2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Cloud RRT*: Sampling Cloud based RRT*

Donghyuk Kim[1], Junghwan Lee[2] and Sung-eui Yoon[3]

***Abstract*— We present a novel biased sampling technique, Cloud RRT*, for efficiently computing high-quality collision-free paths, while maintaining the asymptotic convergence to the optimal solution. Our method uses sampling cloud for allocating samples on promising regions. Our sampling cloud consists of a set of spheres containing a portion of the C-space. In particular, each sphere projects to a collision-free spherical region in the workspace. We initialize our sampling cloud by conducting a workspace analysis based on the generalized Voronoi graph. We then update our sampling cloud to refine the current best solution, while maintaining the global sampling distribution for exploring understudied other homotopy classes. We have applied our method to a 2D motion planning problem with kinematic constraints, i.e., the Dubins vehicle model, and compared it against the state-of-the-art methods. We achieve better performance, up to three times, over prior methods in a robust manner.**

I. INTRODUCTION

Many motion planning algorithms have been proposed to find a collision-free trajectory given constraints (e.g., kinematics of vehicles). Among proposed techniques sampling-based algorithms such as PRM [1] and RRT [2] have been widely used for multiple and single queries, because of their capability of solving high-dimensional motion planning problems. These algorithms have been shown to guarantee probabilistic completeness. Thanks to their theoretical simplicity and possibility of extension to other related problems they have encouraged many follow-on researches.

Most early works for sampling-based techniques focused on finding the existence of a solution for their problems. Recently RRT* [3] was proposed to achieve convergence to the optimal solution and thus has been drawing a lot of attention in finding the optimal solution for single-query motion planning problems. In particular, for mobile robots such as autonomous vehicles(e.g., Google self-driving car) it is important to compute a shorter path to a given goal for saving the limited fuel/power.

Some of recent works [4], [5], [6] introduced different sampling heuristics to accelerate the convergence to the optimal solution, while maintaining the optimality guarantee of RRT*. These techniques proposed an explicit or implicit way 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. V). Especially for environments with complex configurations of obstacles and narrow passages, these prior techniques may not work well.

Main contributions. In this paper we propose Cloud RRT* to achieve a better convergence to the optimal solution, while maintaining the asymptotic convergence of the original RRT*. Our method introduces *sampling cloud* as a decomposition of the configuration space of the robot (Sec. IV-A) and generates samples according to the sampling cloud. Our sampling cloud consists of a set of spheres, each of which is projected on a collision-free spherical region in the workspace. To initialize our sampling cloud, we construct Generalized Voronoi Graph (GVG) as a geometric analysis on the workspace (Sec. IV-B) and generate an initial set of spheres to cover computed GVG. We then update our sampling cloud as we identify a better solution to the current one (Sec. IV-C). We design our update method to locally sample more for refining the current best solution, while maintaining the global sampling distribution for exploring less studied other homotopy classes. We also prune sampling spheres that cannot contribute to shorten the current optimal solution (Sec. IV-D).

II. RELATED WORK

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.

A. Sampling based Motion Planning

Sampling based motion planning has been well studied and a few good books/surveys are available [7], [8]. Among these techniques, PRM [9] and RRT [10] are the most well known algorithms. The basic concept of PRM is to take random samples in a given configuration space, to check whether they are collision-free, and to use a local planner to connect each other configurations for building the roadmap that is used for runtime queries. On the other hand, the RRT algorithm can be considered as a random search biasing towards exploring largest Voronoi regions. It samples a random configuration at each iteration, and then attempts to connect it to the closest configuration in the tree.

RRT is generally known as a more suitable method for the single query problem, while PRM is for the multi query. For this reason RRT has been preferred for a real-time motion planning against dynamic environments [2]. RRT* [3] is a recent extension of the RRT algorithm. Its main characteristics is that it guarantees asymptotic optimality without having substantial computational overheads. Furthermore it can work with numerous existing extensions such as bi-directional RRT [10], anytime RRT [11]. In particular

[1]Donghyuk Kim, [2]Junghwan Lee, and [3]Sung-eui Yoon are at Dept. of CS, KAIST, Daejeon, South Korea; donghyuk.kim@kaist.ac.kr, goolbee@gmail.com, sungeui@gmail.com

anytime RRT* [12] is an execution-time replanning algorithm for RRT* and progressively converges toward the optimal by exploiting additional execution time. We are mainly interested in computing collision-free paths for mobile robots and thus we build our proposed method based upon the RRT* approach that can compute the optimal path for cluttered environments.

