VLSH: Voronoi-based Locality Sensitive Hashing

Tieu Lin Loi*, Jae-Pil Heo*, Junghwan Lee*, and Sung-eui Yoon*

Abstract— **We present a fast, yet accurate k-nearest neighbor search algorithm for high-dimensional sampling-based motion planners. Our technique is built on top of Locality Sensitive Hashing (LSH), but is extended to support arbitrary distance metrics used for motion planning problems and adapt irregular distributions of samples generated in the configuration space. To enable such novel characteristics our method embeds samples generated in the configuration space into a simple l_2 norm space by using pivot points. We then implicitly define Voronoi regions and use local LSHs with varying quantization factors for those Voronoi regions. We have applied our method and other prior techniques to high-dimensional motion planning problems. Our method is able to show performance improvement by a factor of up to three times even with higher accuracy over prior, approximate nearest neighbor search techniques.**

I. Introduction

In the context of sampling-based motion planning, the problem of nearest neighbor search is a very important part of most motion planners. In the Probabilistic Roadmap Method (PRM) [1], nearest neighbor search is performed after generating a set of valid robot configurations in order to form a well-connected roadmap. In the context of Rapidly-exploring Random Tree (RRT) [2], nearest neighbor search is used to expand the tree, enabling the planner to quickly find a collision-free path between the start and goal configurations.

In this paper we focus on solving the problem of finding k-nearest neighbors given a dataset of samples generated for sampling-based motion planning. k-nearest neighbor search in the context of the PRM motion planner is a computationally expensive task, because given n valid samples in configuration space, we need to perform nearest neighbor search for all these samples along with n queries from the same dataset of configurations. A naive approach of a linear scan algorithm gives a $O(n^2)$ time complexity, which is impractical for a large number of samples. Furthermore it is well-known that for points residing in high-dimensional spaces, i.e. high degrees of freedom, the nearest neighbor search problem suffers from the curse of dimensionality [3], reducing many nearest neighbor search data structures (e.g., kd-trees) to perform worse than a simple linear scan algorithm.

Locality Sensitive Hashing (LSH) [4] has been proposed to efficiently provide approximate nearest neighbors for high-dimensional points given the binary Hamming space $\{0,1\}^d$. This technique uses a hashing function that projects high-dimensional points onto randomly generated vectors drawn from a specific distribution. This LSH technique is extended to different metrics including l_2, i.e. the Euclidean distance metric [5]. The LSH technique was recently applied to nearest neighbor search used in motion planners [6]. Although this technique showed meaningful improvements, we have found that it is rather limited in a few metrics and may not handle samples with irregular distributions well.

In this paper we propose a novel, Voronoi-based LSH (VLSH), that supports approximate nearest neighbor search for high-dimesional samples generated in the configuration space, C-space, with an arbitrary distance metric. Our method first embeds samples generated in the C-space into a simple Euclidean space by using pivot points. During this embedding process we cluster samples based implicitly on the concept of Voroni regions constructed by those pivot points. This process enables the use of a localized LSH for each Voronoi region. VLSH has the following characteristics:

- **High-dimensionality.** Our method supports approximate k-nearest neighbor search efficiently for sampling-based motion planning. The algorithm works especially well for high dimensional spaces by utilizing well-established LSHs.
- **Distance metrics.** Our method can support a diverse set of distance metrics used in motion planning, because of our embedding process.
- **Adaptivity.** VLSH can adapt to irregular distributions of samples in the C-space, since we use local LSHs for Voronoi regions with different quantization factors.

We have implemented our method and other prior approximate nearest neighbor search techniques, and applied them to connect a roadmap among samples generated by a PRM-based motion planner. Our method shows performance improvement by a factor of a range of 1.4 to 3.7 times over prior techniques. Furthermore our technique shows even higher accuracy over them. These results are mainly caused by both using localized LSHs adapted to different regions and handling different distance metrics.

II. Related Work

In this section we briefly discuss prior work on nearest neighbor search, embedding motion planning spaces to a normed space, and LSH techniques.

A. Nearest Neighbor Search (NNS)

Motion planning has been well studied and a good survey is available [7]. The most well-known sampling based motion planners are RRT [2] and PRM [1]. These planners heavily use NNS.

There are many techniques for solving the NNS problem. One way is to use spatial subdivision data structures like kd-trees [8]. kd-trees are mostly used for low-dimensional search

* Department of Computer Science, KAIST (Korea Advanced Institute of Science and Technology)

problems, because for high dimensions (e.g., larger than 10), the algorithm suffers from the *curse of dimensionality* [3], reducing the algorithm to running worse than a simple linear scan.

Some of prior techniques [9] focus on culling of sample points by using the triangle inequality [10]. In this approach a set of reference points are selected from the dataset, and distances to these reference points are calculated. At query time with a query q and search radius r, a point u is culled whenever $d(p_i, u) - d(p_i, q) > r$ given a distance function d and reference point p_i. The advantage of using the triangle inequality for culling is that only a distance function is needed along with the triangle inequality for the culling process. This is especially useful for samples that have complex representations (e.g. documents consisting of a set of words instead of real numbers). A disadvantage of using the triangle inequality for culling is that as the dimensionality of data increases, the culling efficiency reduces [9].

