near, X_near)
16     ProcessEdge(BestInQueue(Q_E))
17 return SolutionPath(G)
```

Algorithm 6: PROCESSEDGE ($(v,x)\in Q_E$), $v\in V, x\in X_{samples}$.

```
1  Pop (v,x) from Q_E
2  if g_G(v) + (|(v,x)|)^ + ĥ(x) < g_G(q_goal) then
3      if IsCollisionFree((v,x)) then
4          σ(v,x) = (v,x)
5      else
6          σ(v,x) = Optimize((v,x))
7      if IsCollisionFree((v,x)) ∧ ĝ(v) + |σ(v,x)| + ĥ(x) <
         g_G(q_goal) then
8          if g_G(v) + |σ(v,x)| < g_G(x) then
9              E ← E ∪ σ(v,x)
10             Update V, X_samples and Q_E
11 else
12     Q_V ← ∅; Q_E ← ∅
```

Back to the pesudocodes of Dancing PRM*, $UpdateShortestPathTree(\cdot)$ (Line: 12 in Alg. 2 and line: 13 in Alg. 4) 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 (Hauser 2015) 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.

3.4 Batch Dancing Tree*

The integration of BIT* with our proposed algorithm can be viewed as a variation of RABIT* (Choudhury et al. 2016),

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 (Gammell et al. 2015), which results in an efficient lazy graph expansion.

In this algorithm, we consider BT* (Batch Tree) as a simplified version of BIT* (Gammell et al. 2014) 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. 5, 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. 2, $SampleCollisionFree(m_{batch})$ 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) 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. 6 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. 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. 4.4 to analyze their properties.

4 Experiments

4.1 Experiment setup

For a 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, Sucan et al. 2012).

For visualization and framework integration, we use V-REP simulator (E. Rohmer 2013). 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.

4.2 Comparison against RABIT*

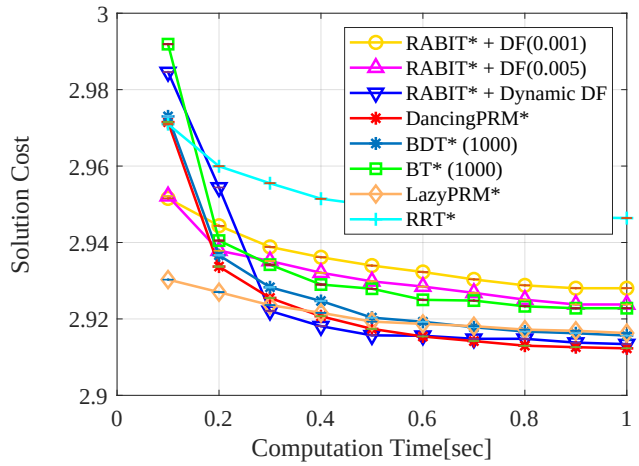
We first compare the performance of DancingPRM* and BDT* against RABIT*, which is applicable only when the explicit representation of $\mathbb{X}_{free}$ is available, i.e., the analytic computation of obstacle potential is available.

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*.

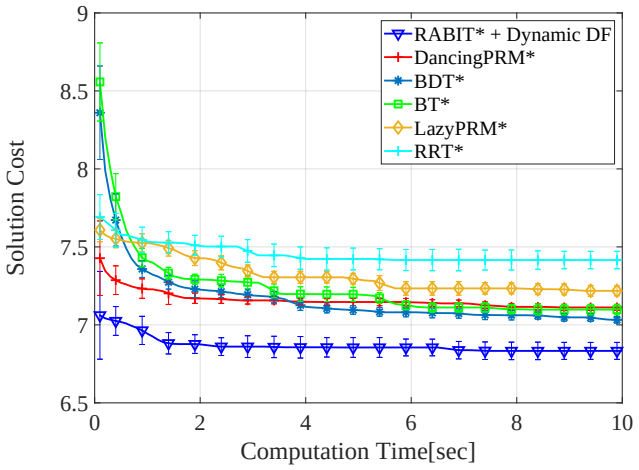
For this test, our evaluation benchmark is constructed by following the ones used in the original paper of RABIT*. Specifically, we consider two synthetic scenes, which are $\mathbb{R}^2$ and $\mathbb{R}^8$ configuration spaces where a wall with ten narrow passages are located at the center in a d-dimensional 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. 6 shows our experimental results of the solution cost as a function of computation time, and the error bars indicate the variance of each algorithm. The value in the parentheses of BDT* means the number of samples per batch, and the



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



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

Fig. 6 Performance comparison over computation time for different algorithms in synthetic benchmarks. Results are averaged over 30 trials. We show the average results with min and max bars in the graph; min and max bars can be invisible, when they are very close to the average values. 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.

