Free-configuration Biased Sampling for Motion Planning

Joshua Bialkowski, Michael Otte, and Emilio Frazzoli

***Abstract*— In sampling-based motion planning algorithms the initial step at every iteration is to generate a new sample from the obstacle-free portion of the configuration space. This is usually accomplished via rejection sampling, i.e., repeatedly drawing points from the entire space until an obstacle-free point is found. This strategy is rarely questioned because the extra work associated with sampling (and then rejecting) useless points contributes at most a constant factor to the planning algorithm's asymptotic runtime complexity. However, this constant factor can be quite large in practice. We propose an alternative approach that enables sampling from a distribution that provably converges to a uniform distribution over *only* the obstacle-free space. Our method works by storing empirically observed estimates of obstacle-free space in a point-proximity data structure, and then using this information to generate future samples. Both theoretical and experimental results validate our approach.**

I. Introduction

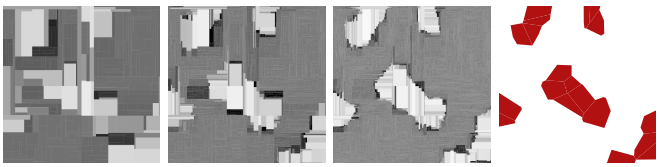
Sampling-based motion planning algorithms have proven to be effective tools for finding solutions to otherwise intractable motion planning problems (e.g., [11], [15], [19] to list a few). While a complete algorithm requires an explicit representation of the configuration free space, sampling-based algorithms merely draw samples from that space and then opportunistically build a graph of collision-free paths between them. By trading (deterministic) completeness in favor of *probabilistic completeness*, sampling-based algorithms are able to solve problems using a fraction of the computational resources required by complete algorithms.

Sampling-based algorithms map a sequence of sampled points to a sequence of graphs, and often rely on a common high-level structure including, e.g., the Probabilistic Roadmap (PRM) [11], the Rapidly Exploring Random Tree (RRT) [15], [14], their asymptotically optimal versions RRT* and PRM* [10], and many others derived from them. Let $G = (V, E)$ denote the output of the algorithm, where G is a graph defined by its vertex set V and edge set E.

At each iteration, a new sample x_{sample} is generated from the obstacle-free space. A set of candidate nodes $V_{\text{near}} \subset V$ is selected for possible connection with x_{sample}. That is, for each $x \in V_{\text{near}}$, if a local planner determines a collision-free path exists from x to x_{sample}, then x_{sample} is added to V and $[x, x_{\text{sample}}]$ is added to E.

Sampling-based algorithms avoid the construction of an explicit free-space map and therefore generally rely on rejection sampling for generating new configurations (i.e., x_{sample}) from the free space [13]. In rejection sampling a point is first drawn from the configuration space[1] and then statically collision checked. If the point is in collision, then it is discarded and a new point is drawn. This continues until a collision-free point is found.

Sampling Distribution Induced by Our Algorithm in 2D



(a) (b) (c) (d)

Fig. 1: The induced sampling distribution of an augmented kd-tree after 10^4, 10^5, and 10^6 samples are shown in (a), (b), and (c), respectively. White-black represent low-high sampling probability density. The actual obstacle configuration appears in (d), obstacles are red.

Once a collision-free sample is found, it is immediately submitted to a proximity query in order to discover candidates for graph connection. There are many data structures which are used for these proximity queries, most notably spatial indices like bounding volume hierarchies (e.g., kd-trees and quad-trees), and tessellations (e.g., triangulations and Voronoi diagrams) [21].

Our key insight is that many spatial indices passively encode information about the location of obstacles within the configuration space; by exposing and exploiting this information we can significantly reduce the number of rejected (i.e. *unnecessary*) future samples. In particular, by augmenting the index to record empirical collision information, it is possible to generate samples from a distribution that converges to uniform sampling over the *free space*, while *simultaneously* finding connection candidates for the new sample. In exchange for reducing the number of rejected samples, we accept a constant-factor increase in the space/memory complexity of our spatial index, and sacrifice independence of consecutive samples.

II. Previous Work

In the context of sampling-based motion planing, a variety of techniques have been proposed to increase the chances of sampling from within narrow corridors [9], [7], [20], goal regions [6], or other particular regions of the configuration space [1], [23], [3]. Surveys of these methods can be found in [4], [16], and [22]. In contrast to our work, none are concerned with sampling uniformly from the free space.