Another tree-based approach is *Geometric Near Access Tree* (GNAT) [11], which is used in well-known motion planning libraries such as OOPSMP [12] and OMPL [13]. A GNAT picks a set of split nodes (e.g., 10 split nodes) from the original dataset, each of which forms a tree branch. Each point is then associated with its closest split node. This process is recursively applied for all the points associated with a split node, until each split node has a smaller number of points than a threshold. At query time, the triangle inequality is used for culling away tree branches to accelerate NNS computation.

Recently GPU based acceleration techniques have been proposed to speed up the NNS computation [14] based on a brute-force NNS algorithm. Pan et. al. [15] also proposed a GPU-friendly bounding volume hierarchy based k-NNS algorithm.

B. Embedding to a Simple Euclidean Space

In the context of motion planning many different distance metrics [16] exist. Examples of complex distance metrics include computing the swept volume between robot configurations [17] and a distance between the centers of mass of two configurations. The goal of a distance metric is to determine configuration similarity through a distance. As a result, a nearest neighbor search algorithm with a bad distance metric may return far away configurations, leading to unnecessary collision checks and thus affecting the overall motion planning process.

In the field of computer science, embedding complex spaces into simpler normed spaces is a well-studied topic, and a good introduction is given in [18]. The main purpose of this process is to reduce the computational overhead of computing distance with a small error caused by the embedding process.

Plaku and Kavraki [19] showed that one can embed points with a complex motion planning distance metric to the simpler l_2 distance metric. This saves computation time for the NNS phase with a small distance error. The embedding is based on Bourgain's theorem [20], which states that any distance metric can be embedded into the l_2 distance metric with a distortion of $O(log(n))$ for a dataset of size n. The algorithm for this embedding process chooses a set of n pivot points from the dataset. All points in the dataset then compute their distance to the pivot points using the original motion planning distance metric. This forms a new vector space, and distance between points can be approximated with the l_2 distance metric. This embedding is used in the proposed technique in order to reduce the computational overhead of NNS.

C. Locality Sensitive Hashing (LSH)

LSH [4] is a fast hashing technique for NNS designed for high dimensional spaces. LSH relies on hashing functions that map nearby points to the same hash-bucket. In query time the query is hashed to a hash-bucket and data points located at the bucket are potential neighbors to the query. The technique was extended to the l_2 distance metric [5]. Also, Pan and Manocha [6] designed a GPU-based LSH technique and applied it for motion planning problems. However, their method is limited to support the Euclidean and a non-scaled non-Euclidean distance metric [16].

III. Preliminaries

In this section we briefly give backgrounds on basic concepts that our work is built on top of, followed by motivating our approach.

A. Background on Motion Planning

The problem of motion planning consists of finding a collision free path from a start configuration of a robot (or several robots), to a goal configuration. A robot can be represented by an n-dimensional vector: $(x_1, x_2, ..., x_n)$, called a configuration, where x_i represents positions, angles, etc. of joints of robots.

All possible n-dimensional vectors of the robot's configuration form a vector space called the C-space. All the collision-free configurations of a robot are within a subset of the C-space and called C_{free}. A sampling-based motion planner generates samples in the C-space and performs collision detection in order to find a valid configuration in C_{free}. Valid samples are then stored as a graph and a search is performed, in order to find a solution path. Examples of well-known sampling-based motion planners are RRT [2] and PRM [1].

B. Background on NNS

In the context of motion planning, NNS is performed with robot configurations. k-nearest neighbors (k-NNs) for a query configuration q are candidate configurations for determining whether a valid path exists between q and those k nearest neighbors.

Formally, given a sample set of configurations $X \subseteq C$ of the C-space, C, a distance metric d computing a distance between two configurations should satisfy the following

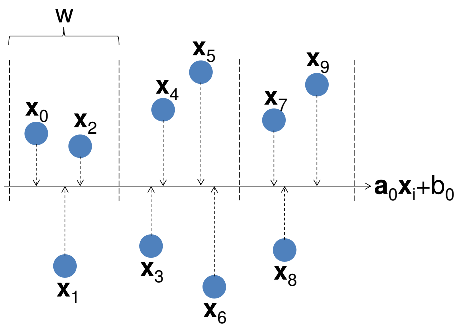

Fig. 1. This figure illustrates an overview of the LSH technique. Given a random projection vector a_0 and quantization factor w, points are projected onto the vector, and hashed to hash-buckets using a hash function $h(\cdot)$ depending on a_0 and w. More details about LSH technique are given in Sec. III-C

properties [21]:

$$d : X \times X \to \mathbb{R}, \tag{1}$$
$$d(x, y) \geq 0, \quad \forall x, y \in C, \tag{2}$$
$$d(x, y) = d(y, x), \quad \forall x, y \in C, \tag{3}$$
$$d(x, x) = 0, \quad \forall x \in C, \tag{4}$$
$$d(x, y) \leq d(x, z) + d(z, y), \quad \forall x, y, z \in C. \tag{5}$$

These properties staring from Eq. 2 are commonly called *non-negativity*, *symmetry*, *reflexivity*, and *triangle in-equality*, respectively.

k-NN search can be then defined in the following way: Given X and a query q, we want to find a set $S \subseteq X$ where $|S| = k$, s.t: $\forall s \in S$, $\forall u \in X \setminus S$, we have that $d(q, s) \leq d(q, u)$. In this paper we focus on its relaxed problem, *approximate k-nearest neighbor search* (Ak-NNS). Without loss of generality, we consider approximate neighbors, s', that an Ak-NNS algorithm returns. Given an approximation factor $\epsilon \geq 1$, s' satisfies that $d(q, s') \leq \epsilon \cdot d(q, s)$, given true nearest neighbors s.