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. 4.4.

Under our experimental setting, "RABIT* + DF(α)" uses a d-dimension distance field with 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.

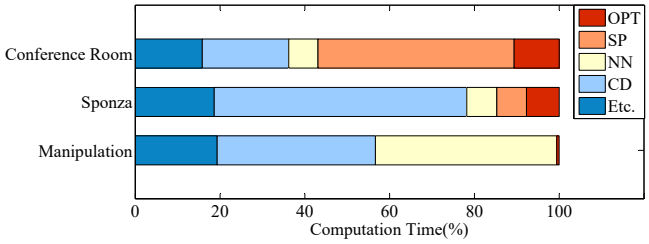


Fig. 7 Computation time breakdown of the proposed algorithm DancingPRM* measured in our benchmarks (Fig. 8). Each abbreviation in legend stands for OPTimization (OPT), Shortest Path tree update (SP), Nearest Neighbor search (NN), and Collision Detection (CD). Note that the computation time for $\widetilde{\mathbb{X}}_{free}$ construction is negligible ($< 1\%$) for all of three benchmarks.

Fig. 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 as negative for the proposed algorithms; especially, $\mathbb{R}^8$ seems to show our limitation in a high-dimensional problem. In prac-

tice, 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.

4.3 Comparison in practical benchmarks

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

Fig. 8(a), 8(c), 8(e) show comparison results tested in our benchmarks illustrated in Fig. 8(b), 8(d) and 8(f), where the configuration space are $\mathbb{R}^2$, $SE(3)$ and $\mathbb{S}^6$, respectively. Fig. 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. 8(d) and 8(f) has a relatively higher computation cost for collision checking as observed in Fig. 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 tend to have a lower variance compared to their original forms, BT* and Lazy PRM*, as shown in Fig. 8(c) and 8(e). Specifically, Dancing PRM* has a variance of 0.0326 at the end of execution, while its original form, Lazy PRM* shown in Fig. 8(e), has 0.0435, which is approximately 30 % higher than ours. 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 improved connectivity can improve not only the convergence speed, but also stability of the solution cost 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.

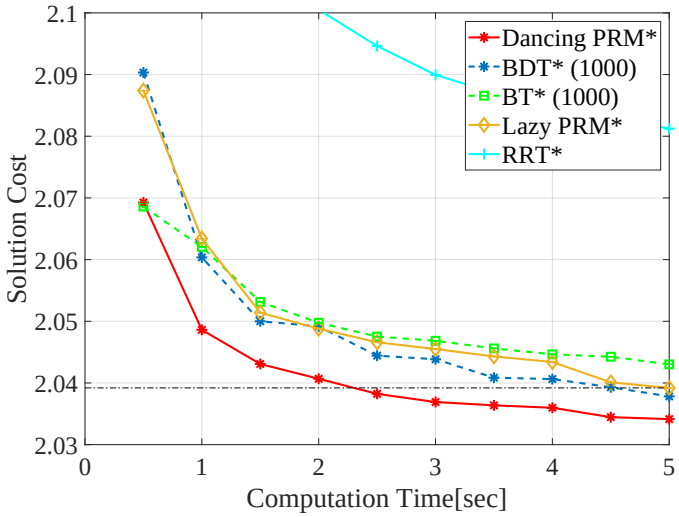
4.4 Comparisons with varying batch sizes

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 Gammell et al. (2015). For a better understanding of BDT*, we discuss how BDT* and Dancing PRM* work differently and the effect of batch sizes in this subsection.