Our idea can be described as an *adaptive sampling algorithm* as defined in [16] (i.e., a sampling algorithm that changes its sampling distribution as it runs, in response to new information that is gathered and/or other stimuli). [5]

[1]Often either from a uniform distribution or deterministically (for example, using a Halton sequence or a space-filling curve).

presents a canonical algorithm for adaptive sampling from a univariate distribution that learns the envelope and squeezing functions. The main differences between [5] and our work is that we consider a multivariate distribution with an arbitrarily large number of variables (i.e., dimensions), and we focus on the problem of uniform sampling from an unknown (but discoverable) subspace of interest.

Related work in adaptive sampling for sampling-based motion planning follows. Hsu *et al.* [8] generate new samples such that the chance of sampling a particular point is inversely proportional to the local density of previous samples near that point. A similar idea is pursued in [22], where samples are drawn less densely in open regions of the configuration space and more densely in cluttered regions. Phillips *et al.* [17] weight the probability of sampling from a particular region based on the properties of the nearest node as follows: the weight is inversely proportional to a function of the nearby node's A^* cost and its number of graph-neighbors, and proportional to a function of its order in the sample sequence. A modification in [18] additionally assign to new samples a weight that is inversely proportional to a function of the local density of previous samples. The main difference between all of these methods and our idea is that they focus on generating samples from sparsely sampled portions of the configuration space—a practice that is actually expected to *decrease* the probability a new point is sampled from the free space[2]. In contrast, we are interested in producing the opposite behavior—as more and more samples are generated, our algorithm has an *increasing* probability of sampling from the free space.

Finally, [12] is arguably related to our idea because it records statistics of total successful vs. unsuccessful samples (although it uses this data to test a stopping criterion that results in probably more than a user defined proportion of the free-space being explored). While we propose recording similar statistics in the spatial index structure, we use this data to guide future sampling instead.

III. Algorithm

In this section we illustrate the application of our method by describing the sampling and search algorithm for an augmented kd-tree. The algorithm presented in this section may be used as a guide to implementing the method for other hierarchical spatial data structures where nodes at any depth form a tessellation of the indexed space.

Our method relies on storing extra data in each node of a kd-tree. A kd-tree is a special type of binary search tree that can efficiently determine the nearest neighbor(s) of a query point x_q within a previously defined finite set of points $\mathcal{X} \subset \mathbb{R}^d$ [2]. Each node in the kd-tree is a `Node` data structure, the fields of which are summarized in table I.

Each node v in a kd-tree defines an axis-aligned hyper-rectangle $H(v) \subset \mathbb{R}^d$. An interior node v is assigned a point $x \in \mathcal{X} \cap H(v)$ and an index $j \in \{1 \dots d\}$. Its two children are the hyper-rectangles found by splitting $H(v)$ with a hyperplane passing through x and orthogonal to the j-th axis. Leaf nodes are the same as interior nodes except that they are not assigned a point and have no children (yet). Finally, for any $H \subset \mathbb{R}^d$, `Measure` (v) returns the measure of the set $H(v)$, and `SampleUniform` (H) returns a point drawn from a uniform distribution over H.

TABLE I: The `Node` data structure

field	type	description
x	vector $\in \mathbb{R}^d$	point associated with this node
j	integer $\in \{1..d\}$	index of the split plane
$c[2]$	`Node` array	references to the children of the node (two in the case of a kd-tree), or a *null* reference $\emptyset$ if this is a leaf node
P	`Node`	reference to the parent node
T	float $\in \mathbb{R}$	weighted number of samples generated from H
F	float $\in \mathbb{R}$	weighted number of collision free samples generated from H
M	float $\in \mathbb{R}$	estimated measure of free space in H

Algorithm 1: `GenerateSample` (H, v)

```
1  if v.c[0] = v.c[1] = ∅ then
2      x ← SampleUniform(H) ;
3      v.T ← v.T + 1 ;
4      r = Collision-free(x) ;
5      if r then
6          v.x ← x ;
7          v.F ← v.F + 1 ;
8          (v.c[0], v.c[1]) ← Split(v, x) ;
9          for i = {0, 1} do
10             v.c[i].P ← v ;
11             v.c[i].j ← (v.j + 1) mod d ;
12             w ← Measure(v.c[i]) /Measure(v) ;
13             v.c[i].T ← w · v.T ;
14             v.c[i].F ← w · v.F ;
15             v.c[i].M = (v.c[i].F / v.c[i].T) Measure(v.c[i]) ;
16 else
17     u ← SampleUniform([0, v.M]) ;
18     if u ≤ v.c[0].M then
19         (x, r) ← GenerateSample(v.c[0]) ;
20     else
21         (x, r) ← GenerateSample(v.c[1]) ;
22     v.M = v.c[0].M + v.c[1].M ;
23 return (x, r)
```