B. Workspace Analysis

Workspace analysis [13], [14], [15] 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 [16], [17], [18], utilizing free space information [19], and retraction-based approaches [20] 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) [14], [13], uniform grid [21], and quad/octree [15]. 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 [22] 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 [23], [24] 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.

C. 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 [4], [5], [6]. Meanwhile, [25] 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 [26], [27] to get more information from the given space. It is well known that there is a trade-off relationship between exploitation and exploration [26], and the balance control is a key to address the optimality issue [5].

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.

III. 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.

A. 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:

$$\dot{x}(t) = f(x(t), u(t)), \quad x(0) = x_0, \tag{1}$$

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 $g : \tau(\mathbf{t}) \rightarrow \mathbb{R}$ be a cost function that associates a non-zero cost, where $\tau : [0:T] \rightarrow X_{free}$ 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. 1, the motion planning problem is to find a continuous path, $\tau : [0,T] \rightarrow \mathbf{X}_{free}$ 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 paper 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.

B. 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 [4], [5], [6] 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.

C. 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 $\mathscr{S}$.

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.

IV. ALGORITHMS

In this section we discuss each component of our method.

A. 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 $\mathscr{S}$ 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:

$$\mathbf{X}_s := \{x \mid Proj(x) \in s \wedge x \in \mathbf{X}\}, \tag{2}$$

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 $\mathscr{S}$. 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 $\mathscr{S}$ instead of the C-space $\mathbf{X}$. Specifically we first choose a sphere s among $\mathscr{S}$ 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.

B. 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 [28]. 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.

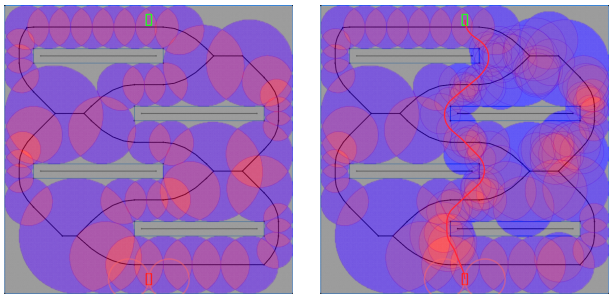
Many different construction techniques for GVG have been available for two and three dimensional cases [29], [30]. Some of them can be performed efficiently on GPUs for two and three dimensional workspaces [30]. For our problem we assume that input environments are two dimensional and consist of points, 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 1-(a).

Algorithm 1: GVG-BASED INITIALIZATION

Input: Obs, a set of obstacles
Output: S, a set of collision-free spheres

1 $(V_{GVG}, E_{GVG}) \leftarrow ConstructVoronoiGraph(Obs)$
2 $\mathscr{S} \leftarrow \emptyset$; $Q \leftarrow \emptyset$
3 $v_{init} \leftarrow FindVisiblePoint(V_{GVG}, E_{GVG})$
4 $r_{init} \leftarrow DistanceToClosestObstacle(v_{init}, Obs)$
5 $Q.push((v_{init}, r_{init}))$ // push a sphere with v_{init} and r_{init}
6 **while** Q *isNotEmpty* **do**
7 $(v, r) \leftarrow Q.pop()$ // fetch a sphere
8 $V \leftarrow ConstructNeighborSpheres((v, r), V_{GVG}, E_{GVG})$
9 **for** ***each*** $v \in V$ // *per each intersected vertex* **do**
10 **if** $IsNotContainedInOtherSpheres(v, \mathscr{S})$ **then**
11 $r \leftarrow DistanceToClosestObstacle(v, Obs)$
12 $\mathscr{S} \leftarrow \mathscr{S} \cup \{v, r\}$
13 $Q.push((v, r))$
14 **return** $\mathscr{S}$



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

Fig. 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.

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. 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. 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. 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.