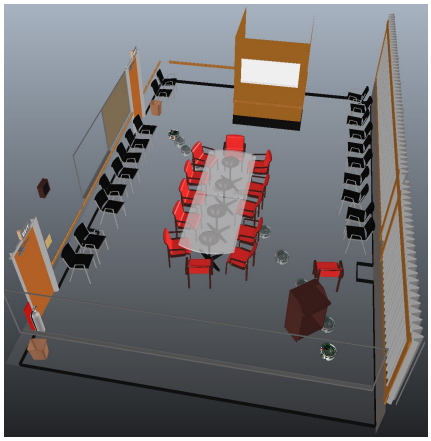
In Fig. 9, we measure the performance with varying batch sizes from 1 to 1000 in the $\mathbb{R}^2$ benchmark (Fig. 8(b)). As we can observe, there is a performance improvement as the number of samples per batch increases. Also, the core of Dancing PRM*, i.e., local trajectory optimization with configuration-free 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. 10. Compared to Fig. 7, the difference can be summarized by follows:

(1) In Dancing PRM*, so-called *optimistic thrashing* (Hauser 2015; Kim et al. 2018a) poses an additional overhead on shortest path computation (SP). Its effect is maximized in the 2D benchmark (Fig. 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 computation overhead of local trajectory optimization is identical regardless of the length of the trajectory, unlike collision checking.

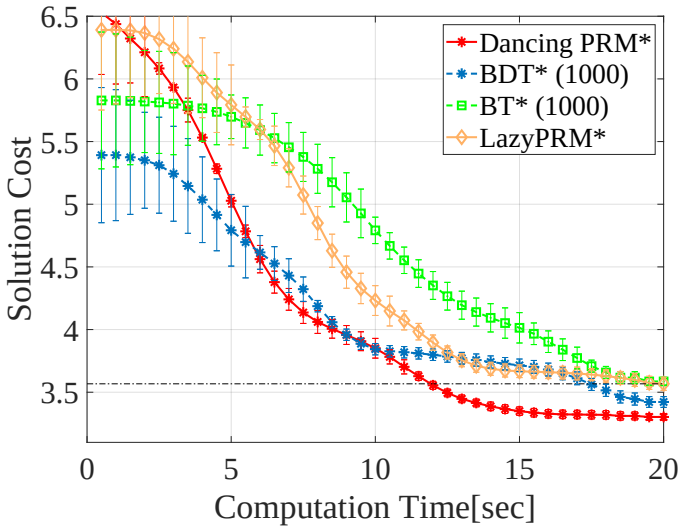
The most important point is that the use of DSPT (Dynamic Shortest Path Tree) gives rise to the optimistic thrashing problem, causing a more overhead on the shortest path computation. Meanwhile, LPA*-based expansion requires more edge collision checking than the DSPT-based approach, because the edges satisfying the constraints at line 2 in Alg. 6, i.e., $g_G(v) + \widehat{|(v,x)|} + \hat{h}(x) < g_G(q_{goal})$ are checked for collision, where $\hat{\cdot}$ stands for an admissible estimation.