Our algorithm appears in Algorithm 1 (this recursive procedure may be replaced with two loops and a stack if desired). Note that we store three additional fields in each node of our our augmented kd-tree: T, F, and M. Both T and F are only used by leaf nodes. T is the total number of samples taken from H, and F is the number of those samples that are collision free. When a leaf node generates a new sample (and thus creates its children), each child inherits a weighted version of T and F from the parent. Both values are weighted by the relative measure of space contained in the child vs. the parent, and both account for the successful sample before weighting (lines 3, 7-14). M is our estimate

[2]The rejection of obstacle space points upon collision detection results in free-space regions becoming increasingly populated with old samples (e.g., relative to obstacle space), and hence the sparsely populated obstacle space is sampled increasingly frequently.

of the measure of free space contained in H. For leaf nodes, $M = \frac{F}{T}\texttt{Measure}(H)$, line 14. For non-leaf nodes, M is the cumulative sum of the values of M contained in the node's children $M = c[0].M + c[1].M$, line 21.

The procedure starts at the root of the tree, and then recursively picks a child using a weighted coin flip (lines 16-20). In particular, the chance of recursing on the i-th child is calculated as $v.c[i].M/v.M$ (lines 16-17). Once a leaf node is reached, the sample point is drawn from a uniform distribution over that leaf's hyper-rectangle (line 2).

The sampled point and the result of the collision check are propagated back up the tree so that the statistics of each interior node in the recursion can be updated (lines 21-22). As more samples are generated by descendants of a particular node, the estimate of that node's free space improves. Thus, future sampling more accurately reflect the true distribution of free space. Formal proofs are presented in Section IV.

Lastly, we recall the algorithm for performing nearest neighbor queries[3] in a kd-tree: For a query point x_q, we perform a depth first search to find the leaf node v_0 containing x_q. During this search we push each touched node into a stack S. We initialize the nearest neighbor x_{NN} with the point corresponding to v_0. Then, while the stack is not empty, we pop the top node v off the stack and do the following:

- for the point x associated with node v, if $d(x, x_q) < d(x_{NN}, x_q)$ we replace x_{NN} with x.
- if the hypercube of any unsearched children contains a point x such that $d(x, x_q) < d(x_{NN}, x_q)$, we push that child onto the stack.

Thus, we may *simultaneously* generate a new sample and perform nearest neighbor queries by replacing the initial depth first search with Algorithm 1, at the same runtime complexity of doing a nearest-neighbor search.

IV. Analysis

A. Convergence to a uniform distribution over free space

We now prove that the sampling distribution induced by our algorithm converges to a uniform distribution over the free space.

Let S_{O}, S_{F}, and S denote the obstacle space, free space, and total space respectively, where $S = S_{\mathrm{O}} \cup S_{\mathrm{F}}$ and $S_{\mathrm{O}} \cap S_{\mathrm{F}} = \emptyset$. We use the notation $\mathbb{S}(\cdot)$ to denote the subset of space associated with a data structure element, e.g., $H = \mathbb{S}(v)$ is the hyper-rectangle of v. We also use $\mathbb{P}(\cdot)$ to denote probability, and $\mathscr{L}(\cdot)$ to denote the Lebesgue measure, e.g., $\mathscr{L}(S_{\mathrm{F}})$ is the hyper-volume of the free space. We assume that the configuration space is bounded and that the boundaries of S_{O}, S_{F}, and S have measure zero.

Let C denote the set of children of v. Each child $c_i \in C$ represents a subset of $\mathbb{S}(v)$ such that $\bigcup_i \mathbb{S}(c_i) = \mathbb{S}(v)$ and $\mathbb{S}(c_i) \cap \mathbb{S}(c_j) = \emptyset$ for all $i \neq j$. In a kd-tree $|C| = 2$.

Note the wording in this section is tailored to the kd-tree version of our algorithm; however, the analysis is generally applicable to any related data-structure[4].

Let $f_n(\cdot)$ be the probability density function for the sample returned by Algorithm 1, when the kd-tree contains n points. Let $f_{\mathrm{F}}(\cdot)$ represent a probability density function such that $f_{\mathrm{F}}(x_a) = f_{\mathrm{F}}(x_b)$ for all $x_a, x_b \in S_{\mathrm{F}}$ and $f_{\mathrm{F}}(x_c) = 0$ for all $x_c \in S_{\mathrm{O}}$. Let X_{F} and X_n denote random variables drawn from the distributions defined by $f_{\mathrm{F}}(\cdot)$ and $f_n(\cdot)$, respectively.