C. Background on LSH

We provide a more in-depth explanation and discussion for LSH, since our technique is based on LSH [5]. The LSH technique relies on the fact that a set of random projections of data can approximate the l_2 distance metric well [22]. The main idea of LSH is to hash potentially close points to the same hash-bucket (Fig. 1). Each hash function used in LSH, $h_i(\mathbf{x})$, is as follows:

$$h_i(\mathbf{x}) = \lfloor \frac{\mathbf{a_i} \cdot \mathbf{x} + b_i}{w} \rfloor, i \in [1, M], \tag{6}$$

where $\mathbf{x}$ is the d-dimensional vector, $\mathbf{a_i}$ is a d-dimensional random vector, each of whose component is drawn from a normal distribution $\mathcal{N}(0, 1)$, b_i is a scalar value in the interval $[0, w]$, and w is the quantization factor that determines the quantization width for the projection. The floor function is used to map points in the same quantization segment together. Fig. 1 illustrates these projection and quantization procedures. We then use M different hash functions to create

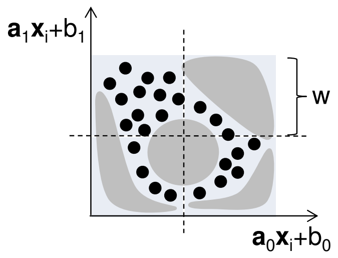

Fig. 2. This figure illustrates a 2D workspace with a point robot, where the dark grey objects are obstacles. It illustrates hashing of data points (black dots) with two different projections with the fixed quantization factor w. When we have irregular distributions for points, hash-buckets of the standard LSH technique can have a drastically varying number of points that lead to inferior performance.

a hash table, $g_j(\mathbf{x})$, as the following:

$$g_j(\mathbf{x}) =< h_1(\mathbf{x}), h_2(\mathbf{x}), ..., h_M(\mathbf{x}) > . \tag{7}$$

We use such M dimensional hash-vector, to increase a probability that close points are hashed to the same hash bucket. We can also use L independent hash tables to further increase the chance of finding the true neighbors:

$$H(\mathbf{x}) =< g_1(\mathbf{x}), g_2(\mathbf{x}), ..., g_L(\mathbf{x}) > . \tag{8}$$

At query time given a query q, $H(\mathbf{q})$ is computed, and data points located at corresponding buckets are retrieved as candidates for nearest neighbors to the query q.

D. Motivations

LSH techniques are very fast and accurate with a high number of hash tables. Nonetheless, there are a couple of issues that hinder efficient and effective approximate NNS for motion planning problems.

First, these algorithms are mainly designed for the l_2 distance metric and other generic metrics including l_p norms. As a result, a diverse set of distance metrics used in motion planning [16] is not supported well. A recent work [6] extended the standard LSH to support rotational degrees of freedoms (dof), but did not aim to support arbitrary distance metrics.

Second, LSH techniques are fundamentally *data-independent*, since they do not adapt their quantization factors or other parameters depending on data points to be hashed. As a result, these techniques may not be the optimal NNS techniques, since most data points generated in most motion planners tend to have irregular distribution (Fig. 2). Especially when we have narrow passages on environments under motion planning, it is highly likely to have drastic variation on the sampling density. As a result, a naive application of LSH techniques may require an excessive amount of memory with fine quantization factors or produce low accuracy for identifying nearest neighbors with a high quantization factor.

IV. VLSH

To support high-dimensional points efficiently and support arbitrary metrics, VLSH consists of two phases: a distance embedding phase and a localized LSH invocation phase.

Algorithm 1 VLSH algorithms.

```
Require: $\hat{X} \subset X(\subseteq C)$, $k$: the number of used pivots, $X'$: the new vector space for embedded points.
 1: function EMBED($\hat{X}$)
 2:     $p_1 \leftarrow$ randomly choose an element in $\hat{X}$
 3:     for $i \leftarrow 2$ to $k$ do
 4:         $p_i \leftarrow$ maximize $\min_{j=1}^{i-1} d_{MP}(p_i, p_j)$
 5:     end for
 6:     for $u \in X$ do
 7:         $u' \leftarrow (d_{MP}(u,p_1), d_{MP}(u,p_2), ..., d_{MP}(u,p_k))$
 8:     end for
 9:     return $(X', \{p_1, p_2, ..., p_k\})$
10: end function
11:
12: function LOCAL_LSH($X$, $\hat{X}$, $k$, $X'$, $\alpha$, $\beta$)
13:     $(X', \{p_1, p_2, ..., p_k\}) \leftarrow Embed(\hat{X})$
14:     for $i \leftarrow 1$ to $k$ do
15:         $vr'(p_i) \leftarrow \{u' | \forall u' \in X' s.t.\ argmin_j(u'_j) = i\}$
16:     end for
17:     for $i \leftarrow 1$ to $k$ do
18:         $evr'(p_i) = vr(p_i) \cup top\_subset(vr'_2(p_i), \alpha)$
19:         $\sigma_i \leftarrow$ std. dev. of dist. in $evr'(p_i)$ to $p_i$
20:     end for
21:     for $i \leftarrow 1$ to $k$ do
22:         $w_i \leftarrow \sigma_i \cdot \beta$
23:         Construct $LSH_i(evr'(p_i), w_i)$
24:     end for
25:     return $LSH_1$, $LSH_2$, ... ,$LSH_k$
26: end function
27:
28: function QUERY_VLSH($q$)
29:     $q' \leftarrow Embed(q)$
30:     $i \leftarrow argmin_j(q'_j)$
31:     return results of $LSH_i(q')$
32: end function
```