(a) Performance vs. time

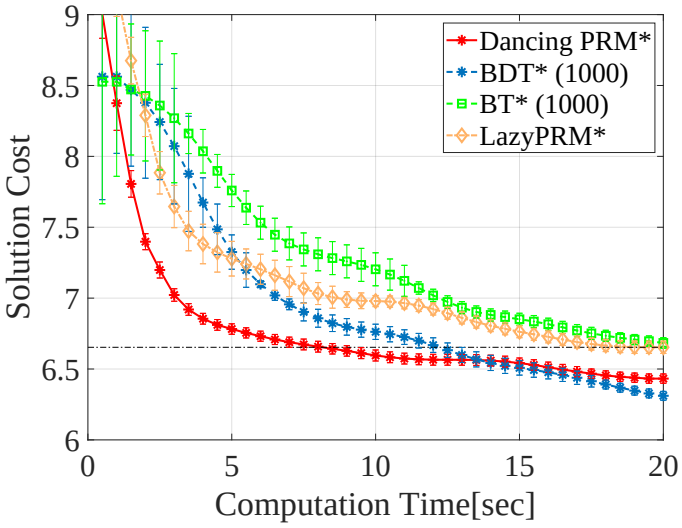


(b) Conference Room : $\mathbb{R}^2$

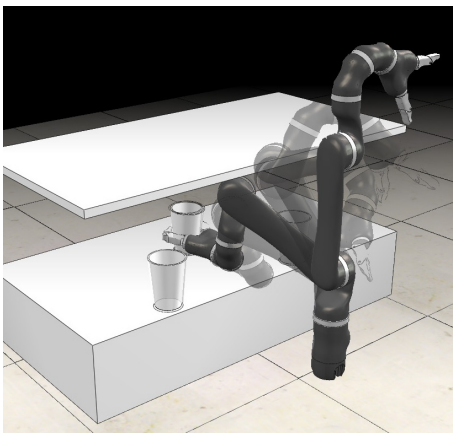


(c) Performance vs. time

(d) Sponza : $SE(3)$



(e) Performance vs. time



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

Fig. 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 the visualization of benchmarks. Results are averaged over 30 attempts, and RRT* is not reported for Fig. 8(c) and 8(e) due to a high performance gap. The error bars stand for the variance of solution costs.

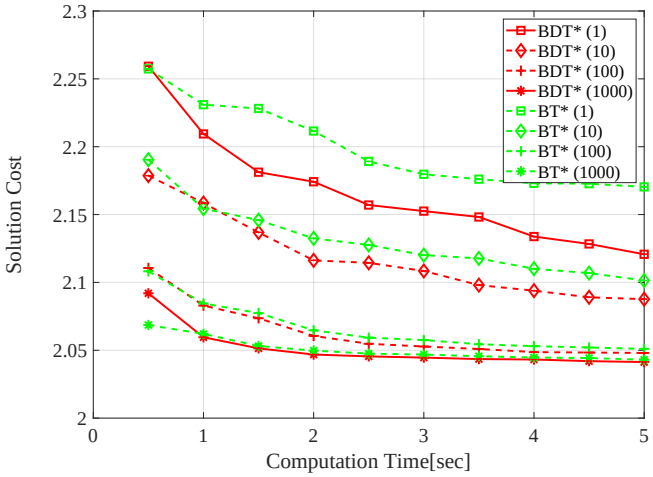


Fig. 9 Performance comparison with varying the number of samples per batch in BDT*. The results are averaged over 30 attempts and tested in Fig. 8(b). A higher number of samples per batch provides a better convergence speed at the expense of anytime property.

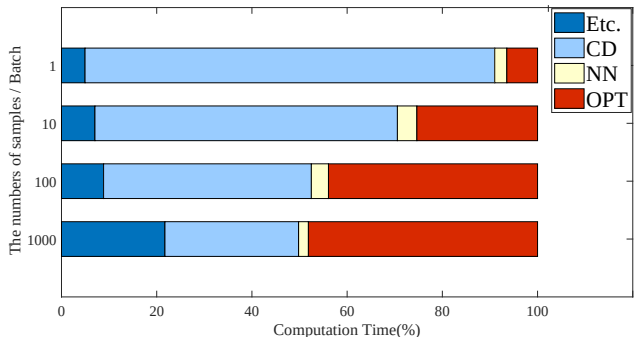


Fig. 10 Computation time breakdown of Batch Dancing Tree* tested in Fig. 8(b). The result shows the distribution of computation time spent on each component over varying batch size. Note that the majority of *Etc.* is a graph expansion based on LPA*, which corresponds to *SP* in Fig. 7. The overhead for the configuration-free space approximation is measured $< 3\%$ in this experiment.

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. 6, 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 different benchmarks.

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.

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* (Hauser 2015), 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. 4, 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. 4). 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. 1 and 2, 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 Hauser (2015).

For BDT*, it considers more edges due to the weak condition for local planning as discussed in Sec. 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 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 of the environment.

5.2 Computational Complexity

The complexity and analysis of primitive operations used for our method follow the discussion in Kleinbort et al. (2016). We thus deal with overheads introduced mainly by the proposed algorithms in this section.