In the appendix, we prove that $\mathbb{P}(\lim_{n\to\infty} f_n(x) = f_{\mathrm{F}}(x)) = 1$, for almost all $x \in S$, i.e., that the induced distribution of our algorithm converges to a distribution that is almost surely equal to $f_{\mathrm{F}}(x)$ almost everywhere in S, possibly excluding a measure-zero subset.

B. Performance and runtime

We now discuss the runtime of our algorithm (kd-tree based) in order to evaluate when it should be used vs. more traditional rejection sampling.

Let c_{kd} be the work associated with performing graph operations (nearest neighbor searching, or insertion) on a standard kd-tree. Let c_{gen} denote the time to generate a sample from a distribution uniform over a hyper-rectangle, and let c_{cc} denote the time required to collision check a point. Note that $c_{kd} = \mathscr{O}(\log n)$ for a balanced kd-tree of n points and $c_{gen} = \mathscr{O}(d)$, where d is the dimensionality of the configuration space. As compared to rejection sampling, our method changes the computation time required to generate a candidate sample from c_{gen} to $c_{gen} \cdot c_{kd}$.

Our method also changes the expected number of trials required to sample a point $x \in S_{\mathrm{F}}$ from S/S_{F} to $\mathbb{E}_{trials}$, where $\mathbb{E}_{trials} = S/S_{\mathrm{F}}$ for the first sample ($n = 1$) and then $\lim_{n\to\infty} \mathbb{E}_{trials} = 1$ (i.e., $\mathbb{E}_{trials}$ approaches 1 as the number of successful samples increases).

Thus, our method changes the expected time to both find a point $x \in S_{\mathrm{F}}$ and then perform graph operations on kd-tree from $(c_{gen} + c_{cc})(S/S_{\mathrm{F}}) + c_{kd}$ to $(c_{gen} \cdot c_{kd} + c_{cc})\mathbb{E}_{trials}$. Note that S/S_{F} is a constant (assuming a static environment). We expect our method to have a practical improvement in cases where we may reasonably expect $\mathbb{E}_{trials} < (S/S_{\mathrm{F}})(1/c_{kd})$.

V. Experiments and Results

In order to validate our method and profile its performance we perform several experiments for different robotic systems and obstacle sets. We present results for three systems: a planar point, a concave planar object, and a 4-link thin-arm manipulator in the plane. Figure 2 shows the obstacle sets used for the the planar object and the manipulator, respectively. Those used for the planar point are identical to those used for the planar object.

[3] k-nearest neighbor, range search, and many other proximity queries follow this same general structure.

[4] In particular, we only require a tree-based space partitioning spatial index that is theoretically capable of containing any countably infinite set of points $\mathcal{X}$, and such that the hyper-space of the leaf nodes covers the configuration space $S = \bigcup_{v \in \mathcal{V}^{\mathrm{L}}} \mathbb{S}(v)$. Our proofs can be modified to the general case by replacing 'hyper-rectangle' with 'hyper-space' and assuming that a weighted die determines the recursion path (instead of a coin). When the die is thrown at v it has $|C|$ sides and the weight of the i-th side is determined by the estimated value of $\mathscr{L}(\mathbb{S}(c_i))/\mathscr{L}(\mathbb{S}(v))$ (i.e. the relative amount of free space believed to exist in child c_i vs. its parent v).

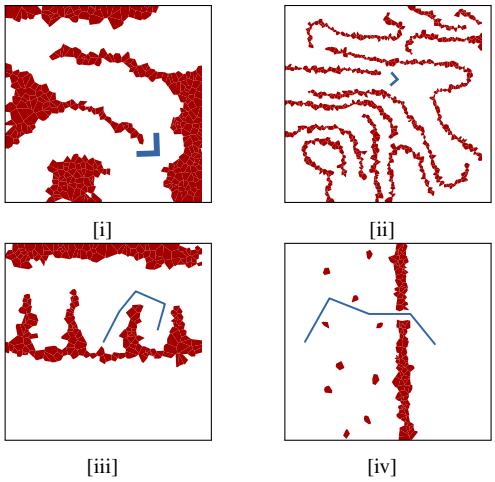


Fig. 2: Obstacle sets for experiments. For experiments with the planar point and planar object, [i] and [ii] are mazes of different complexity. For the planar manipulator [iii] is a multi-crevice obstacle set with the manipulator in the center crevice and [iv] is a wall with a narrow opening.