A. Embedding

The purpose of this phase is to transform coordinates of data points from a complex motion planning space into a simpler l_2 distance metric space. The main reasons why we conduct such transformation is that the computation overhead of NNS algorithms can be reduced [19], and we can use a LSH optimized for l_2 distance metric.

The embedding process is performed by first choosing a set of pivot points, $P \subset X$, from the valid sample set X. A criterion for choosing pivot points is that they should serve as a good approximation of samples of X. Considering all the samples of X can take a prohibitive computing cost, and thus we select pivot points from a subset, $\hat{X}$, an approximation of the original sample set X. To construct $\hat{X}$, we randomly select samples from X. In practice we set $|\hat{X}|$ to be 10% of $|X|$.

Let $p_1, p_2, ..., p_k$ denote the chosen pivots. The first pivot p_1 is chosen from $\hat{X}$ uniformly at random, and $p_i, 2 \leq i \leq k$, are chosen from $\hat{X}$ in order to maximize $\min_{j=1}^{i-1} d_{MP}(p_i, p_j)$, where $d_{MP}(\cdot)$ is a motion planning distance metric. This pivot selection strategy avoids choosing a pivot point that is close to already chosen pivots, in order to increase the information gain of having additional pivots and thus increase the accuracy of NNS for embedded points constructed after the embedding process.

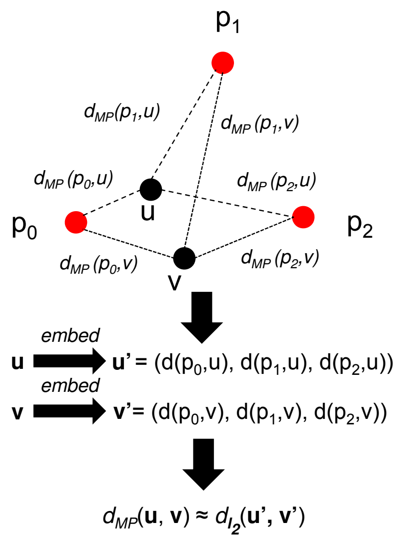

Fig. 3. This figure illustrates our embedding process for two sample points, u and v, given three pivots, p_1, p_2, and p_3.

Given a point $u \in C$, its embedded point $u' \in \mathbb{R}^k$ with the chosen pivots P is defined as follows:

$$u' = (d_{MP}(u,p_1), d_{MP}(u,p_2), ..., d_{MP}(u,p_k)). \quad (9)$$

This embedding procedure is performed for all the points generated by a motion planner. Let us denote this new vector space of embedded points $X' \subset \mathbb{R}^k$. We then approximate $d_{MP}(\cdot)$ in X as l_2 distance between embedded points located in this new vector space X'. Fig. 3 illustrates our embedding process with a simple example. A pseudocode of our embedding process is shown as *Embed* $(\cdot)$ in Algorithm 1.

Note that our embedding process can lower down the accuracy of NNS, since we perform our NNS with the LSH coupled with l_2 distance metric, instead of the original motion planning metric d_{MP}. Nonetheless, the distortion factor is not significant according to the Bourgain's theorem [20], and we have observed that the accuracy loss of our embedding process is very minor in practice, while the embedding process greatly accelerates the runtime computation. Furthermore we use a localized LSH for each Voronoi region, which is discussed in the next section, in order to improve the accuracy of our method.

B. Invoking a Localized LSH

Given a query point q we would like to identify a set of nearest neighbor candidates for the query point q. In this section we explain how to define such set of points based on our localized LSHs.

Given pivot points $\{p_1, p_2, ..., p_k\}$ for embedding, we associate sample points of X' based on the concept of

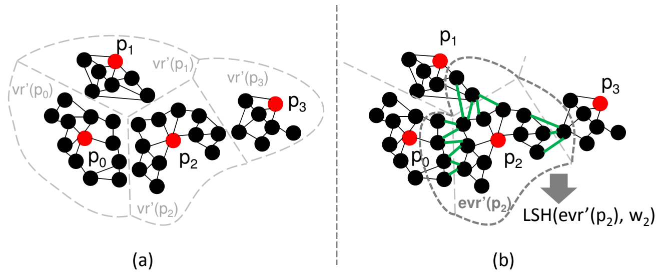

Fig. 4. The left figure shows disconnected roadmaps when we use only points contained in Voronoi regions $vr'(p_i)$, while the right figure shows a well-connected roadmap by using the extended Voronoi regions $evr'(p_i)$. Green edges in the right figure are additionally connected by using $evr'(p_2)$ of a pivot p_2.