Time Complexity In Alg. 2, 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. 3.2) ; thus the time complexity of the entire while loop in Alg. 2 and the while loop for $ExpandVertex(\cdot)$ in Alg. 5 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. 2 and 6 in Alg. 5), 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 that sampling phase compared to RRG (Karaman and Frazzoli 2011).

The lazy collision checking is performed in both Dancing PRM* and BDT* in Alg. 4 and 6, 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. 5 and 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. 3.2.

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. 7 and 10. For this reason, its complexity heavily depends on the cost

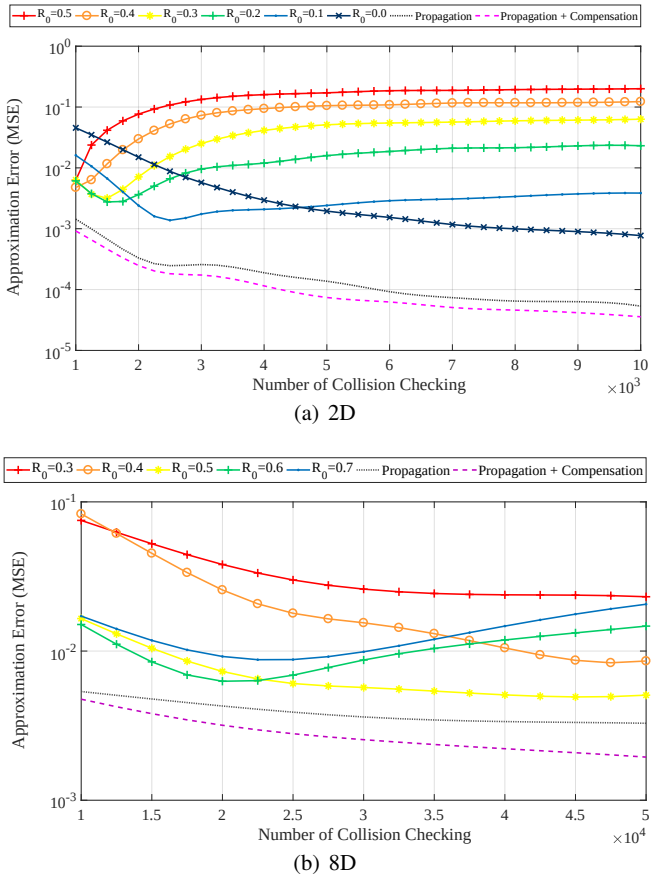


Fig. 11 Mean squared errors measured at intermediate configurations during the local optimization by both our configuration-free space approximation and the ball-tree algorithm with different initial R_0 values. Ours shows a monotonic decreasing result, while the ball-tree with fixed initial radii tends 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.

of the solution path, $g_G(q_{goal})$, during the execution, which is the bounds for the edge expansion as discussed in Sec. 4.4.

Memory Complexity The proposed algorithm additionally maintains the near neighbor set, $V_v, \forall v \in V$ in Sec. 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 Otte and Frazzoli (2015) where the planner culls out neighbors outside the ball of radius r centered at v, where r for r-nn search (Sec. 2.1).

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. 3.2 by measuring the error of our approximate free space $\widetilde{\mathbb{X}}_{free}$ against the inexact version of the ball-tree algorithm (Shkolnik and Tedrake 2011). 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. 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. 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. 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. 4 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. 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. Meanwhile a sample configuration with our approximation is capable of inheriting those information from its near neighbors that can be accessed without having any additional overhead in the nature of sampling-based motion planner.

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. 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 option 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 the planning process, demonstrating the robustness experimentally.

5.4 Radius compensation and dispersion

The purpose of radius compensation (Sec. 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 (Karaman and Frazzoli 2010). 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:

$$\delta(P) = \sup_{x \in X} \{ \min_{p \in P} \{ \rho(x, p) \} \}. \tag{10}$$

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 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 Deheuvels (1983) and Niederreiter (1992). It also has been studied in the perspective of sampling-based motion planning (Karaman and Frazzoli 2011; Lindemann and LaValle 2005) 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, we presented a radius compensation for the local trajectory optimization, i.e.,

$$\omega(|V^*|) \cdot r_q, \text{ where } \omega(n) = 1.0 - \zeta \cdot \delta(n). \tag{11}$$

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. 12 shows the concept of dispersion in an illustrative way.