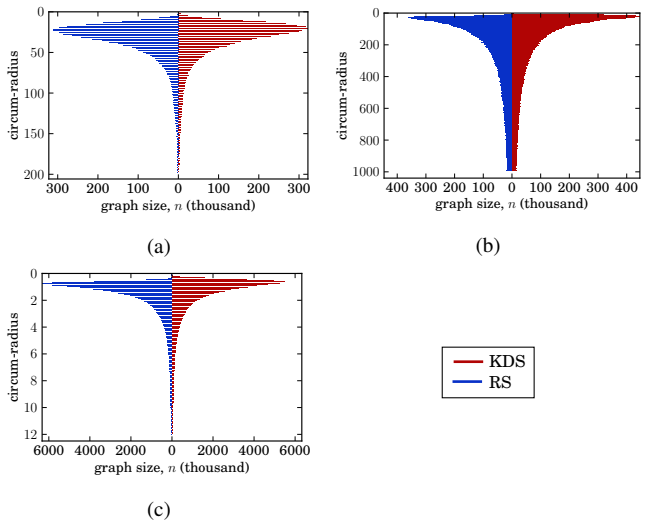


Fig. 3: Histogram of circumsphere sizes for the Delaunay Triangulation of the point set generated after 1000 collision-free samples in experiments with (a) the planar point and obstacle set [i], (b) the planar object and obstacle set [i], (c) the planar manipulator and obstacle set [iii].

In each experiment we compare the results for both (1) sampling a collision-free configuration and (2) sampling a collision-free configuration *and* finding it's nearest neighbor among previously sampled collision-free configurations. We compare the results for (a) the kd-tree implementation described in Section III and (b) classical rejection sampling. We use the abbreviations KDS (1a), KDSS (2a), RS (1b) and RSS (2b). Note that (2b) is comprised of the subroutines (1b) would replace in an otherwise standard implementation of RRT, RRT*, PRM, or PRM*.

Results are averaged over thirty runs, and collision checking is performed using an axis-aligned bounding box tree.

A. Sampling Performance

The biased sampling method sacrifices independence of the sampling process, so we must question the quality of the induced sampling distribution. We compare the sample sets generated by KDS and RS by comparing the distribution of circumspheres in the Delaunay Triangulation of the point set (the radius of largest circumsphere being the L2 dispersion of the point set [13]).

Figure 3 illustrates the histogram of the radii of these circumspheres for obstacle sets [i] and [iii] using KDS and RS. As these plots show, the point set generated using KDS is quite similar to that of the point set generated using RS. The Kolmogorov-Smirnov p-values for the tests are 1.000, 0.893, and 0.999 respectively, suggesting the way in which KDS sacrifices independence in the sampling process does not significantly affect the quality of the sampling sequence.

B. Planar Point Robot

Figure 4 shows the profiling results for the planar point experiments. In obstacle set [i] the obstacles are grouped to create wide regions of free space, while in [ii] they are arranged to divide up the free-space into many narrow passages. The latter should require a larger number of samples before the information encoded in the kd-tree is sufficient

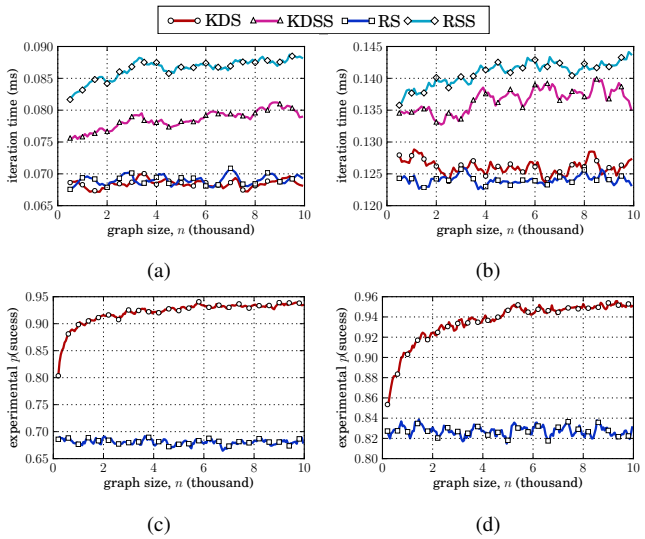


Fig. 4: Observed iteration times (a), (b), and incremental sampling success rates (c), (d), for the planar point experiments. Figures (a) and (c) are for obstacle set [i] while (b) and (d) are for obstacle set [ii].

to improve the sampling success rate in KDSS. Indeed, as Figure 4(d) shows the sampling success rate for this obstacle set increases more slowly than for obstacle set [i], shown in Figure 4(c). Note also that the proportion of the configuration space that is collision-free is higher for obstacle set [ii] than [i]. For these two reasons the iteration time for KDSS and obstacle set [ii] is not much different than for RSS. We see a 3-9% improvement, however, in the runtime time of KDSS over RSS for both obstacle sets as shown in Figures 4(a) and 4(b).