C. 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 $\mathscr{S}_c$; the subscript c of $\mathscr{S}_c$ 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 $\mathscr{S}_c$. We therefore compute the radius, r_n, of n by reducing radii of $\mathscr{S}_c$ by a factor of $\alpha \in (0,1)$. We then set the importance, i_n, of the new sphere to be computed relatively to importances of $\mathscr{S}_c$ based to their radii. In summary, r_n and i_n of the new sphere n are defined as the following:

$$r_n = \frac{1}{|\mathscr{S}_c|} \sum_{s \in \mathscr{S}_c} r_s \cdot \alpha, \tag{3}$$

$$i_n = \frac{1}{|\mathscr{S}_c|} \sum_{s \in \mathscr{S}_c} \frac{i_s \cdot r_n}{r_s + r_n}. \tag{4}$$

We then reduce the importance of $\mathscr{S}_c$ 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:

$$\Delta i_s = \frac{i_s \cdot r_n}{(r_s + r_n) \cdot |\mathscr{S}_c|}, \forall i_s \in \mathscr{S}_c. \tag{5}$$

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:

$$\theta_n = \frac{1}{|\mathscr{S}_c|} \sum_{s \in \mathscr{S}_c} \theta_s \cdot \alpha. \quad (6)$$

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 $\mathscr{S}_c$, while maintaining the global sampling distribution to other regions (Fig. 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.

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.

D. 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 $\mathscr{S}$ that cannot provide a better solution than the current one. 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.

E. 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.

Algorithm 2: Cloud RRT*

Input: A sampling cloud $\mathscr{S}$, an init. configuration, q_{init}, and a goal region, Q_{goal}
Output: A random tree, $\mathscr{T}$, and a solution path, Q_{sol}

$\mathscr{T} \leftarrow \{q_{init}\}$ and $Q_{sol} \leftarrow \emptyset$
while ***not*** *termination conditions are satisfied* **do**
  s$\leftarrow$ $SampleSphere(\mathscr{S})$
  $q_s \leftarrow Sample(\text{s})$
  $q_n \leftarrow Nearest(q_s)$
  $(q_{new}, u_{new}) \leftarrow Steer(q_n, q_s)$
  if $IsCollisionFree(q_{new}, u_{new}, q_n)$ **then**
    $Q_{near} \leftarrow Near(q_{new})$
    $q_{min} \leftarrow ChooseParent(Q_{near}, q_n, q_{new})$
    $\mathscr{T} \leftarrow InsertNode(q_{min}, q_{new}, \mathscr{T}, s)$
    $\mathscr{T} \leftarrow Rewire(\mathscr{T}, Q_{near}, q_{min}, q_{new})$
    if *A better solution is found* **then**
      $Q_{sol} \leftarrow UpdateSolution(\mathscr{T})$
      $M \leftarrow UpdateMilestone(Q_{sol})$
      $\mathscr{S} \leftarrow UpdateSamplingCloud(\mathscr{S}, M)$
return $\mathscr{T}$

We first choose a sphere, s, from the sampling cloud $\mathscr{S}$ according to importance values of spheres, and prune s, if possible. We then generate a random sample, q_s [3-4:Alg. 2]. We then 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, we 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. IV-C.

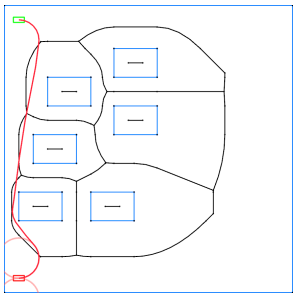
V. RESULTS

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 [31] for our GVG-guided initialization.

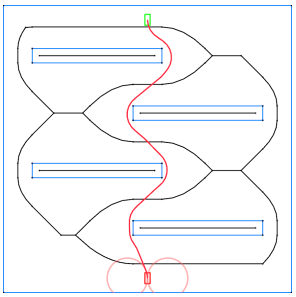
To demonstrate benefits of our method, we have compared our method with the original RRT*, RRT*-Smart [4], and local biasing and node rejection techniques, denoted as LN, proposed by Akgun et al. [5]. We have tested different methods for a kinematic motion planning problem with the Dubins vehicle model [32].

A. Dubins Vehicle

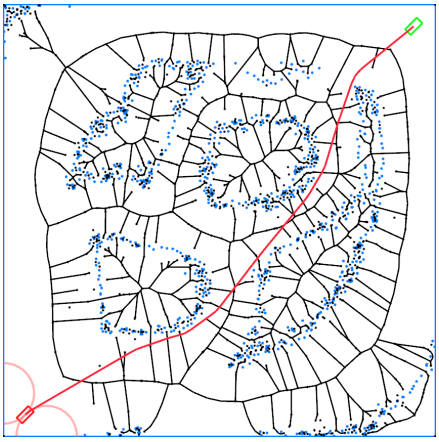
We aim to generate optimal collision-free paths in an efficient manner for autonomous vehicle models. For the problem we use the Dubins vehicle model and need to respect kinematics of the model for computing collision-free paths. While our initial path is computed by only considering the geometry of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



(a) Six squares



(b) Four squares



(c) Many points

Fig. 2. 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.