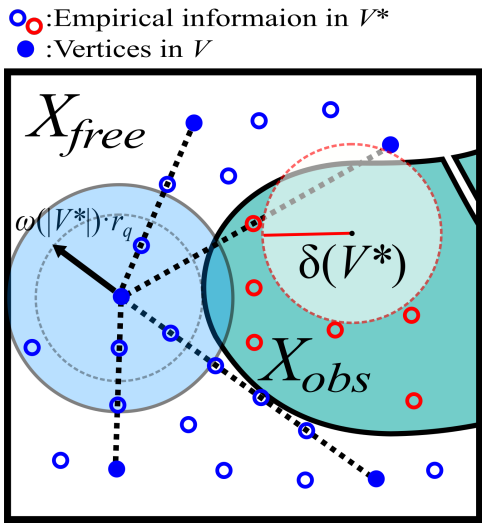


Fig. 12 A concept of dispersion for the sample set V^* and the radius compensation. V^* consists of all sampled configurations in both $\mathbb{X}_{free}$ (blue) and $\mathbb{X}_{obs}$ (red) regardless of feasibility and including samples for edge collision checkings (dotted segments). The radius of 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).

We use $\zeta = 0.3$ as explained in Sec. 4, which is chosen experimentally and provides the best performance regardless of the dimension of given problems.

Finally, the result of Fig. 11 provably shows that the radius compensation improves the accuracy of our configuration-free space approximation with a negligible overhead as revealed in Fig. 7 and 10.

6 CONCLUSION and FUTURE WORK

In this paper, 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. On the other side, they exploit the learned spatial information to optimize local trajectories around narrow passages or the boundary space.

The proposed algorithm does not depend on the precomputed space information or expensive proximity computation for obstacle potentials throughout the entire process for 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.

The next research direction includes the accuracy and representation issue for the configuration-free space. Even though the hypersphere-based representation is simple and effective in low or moderate dimensional spaces, it would not yet be an all-purpose solution. Furthermore, as observed from Fig. 11, the most of local trajectory optimizations at an early phase would be unsuccessful with premature space information, while the search graph in the later phase will be getting dense enough to represent fine-grained trajectories without the local optimization as well. We, therefore, expect that study on the conditional activation of local trajectory optimization can be beneficial for further improvement.

Acknowledgements We appreciate the anonymous reviewers for constructive comments and insightful suggestions. This work was partially supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT, No. 2019R1A2C3002833), Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT (NRF-2017M3C4A7066317), and Defense Acquisition Program Administration and Defense Industry Technology Center under the contract UC160003D, Korea.

References

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

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

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

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

Burns, Brendan, & Oliver Brock (2005b). Toward Optimal Configuration Space Sampling. *In RSS*, Cambridge, USA.

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

Deheuvels, Paul (1983). Strong bounds for multidimensional spacings. *Probability Theory and Related Fields*, 64(4):411–424.

Denny, Jory, & Nancy M. Amato (2011). 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)*, pp 2632–2639.

Denny, Jory, & Nancy M. Amato (2012). The Toggle Local Planner for sampling-based motion planning. *In IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, pp 1779–1786.

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

E. Rohmer, M. Freese, S. P. N. Singh (2013). V-REP: a Versatile and Scalable Robot Simulation Framework. *In IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems (IROS)*.

Frigioni, Daniele, Alberto Marchetti-Spaccamela, & Umberto Nanni (2000). Fully dynamic algorithms for maintaining shortest paths trees. *Journal of Algorithms*, 34(2):251–281.

Gammell, Jonathan D, Siddhartha S Srinivasa, & Timothy D Barfoot (2014). Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. *In Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, pp 2997–3004.

Gammell, Jonathan D, Siddhartha S Srinivasa, & Timothy D Barfoot (2015). 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)*, pp 3067–3074.