C. Planar Object Robot

While the planar point experiment is useful for gaining some intuition the low dimension of the configuration space and ease of collision checking are somewhat unrepresentative

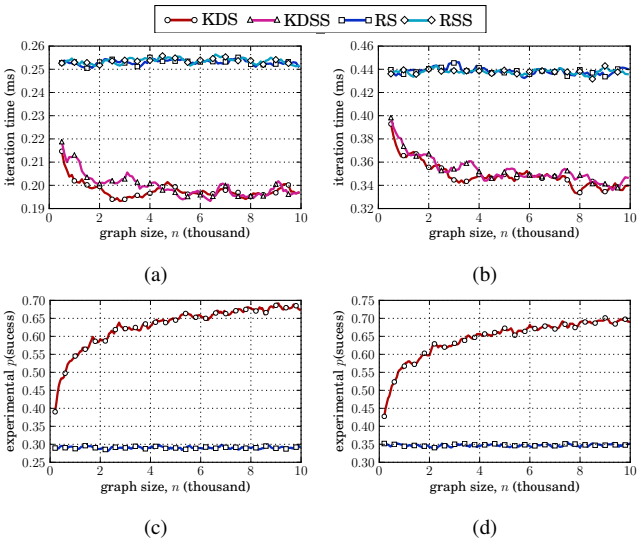


Fig. 5: Observed iteration times (a), (b), and incremental sampling success rates (c), (d), for the planar object experiments. Figures (a) and (c) are for obstacle set [i] while (b) and (d) are for obstacle set [ii].

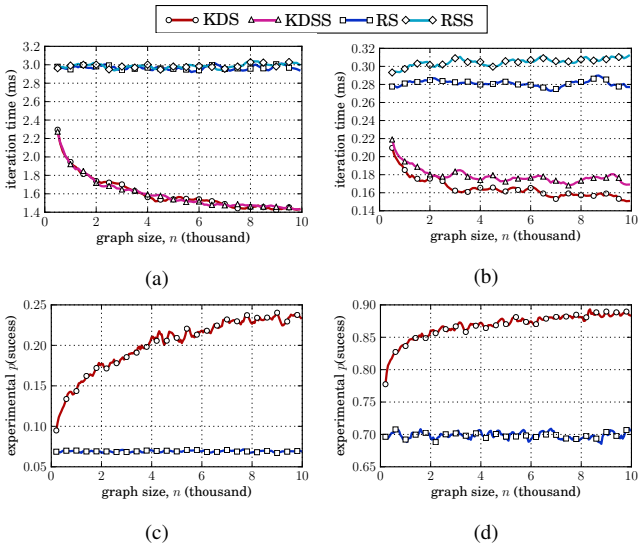


Fig. 6: Observed iteration times (a), (b), and incremental sampling success rates (c), (d), for the planar manipulator experiments. Figures (a) and (c) are for obstacle set [iii] while (b) and (d) are for obstacle set [iv].

of the challenging planning problems which are typically solved with sampling based planning algorithms.

Figure 5 shows the profiling results for the planar object experiments. Both KDS and KDSS are approximately 20% faster than RS and RSS for both obstacle sets [i] (Figure 5(a)) and [ii] (Figure 5(b)). Note that for these obstacle sets, sampling (RS and KDS) alone takes roughly the same amount of time as sampling and searching (RSS and KDSS). Clearly in these experiments sampling (and static collision checking) contributes to the majority of the runtime while graph searching, which is the theoretical bottleneck in sampling based planning algorithms, has yet to begin dominating the runtime with an index of 10,000 points. The experimental sampling success rates (Figures 5(c) and 5(d)) show that the hyper-rectangular decomposition of the configuration space does not cover the configuration space as well in three dimensions as it does in two, with the success rate topping out at around 70% in these experiments. Never the less, this is a significant improvement over the 30%-35% success rate of rejection sampling in these experiments, leading to the significant reduction in runtime.