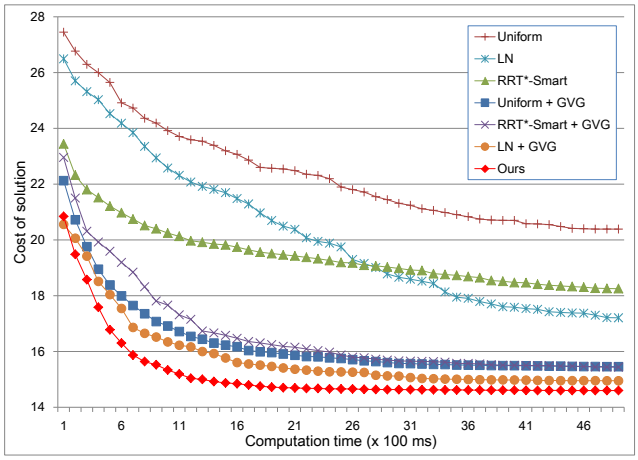
to demonstrate how our method behaves well with such kinematic constraints.

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:

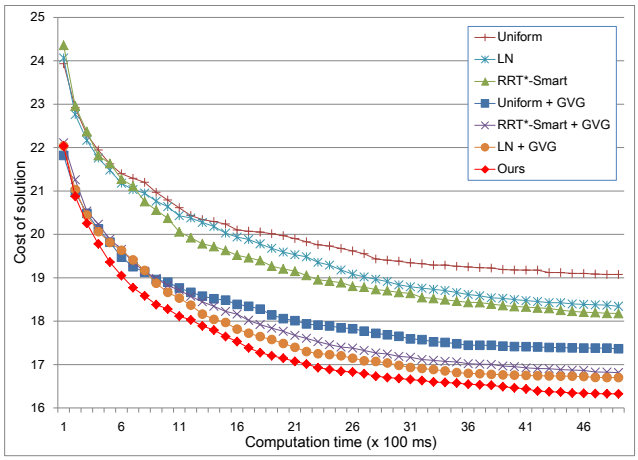
$$\begin{aligned} \dot{x} &= v_c cos\theta, \\ \dot{y} &= v_c sin\theta, \\ \dot{\theta} &= \frac{v_c}{L} \cdot tan(u), \end{aligned} \tag{7}$$

where v_c is a constant speed of the vehicle, and u is a steering input within $U = [-\phi, \phi]$. Its C-space is $\mathscr{C} = \mathbb{R}^2 \times \mathbb{S}$, 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) 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. [33]. We have tested three different environments (Fig. 2) 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



(a) Six squares



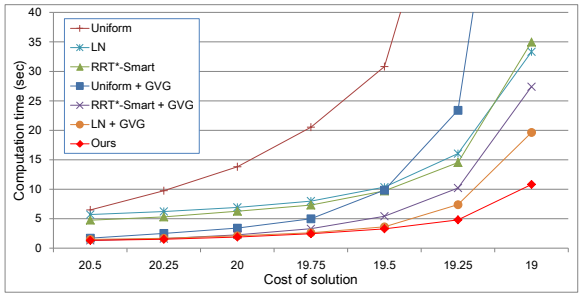
(b) Four squares

Fig. 3. 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.

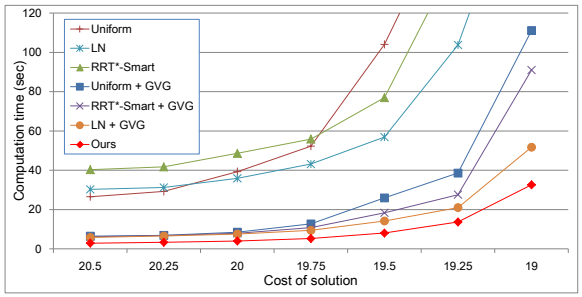
with GVG, we use our initial sampling cloud computed from GVG (Sec. IV-B), but do not use our update nor pruning methods. By comparing our method with those prior methods combined with GVG, we can observe benefits of our update and pruning methods.

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. 3 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 op-



(a) Four squares



(b) Many points

Fig. 4. Graphs showing the time required to reach a cost of solutions shown in the X-axis.

erations 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. 4 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. 3, 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.

TABLE I

VARIOUS STATICS 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

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. 3(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. 3(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. 4(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 I showing various results including a cost of the 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 II 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

TABLE II

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

spheres, N_{init}, constructed by the GVG-guided initialization.