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

Hauser, Kris (2015). Lazy Collision Checking in Asymptotically-Optimal Motion Planning. *In IEEE Int'l. Conf. on Robotics and Automation (ICRA)*.

Jaromczyk, Jerzy W, & Godfried T Toussaint (1992). Relative neighborhood graphs and their relatives. *Proceedings of the IEEE*, 80(9):1502–1517.

Jeon, Jeong Hwan, Sertac Karaman, & Emilio Frazzoli (2011). Anytime computation of time-optimal off-road vehicle maneuvers using the RRT*. *In IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, pp 3276–3282.

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

Karaman, Sertac, & Emilio Frazzoli (2010). Incremental sampling-based algorithms for optimal motion planning. *arXiv preprint arXiv:1005.0416*.

Karaman, Sertac, & Emilio Frazzoli (2011). Sampling-based algorithms for optimal motion planning. *Int'l. Journal of Robotics Research (IJRR)*, 30(7):846–894.

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

Kavraki, Lydia E, Petr Svestka, J-C Latombe, & Mark H Overmars (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580.

Kim, Donghyuk, Youngsun Kwon, & Sung-eui Yoon (2018a). Adaptive Lazy Collision Checking for Optimal Sampling-based Motion Planning. *In Int'l. Conf. on Ubiquitous Robots*, pp 2519–2526.

Kim, Donghyuk, Youngsun Kwon, & Sung-eui Yoon (2018b). Dancing PRM* : Simultaneous Planning of Sampling and Optimization with Configuration Free Space Approximation. *In IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, pp 2519–2526. IEEE.

Kleinbort, Michal, Oren Salzman, & Dan Halperin (2016). 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)*.

Koenig, Sven, Maxim Likhachev, & David Furcy (2004). Lifelong planning A*. *Artificial Intelligence*, 155(1-2):93–146.

Kuntz, Alan, Chris Bowen, & Ron Alterovitz (2017). Fast Anytime Motion Planning in Point Clouds by Interleaving Sampling and Interior Point Optimization. *In Proc. International Symposium on Robotics Research (ISRR)*, pp 1–16.

LaValle, Steven M (1998). Rapidly-Exploring Random Trees: A new Tool for Path Planning. Technical Report 98-11, Iowa State University.

LaValle, Steven M (2006). *Planning algorithms*. Cambridge university press.

Lindemann, Stephen R, & Steven M LaValle (2005). Current issues in sampling-based motion planning. *In Robotics Research. The Eleventh International Symposium*, pp 36–54. Springer.

Niederreiter, Harald (1992). *Random number generation and quasi-Monte Carlo methods*, volume 63. SIAM.

Otte, Michael, & Emilio Frazzoli (2015). RRT-X: Real-Time Motion Planning/Replanning for Environments with Unpredictable Obstacles. In Algorithmic Foundations of Robotics XI, pp 461–478. Springer.

Pan, Jia, & Dinesh Manocha (2016). Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing. *Int'l. Journal of Robotics Research (IJRR)*, 35(12):1477–1496.

Park, Chonhyon, Jia Pan, & Dinesh Manocha (2012). ITOMP: Incremental Trajectory Optimization for Real-Time Replanning in Dynamic Environments. *In Int'l Conf. on Automated Planning and Scheduling*.

Rickert, Markus, Arne Sieverling, & Oliver Brock (2014). Balancing exploration and exploitation in sampling-based motion planning. *IEEE Transactions on Robotics (T-RO)*, 30(6):1305–1317.

Shkolnik, A., & R. Tedrake (2011). Sample-based planning with volumes in configuration space. *arXiv preprint arXiv:1109.3145*.

Sucan, Ioan A, Mark Moll, & Lydia E Kavraki (2012). The open motion planning library. *"IEEE Robotics & Automation Magazine."*, 19(4):72–82.

Zucker, Matt, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, & Siddhartha S Srinivasa (2013). CHOMP: Covariant Hamiltonian optimization for motion planning. *Int'l. Journal of Robotics Research (IJRR)*, 32(9-10):1164–1193.