Voronoi regions. Specifically we define $vr'(p_i)$ to contain embedded samples located within the Voronoi region of a pivot point p_i in the embedded space, as the following:

$$vr'(p_i) = \{u' | \forall u' \in X' s.t.\, argmin_j(u'_j) = i\}, \tag{10}$$

where u'_j denotes the j-th component of the vector u', $argmin_j(u'_j)$ denotes the u''s smallest vector component. In another view, we associate all the points in X' with their closest pivot points. As a result, given an embedded query point q' from the original query point q, we can identify its closest pivot point, say p_i and use the points contained in its Voronoi region, $vr'(p_i)$, in the embedded space, as nearest neighbor candidates for q. To realize this process, we construct a localized LSH for the embedded points in $vr'(p_i)$ of each pivot point p_i.

We could simply build a localized LSH for only points contained in each $vr'(p_i)$. This simple approach, however, has a serious problem: when two points are located very closely in the original space, but their closest pivot points are different, those two points can belong to different Voronoi regions and thus cannot be identified as nearest neighbor candidates between them. This results in disconnected roadmaps for PRM methods, as shown in Fig. 4-a).

To address this problem, we construct an extended Voronoi region, $evr'(p_i)$, for each pivot point such that it can contain even nearby points from other neighboring Voronoi regions along the boundary of the Voronoi region of p_i. To realize this idea, we also compute points whose second-closest pivot is p_i and store them in a new set, called $vr'_2(p_i)$. We can consider all the points in $vr'_2(p_i)$ in addition to $vr'(p_i)$, but we found that a subset of $vr'_2(p_i)$ is sufficient for NNS with a reasonable amount of the memory overhead and high accuracy. As a result, we construct the extended Voronoi region $evr'(p_i)$ as the following:

$$evr'(p_i) = vr'(p_i) \cup top\_subset((vr'_2(p_i), \alpha), \tag{11}$$

where $vr'_2(p_i)$ contains embedded points whose second smallest component is from p_i and $top\_subset(S, \alpha)$ returns points from the set S within α percentile in terms of its distance to p_i. In practice 0.6 to 0.9 for α gives a good trade-off in terms of high accuracy and low memory overhead. Fig. 4-b) shows an illustration of extended Voronoi regions.

Once we construct the extended Voronoi region $evr'(p_i)$ for a pivot point p_i, we build a localized LSH with points contained in each extended Voronoi region. Our VLSH is then defined as the following:

$$VLSH = LSH^k_{i=1}(evr'(p_i), w_i),$$

where w_i is a local quantization factor for the localized LSH specialized to the i-th extended Voronoi region $evr'(p_i)$. Each w_i can be computed locally based on points contained in $evr'(p_i)$. Specifically, it is computed based on the standard deviation, σ_i, of distances between points of $evr'(p_i)$ and the pivot point p_i in the embedded space. We then set w_i to be $\sigma_i \beta$, where β is a user-specified constant and is used to fine tune the accuracy and memory overhead of the used LSH. The pseudocode of constructing localized LSHs is shown as *Local_LSH* $(\cdot)$ in Algorithm 1.

C. Query-time Algorithm

Given a query point q, the corresponding embedded point q' is computed, and the closest pivot p_i is determined by scanning vector components of the embedded point q'. We then use the localized LSH associated with $evr'(p_i)$, and return candidate nearest neighbors located in hash-buckets corresponding to q'. Its pseudocode, called *Query_VLSH (q)*, is shown in Algorithm 1.

V. Test Configurations

In our experiments we compare VLSH against LSH and GNAT. The implementation of LSH is based on details discussed in the paper by Datar and Indyk [5]. All algorithms are written in C++. The implementation of GNAT is from OOPSMP, the open-source motion planning library [12]. We conduct all the experiments in a Windows XP 64-bit machine with an Intel i7 3.3 GHz CPU.

A. Benchmarks

For our experiments we use three benchmark scenes: bug trap and wiper scenes (Fig. 5). The wiper benchmark consists of a wiper as a robot with a windscreen as an obstacle. The dimension of the configuration space of this benchmark is six including 3D positional and rotation parts. It is also relevant to test the behavior of the algorithms for high-dimensional C-spaces, because LSH and VLSH are designed for high-dimensional search problems. We test different methods with the bug trap benchmark with multiple robots to see their behaviors in high-dimensional points. Plaku and Kavraki [19] showed that in the context of motion planning, that the critical dimension of C-space is between 15 and 30. So testing our algorithm in this interval and on higher dimensions is relevant.

We therefore have two different versions of the bug trap benchmark in those interval for the dimensionality: one using four robots to create 24 dimensions, another one using six robots to create 36 dimensions. For this benchmark the bug trap is the obstacle, and sticks serve as robots.

A distance metric used for both benchmarks are the non-scaled Euclidean [16], which computes the distance for each

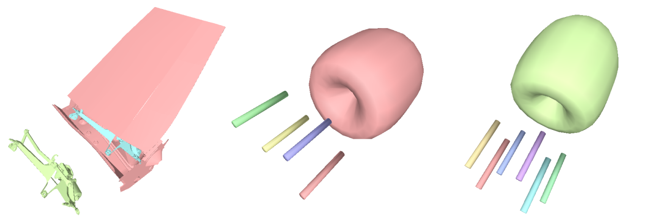

Fig. 5. The wiper benchmark (the leftmost) is of six dimensions, and the bug trap benchmarks have higher dimensions: 24 with four sticks (the middle) and 36 with six sticks (the rightmost).

robot's translational parts and rotational parts separately and add them together without any scaling.