D. Planar Manipulator Robot

Experiments with the planar manipulator allows us to consider the performance of this method in higher dimension configuration spaces. Figure 6 shows the profiling results for the planar manipulator experiments. Note that obstacle set [iii] has a more complex representation with more of the configuration space in collision. In this case KDS and KDSS yield a 30% runtime improvement over both RS and RSS (Figure 6(a)). For obstacle set [iv] KDS and KDSS yield a 40% runtime improvement (Figure 6(b)). We see in Figure 6(d) that the proportion of the configuration space which is collision-free is quite high in this case (nearly 70% collision-free) which is similar to the proportion of free space in the planar point experiment for obstacle set [i]. However, because collision checking the manipulator requires much more work than the point, the increased sampling success rate yields higher payoff.

We also note that for obstacles set [iii] our method tops out at a sampling success rate of only 25% which is even less than in the planar object experiments. However, just as with the planar object experiments, this increase in sampling success rate is significant enough to lead to the runtime improvement that we observe.

VI. Summary and Conclusions

We present a new method for sampling such that samples are drawn from a distribution that provably converges to uniform sampling over an initially unknown but discoverable and static subset of space. Our method works by recording the observed distribution of the subset of interest vs. the total space within subspaces covered by a spatial index. We demonstrate the specifics of this method with an algorithm for augmenting kd-trees. The observed number of samples (e.g., from the subset of interest vs. the total number samples) at each node is used to guide future sampling.

Our method can also be viewed as constructing an *approximate* obstacle representation that becomes more refined as more samples are generated (and samples are generated from areas that contain obstacles with decreasing probability). However, experiments show that the number of samples required to characterize the free space sufficiently well to reduce rejection sampling over head is relatively small in many complex planning problems. The complexity of generating new samples also inherits a $\log n$ factor of complexity from the kd-tree around which it is built, but in practice this appears to be a useful compromise. In particular, this compromise is worthwhile when

- the complexity of the obstacle field is high, i.e. a large number of and/or high resolution obstacles

- the complexity of interference testing is high, i.e. a complex mapping from (high) configuration space to workspace volumes
- the proportion of the configuration space which is collision-free is low

We expect our method to be especially useful as a subroutine in sample based motion planning which are already bound to search operations on a spatial index.

REFERENCES

[1] Nancy M. Amato, O. Burchan Bayazit, Lucia K. Dale, Christopher Jones, and Daniel Vallejo. Obprm: an obstacle-based prm for 3d workspaces. In *Proceedings of the third workshop on the algorithmic foundations of robotics on Robotics : the algorithmic perspective: the algorithmic perspective*, WAFR '98, pages 155–168, Natick, MA, USA, 1998. A. K. Peters, Ltd.

[2] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517, September 1975.

[3] V. Boor, M. H. Overmars, and A. F. van der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. In *IEEE Int. Conf. on Robotics and Automation*, pages 1018–1023, 1999.

[4] Roland Geraerts and Mark H. Overmars. A comparative study of probabilistic roadmap planners. In *Workshop on the Algorithmic Foundations of Robotics*, pages 43–57, 2002.

[5] W.R. Gilks and P. Wild. Adaptive rejection sampling for gibbs sampling. *Applied Statistics*, pages 337–348, 1992.

[6] B. Glavina. Solving findpath by combination of goal-directed and randomized search. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 1718 –1723 vol.3, may 1990.

[7] D. Hsu, Tingting Jiang, J. Reif, and Zheng Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 3, pages 4420 – 4426 vol.3, sept. 2003.

[8] D. Hsu, J.C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 3, pages 2719–2726. IEEE, 1997.

[9] David Hsu, Lydia E. Kavraki, Jean-Claude Latombe, Rajeev Motwani, and Stephen Sorkin. On finding narrow passages with probabilistic roadmap planners. In *Proceedings of the third workshop on the algorithmic foundations of robotics on Robotics : the algorithmic perspective: the algorithmic perspective*, WAFR '98, pages 141–153, Natick, MA, USA, 1998. A. K. Peters, Ltd.

[10] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. Journal of Robotics Research*, 30(7):846–894, June 2011.

[11] L. Kavraki, P. Svestka, J. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on ...*, January 1996.

[12] J.-P. Laumond and C. Nissoux. Visibility-based probabilistic roadmaps for motion planning. *Journal of Advanced Robotics*, 14(6):477–493, 2000.

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

[14] S. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, 2001.

[15] S.M. LaValle and J.J. Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and computational robotics: new directions: the fourth Workshop on the Algorithmic Foundations of Robotics*, page 293. AK Peters, Ltd., 2001.

[16] Stephen R. Lindemann and Steven M. LaValle. Current issues in sampling-based motion planning. In Paolo Dario and Raja Chatila, editors, *Robotics Research*, volume 15 of *Springer Tracts in Advanced Robotics*, pages 36–54. Springer Berlin / Heidelberg, 2005. 10.1007/11008941_5.