VI. CONCLUSION

We have proposed Cloud RRT*, which utilizes sampling cloud, a set of sampling spheres initialized by GVG. Our update method for the sampling cloud is based on newly identified configurations of better solutions 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 have also proposed a simple pruning method for spheres to cull out unpromising spheres. We have applied our method to the 2D motion planning problem with kinematic constraints. We have found that our method shows higher performance and better convergence rate over the state-of-the-art methods in a robust manner.

There are many interesting research avenues. In this work we have mainly tested with 2D problems. It requires additional efforts to handle three dimensional problems. All the concepts and operations of sampling clouds are applicable to 3D workspace. On the other hand, GVG in 3D has problems of disconnectivity [28] and can be slow because of its time complexity. Fortunately, improving GVG has been well studied. Hierarchical GVG and GPU-based efficient approximation techniques are available [28], [30] for addressing aforementioned issues. Nonetheless, integrating them within our method in a way for achieving high performance is left for future work. We would like to also investigate efficient geometry/homotopy analysis and maintain multiple local optimum located in different homotopy classes within our sampling cloud.

ACKNOWLEDGMENT

We would like to thank SGLab members and anonymous reviewers for constructive comments. This work was supported in part by NRF-2013R1A1A2058052, DAPA/ADD (UD110006MD), MEST/NRF (2013-067321), and IT R&D program of MOTIE/KEIT [10044970]. Sung-eui Yoon is a corresponding author of the paper.

REFERENCES

[1] L. Guibas, C. Holleman, and L. Kavraki, "A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach", in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 1999, pp. 254 – 259.

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

[3] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning", in *Robotics: Science and Systems*, 2010.

[4] 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. Conf. on*, 2012, pp. 1651–1656.

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

[6] R. Luna, I. Sucan, M. Moll, and L. Kavraki, "Anytime solution optimization for sampling-based motion planning", in *ICRA*, 2013.

[7] S. M. LaValle, *Planning Algorithms*, Cambridge University Press, 2006.

[8] Bruno Siciliano and Oussama Khatib, *Springer handbook of robotics*, Springer, 2008.

[9] L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces", *IEEE Trans. Robot. & Automat.*, pp. 12(4):566–580, 1996.

[10] J. Kuffner and S.M. LaValle, "Rrt-connect: An efficient approach to single-query path planning", in *ICRA*, 2000, pp. 995–1001.

[11] Dave Ferguson and Anthony Stentz, "Anytime rrts", in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2006.

[12] S. Karaman, M. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt*", in *ICRA*, 2011, pp. 1478–1483.

[13] Roland Geraerts, "Planning short paths with clearance using explicit corridors", in *ICRA*, 2010, pp. 1997–2004.

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

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

[16] V. Boor, M.H. Overmars, and A.F. van der Stappen, "The gaussian sampling strategy for probabilistic roadmap planners", in *ICRA*, 1999, pp. 1018–1023.

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

[18] D. Hsu, Tingting Jiang, J. Reif, and Zheng Sun, "The bridge test for sampling narrow passages with probabilistic roadmap planners", in *ICRA*, 2003, vol. 3, pp. 4420 – 4426 vol.3.

[19] S Dalibard and J.P. Laumond, "Linear dimensionality reduction in random motion planning", *Int. Journal of Robotics Research*, 2011.

[20] Junghwan Lee, OSung Kwon, Liangjun Zhang, and Sungeui Yoon, "Sr-rrt: Selective retraction-based rrt planner", in *IEEE Int. Conf. on Robotics and Automation*, 2012, pp. 2543–2550.

[21] 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.

[22] O. Brock and L. Kavraki, "Decomposition-based motion planning: A framework for real-time motion planning in high-dimensional configuration spaces", in *ICRA*, 2001, vol. 2, pp. 1469–1474.

[23] Y. Yang and O. Brock, "Efficient motion planning based on disassembly", in *Robotics: science and systems I*, 2005, vol. 1, p. 97.

[24] Yuandong Yang and Oliver Brock, "Adapting the sampling distribution in prm planners based on an approximated medial axis", in *ICRA*, 2004, vol. 5, pp. 4405–4410.

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

[26] M. Rickert, O. Brock, and A. Knoll, "Balancing exploration and exploitation in motion planning", in *ICRA*, 2008, pp. 2812–2817.

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

[28] H. Choset and J. Burdick, "Sensor based planning: The hierarchical generalized voronoi graph", *Workshop on Algorithmic Foundations of Robotics*, 1996.

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

[30] 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.

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

[32] 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.

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