We use a PRM planner to generate samples and find collision-free paths. To test the accuracy and runtime performance of different NNS algorithms, we query all the generated samples from the PRM planner and find their k nearest neighbors.

B. Parameter Settings

In all experiments we use the same values for the parameters that LSH and our VLSH share. Specifically, these parameters include the number of hash-tables L (10 for the wiper and 3 for the bug trap) and the hash-vector size M (10 for the wiper and 15 for the bug trap); we vary them for different benchmarks to see behaviors of tested methods in a wider setting.

For the quantization factor w, LSH uses 4.0 for both the wiper benchmark and the bug trap benchmark. This parameter value is the giving the best performance among a set of tested parameters. In VLSH we use two specialized parameters α and β (Sec. IV-B), and we set $\alpha = 0.8$, $\beta = 14.0$ for the wiper benchmark, $\alpha = 0.6$, $\beta = 20.0$ for the four robots bug trap benchmark and $\alpha = 0.7$, $\beta = 22.0$ for the six robot bug trap. The chosen parameters show the best performance among a set of tested parameters. We use 10 pivot points, which work well in the tested benchmarks.

GNAT has its own parameters including the min degree, main degree, and max degree for its tree related to the number of branches, and a maximum number of points in a leaf node. Parameter values used are 2, 10, 10, and 20, respectively, which are default parameter values in OOPSMP. We also found that these parameter values work well for our benchmarks.

For all the tested NNS methods, we set to find 15 nearest neighbors, i.e. $k = 15$, which is the default value used in PRM of OOPSMP. In addition, GNAT and LSH use the embedding process explained in Sec. IV-A. For GNAT this embedding reduces its computational overhead, but for LSH this embedding is mandatory in order to make it support non-Euclidean distance metrics. It also reduces its computational cost. GNAT and LSH with the embedding process are denoted as GNAT (Em) and LSH (Em), respectively.

C. Quality and Performance Evaluation

In order to evaluate the quality of search results of different methods, we compute the fractional distance error (fde), which is used as an evaluation protocol in prior work [19]. fde captures a difference between the ground truth's summed distance from the query point and summed distance of results reported by approximate nearest neighbor methods. Let $d_{sum}^{NN}(q)$ denote the sum of distances of the true k nearest neighbors for a query q, and $d_{sum}^{ANN}(q)$ denotes the corresponding distance, but with a ANN algorithm. fde is then defined as the following:

$$fde(q) = 1 - \frac{d_{sum}^{NN}(q)}{d_{sum}^{ANN}(q)}. \quad (12)$$

fde is in a range of $[0, 1]$, and has a lower value, as results of approximate techniques close to those of the ground truth results; we can easily compute the ground truth results by exhaustively searching all the sample points with the original motion planning metric. We report the average fde value of all the tested queries.

In addition to the search quality evaluation we also measure the runtime cost. Since the construction time of data structures is negligible, we add the construction time of different methods with query time spent on querying all the sample points in the dataset.

VI. Results and Analysis

Fig. 6 shows search quality and runtime cost of different methods in our three tested benchmarks. Overall our method shows the best search quality while requiring the minimum runtime cost. For the wiper benchmark VLSH is up to 2.5 times faster than GNAT with embedding, while showing a smaller distance error. Furthermore, VLSH is up to 1.8 times faster running time than LSH with embedding, while showing a smaller distance error. It is clear that VLSH shows better results than LSH (Em) and GNAT (Em) even for this low-dimensional benchmark.

For the bug trap benchmark with four robots, we see that VLSH against GNAT (Em) gives up to 3.3 times faster running time with a smaller distance error. For VLSH against LSH (Em) there is up to 1.7 times faster running time with a smaller distance error. We can see a similar trend among the tested methods even with six robots; VLSH shows up to 4.0 times and 1.4 times faster than GNAT (Em) and LSH (Em) respectively, while showing smaller distance errors.

Timing breakdown between VLSH and LSH (Em). Table I shows the timing breakdown for the wiper benchmarks between VLSH and LSH (Em). The other benchmarks are omitted due to limited space, but their trends are similar to the one reported here. We see that for all the tested benchmark, VLSH has a higher overhead in the hashing phase, where it has to compute Voronoi regions. The overhead is around two times, but this overhead gives us large improvements on the query time, which is the bottleneck of nearest neighbor search algorithms.

Comparison with GNAT without embedding. We have also tested GNAT without any embedding on our benchmarks and compared it to our method. GNAT without any embedding returns no distance error for all benchmarks, but gives inferior running times. For the wiper benchmark VLSH runs

	Wiper							
Dataset size	10000		15000		20000		25000	
Phase	LSH (Em)	VLSH	LSH (Em)	VLSH	LSH (Em)	VLSH	LSH (Em)	VLSH
Embedding	0.16		0.24		0.32		0.40	
Hashing	0.09	0.19	0.12	0.30	0.16	0.38	0.20	0.48
Query	1.15	0.71	2.72	1.50	3.76	1.93	6.38	3.10

TABLE I

THIS TABLE SHOWS THE TIMING BREAKDOWN OF THE DIFFERENT PHASES FOR VLSH AND LSH WITH EMBEDDING.

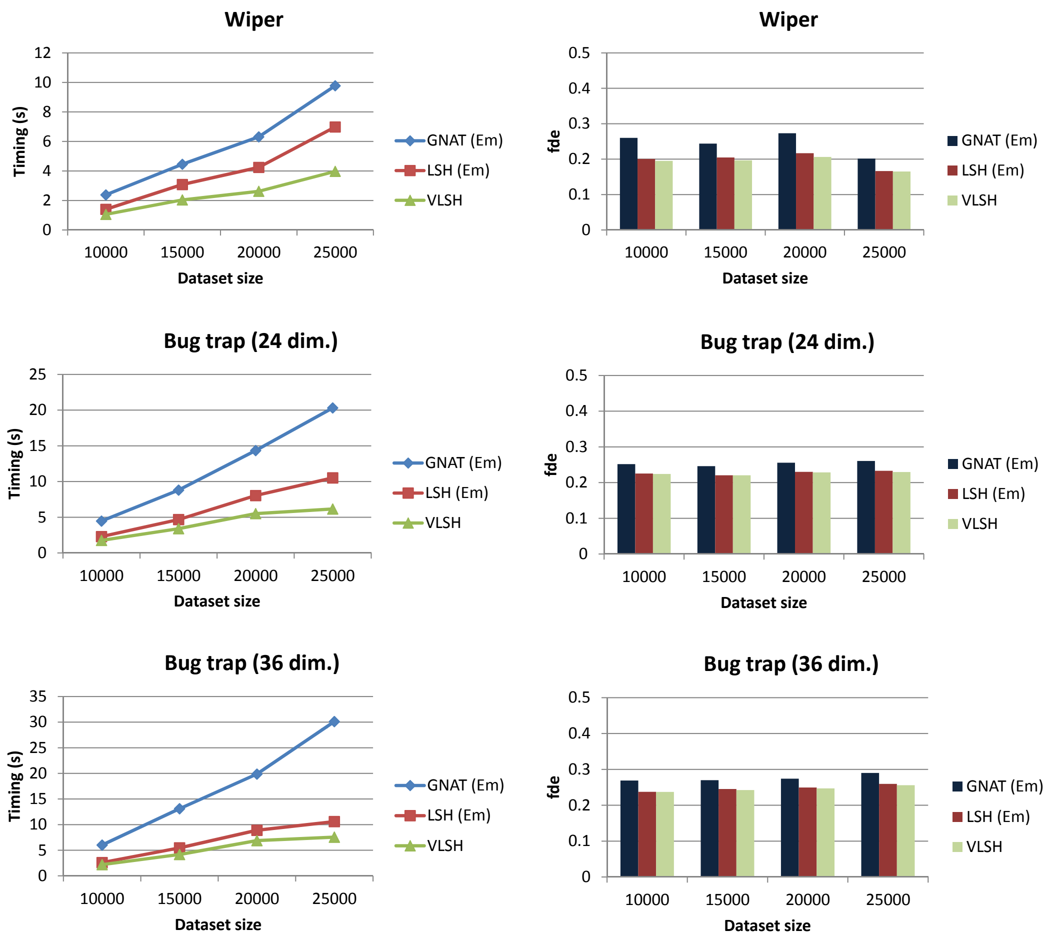

Fig. 6. These graphs show the runtime cost and accuracy of different techniques in the three tested benchmarks.

up to 9.7 times faster over GNT without any embedding. For the high dimensional bug trap benchmark the difference is even larger. For the four robot bug trap benchmark VLSH is up to 137 times faster than GNAT and up to 208 times faster for the six robot bug trap benchmark. This clearly shows that the embedding process can greatly reduce the computational overhead of NNS in high dimensional spaces.

Comparison with a GPU-friendly LSH [6]. Pan et al. [6] proposed a GPU-friendly LSH technique. While its GPU acceleration techniques can be applied to our method, this GPU technique also proposed a way to handle rotational degrees-of-freedoms (*dof*) that are represented in Euler angles. In addition, this method supports non-Euclidean metrics, but does not handle cases when distances of translational parts and rotational parts are scaled. For example, in a 3D environment where a robot is mostly restricted to moving in a 2D plane, scaling the direction orthogonal to the plane should be better for exploring its C-space. Our method supports those scaled Euclidean distance and other types of more complicated metrics listed by Amato et al. [16], while handling quaternions.

Memory overhead of VLSH. Because VLSH duplicate some points from nearby Voronoi regions, there is some memory overhead compared to LSH. In terms of memory usage, the only difference between VLSH and LSH is the hashing phase. The memory usage in VLSH is affected by the parameter α and in our experiment we observed that our method uses on average $1+\alpha$ time more memory than LSH.

In our experiments α ranges from 0.6 to 0.8 giving 1.6 to 1.8 times more memory overhead. Nonetheless, the overall memory requirement is not a serious constraint, since all the data structures of our method for the tested benchmarks take tens of MB.

VII. Conclusion

In this paper we have proposed novel, VLSH technique for high-dimensional motion planning problems. Our method consists of two steps. We first embed high-dimensional points into low-dimensional spaces based on pivots, and perform l_2 distance metric in the embedding space. We then localize a LSH that contains nearest neighbors given a query point. Based on these two steps, we can support high-dimensional spaces efficiently even with arbitrary distance metrics. Furthermore we can achieve high search accuracy over prior techniques.

Interesting future research directions lie ahead. Like many LSH techniques, our VLSH method has different parameters including the number of hash-tables L, hash-vector size M, and quantization factors w_i, plus our own parameters α and β. This makes it harder to configure VLSH to run optimally for a particular dataset. We would like to design a technique that identifies optimal values for these parameters, as a prior work took a similar approach for the common LSH [23]. Motion planners such as RRT incrementally build their internal data distribution in order to quickly find a valid path. This means that a NNS algorithm has to adapt its data structures for dynamically changing datasets. Currently our work is tested only for static datasets generated by PRM techniques. Our technique can be applied to dynamic datasets, but it may not show the optimal performance when the density of underlying point datasets changes drastically. We would like to extend and optimize our work better for such dynamic models. Finally, we would like to parallelize our method especially with a GPU for real-time motion planning. Since our technique is based on the LSH approach, we believe that our technique can be easily parallelized and show high performance, as shown in a prior work [6].

ACKNOWLEDGMENT

We would like to thank SGLab members and anonymous reviewers for constructive comments. This work was supported in part by MCST/KOCCA/CT/R&D 2011, DAPA/ADD (UD110006MD), MEST/NRF/WCU (R31-2010-000-30007-0), MSRA, MEST/NRF (2012-0009228), and IT R&D program of MOTIE/KEIT [10044970].

References

[1] 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.

[2] J. Kuffner and S. LaValle, “Rrt-connect: An efficient approach to single-query path planning,” in *Proceedings of IEEE Int. Conf. on Robotics and Automation*, 2000, pp. 995–1001.

[3] R. Weber, H.-J. Schek, and S. Blott, “A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces,” in *Int. Conf. on Very Large Data Bases*, 1998, pp. 194–205.

[4] P. Indyk and R. Motwani, “Approximate nearest neighbors: toward removing the curse of dimensionality,” in *STOC*, 1998.

[5] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, “Locality-sensitive hashing scheme based on p-stable distributions,” in *SoCG*, 2004.

[6] J. Pan, C. Lauterbach, and D. Manocha, “Efficient nearest-neighbor computation for gpu-based motion planning,” in *IROS*, 2010, pp. 2243–2248.

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

[8] A. Yershova and S. M. LaValle, “Improving motion-planning algorithms by efficient nearest-neighbor searching,” *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 151–157, 2007.

[9] E. Chávez, G. Navarro, R. A. Baeza-Yates, and J. L. Marroquín, “Searching in metric spaces,” *ACM Comput. Surv.*, vol. 33, no. 3, pp. 273–321, 2001.

[10] E. V. Ruiz, “An algorithm for finding nearest neighbours in (approximately) constant average time,” *Pattern Recogn. Lett.*, vol. 4, no. 3, pp. 145–157, 1986.

[11] S. Brin, “Near neighbor search in large metric spaces,” in *In Proceedings of the 21th International Conference on Very Large Data Bases*, 1995.

[12] E. Plaku, K. E. Bekris, and L. E. Kavraki, “Oops for motion planning: An online open-source programming system,” in *Proceedings of IEEE Int. Conf. on Robotics and Automation*, 2007, pp. 3711–3716.

[13] I. A. Şucan, M. Moll, and L. E. Kavraki, “The Open Motion Planning Library,” *IEEE Robotics & Automation Magazine*, 2012. [Online]. Available: http://ompl.kavrakilab.org

[14] V. Garcia, E. Debreuve, and M. Barlaud, “Fast k nearest neighbor search using gpu,” in *IEEE Conf. on Computer Vision and Pattern Recognition Workshops*, 2008, pp. 1–6.

[15] J. Pan, C. Lauterbach, and D. Manocha, “g-planner: Real-time motion planning and global navigation using gpus,” in *AAAI*, 2010.

[16] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo, “Choosing good distance metrics and local planners for probabilistic roadmap methods.” *IEEE Transactions on Robotics*, vol. 16, no. 4, pp. 442–447, 2000.

[17] P. Xavier, “Fast swept-volume distance for robust collision detection,” in *Proceedings of IEEE Int. Conf. on Robotics and Automation*, vol. 2, 1997, pp. 1162 –1169.

[18] P. Indyk and J. Matousek, “Low-distortion embeddings of finite metric spaces,” in *in Handbook of Discrete and Computational Geometry*, 2004, pp. 177–196.

[19] E. Plaku and L. E. Kavraki, “Quantitative analysis of nearest-neighbors search in high-dimensional sampling-based motion planning,” in *in Workshop on Algo. Found. of Robot*, 2006.

[20] J. Bourgain, “On lipschitz embeddings of finite metric spaces in hilbert space,” *Israel Journal of Mathematics*, vol. 52, no. 1, pp. 46–52, Mar. 1985.

[21] D. Burago, I. Burago, and S. Ivanov, “A course in metric geometry,” *American Math. Soc.*, 2001.

[22] W. Johnson and J. Lindenstrauss, “Extensions of Lipschitz mappings into a Hilbert space,” in *Conference in modern analysis and probability*, ser. Contemporary Mathematics. American Mathematical Society, 1984, vol. 26, pp. 189–206.

[23] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li, “Modeling lsh for performance tuning,” in *Proceedings of the 17th ACM conference on Information and knowledge management*, ser. CIKM ’08. ACM, 2008, pp. 669–678.