[17] Jeff M. Phillips, L. E. Kavraki, and N. Bedrosian. Spacecraft rendezvous and docking with real-time randomized optimization. In *AIAA (American Institute of Aeronautics and Astronautics) Guidance, Navigation and Control Conference*, Austin, TX, August 2003.

[18] J.M. Phillips, N. Bedrossian, and L.E. Kavraki. Guided expansive spaces trees: a search strategy for motion- and cost-constrained state spaces. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 4, pages 3968 – 3973 Vol.4, 26-may 1, 2004.

[19] E. Plaku, K.E. Bekris, B.Y. Chen, A.M. Ladd, and L.E. Kavraki. Sampling-based roadmap of trees for parallel motion planning. *Robotics, IEEE Transactions on*, 21(4):597 – 608, Aug 2005.

[20] Mitul Saha, Jean-Claude Latombe, Yu-Chi Chang, and Friedrich Prinz. Finding narrow passages with probabilistic roadmaps: The small-step retraction method. *Autonomous Robots*, 19(3):301–319, December 2005.

[21] H. Samet. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.

[22] G. Sánchez and J.C. Latombe. On delaying collision checking in prm planning: Application to multi-robot coordination. *The International Journal of Robotics Research*, 21(1):5–26, 2002.

[23] S.A. Wilmarth, N.M. Amato, and P.F. Stiller. Maprm: a probabilistic roadmap planner with sampling on the medial axis of the free space. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 2, pages 1024 –1031 vol.2, 1999.

APPENDIX

We begin by observing that the nodes in the kd-tree may be classified into three sets:

- free nodes, $\mathcal{V}^{\mathrm{F}}$, the set of nodes v such that $\mathscr{L}(\mathbb{S}(v) \cap S_{\mathrm{O}}) = 0$ and $\exists\, x \,|\, x \in \mathbb{S}(v) \wedge x \in S_{\mathrm{F}}$.
- obstacle nodes, $\mathcal{V}^{\mathrm{O}}$, the set of nodes, v such that $\mathscr{L}\left(\mathbb{S}\left(v^{\mathrm{O}}\right) \cap S_{\mathrm{F}}\right) = 0$ and $\exists\, x \,|\, x \in \mathbb{S}(v) \wedge x \in S_{\mathrm{O}}$
- mixed nodes, $\mathcal{V}^{\mathrm{M}}$ contains all nodes do not fit the definition of a free node or a mixed node.

Although our algorithm is ignorant of the type of any given node (otherwise we would not need it to begin with), an oracle would know that free nodes contain free space almost everywhere, obstacle nodes contain obstacle space almost everywhere, and mixed nodes contain both free space and obstacle space. Note that if $v \in \mathcal{V}^{\mathrm{M}}$ and $\mathscr{L}(\mathbb{S}(v)) > 0$ then $\mathscr{L}(\mathbb{S}(v) \cap S_{\mathrm{O}}) > 0$ and $\mathscr{L}(\mathbb{S}(v) \cap S_{\mathrm{F}}) > 0$. It is possible for v such that $\mathscr{L}(\mathbb{S}(v)) = 0$ to be both an obstacle node and a free node if it exists on the the boundary between S_{O} and S_{F}; because their cumulative measure is zero, such nodes can be counted as both obstacle nodes and free nodes (or explicitly defined as either one or the other) without affecting our results.

We are particularly interested in the types of leaf nodes, because they cover S and also hold all of the mass that determines the induced sampling distribution.

Let $\mathcal{V}^{\mathrm{L}}$ denote the set of leaf nodes, and let $\mathcal{V}^{\mathrm{FL}}$, $\mathcal{V}^{\mathrm{OL}}$, $\mathcal{V}^{\mathrm{ML}}$ denote the set of leaf nodes that are also free nodes, obstacle nodes, and mixed nodes, respectively. We use $\mathbb{S}(\mathcal{V}) = \bigcup_{v \in \mathcal{V}} \mathbb{S}(v)$ to denote the space contained in all nodes in a set $\mathcal{V}$. Figure 7 depicts the space contained in the set of leaf nodes, $\mathcal{V}^{\mathrm{L}}$, of a particular kd-tree.

Recall that $v.M$ is the estimated probability mass that our algorithm associates with node v. Let D denote tree depth.

Proposition 1. $\mathbb{P}(X_n \in \mathbb{S}(v)) = v.M / \sum_{v' \in \mathcal{V}^{\mathrm{L}}} v'.M$

Proof. This is true by the construction of our algorithm. In particular, from lines 16-17 and 21. □

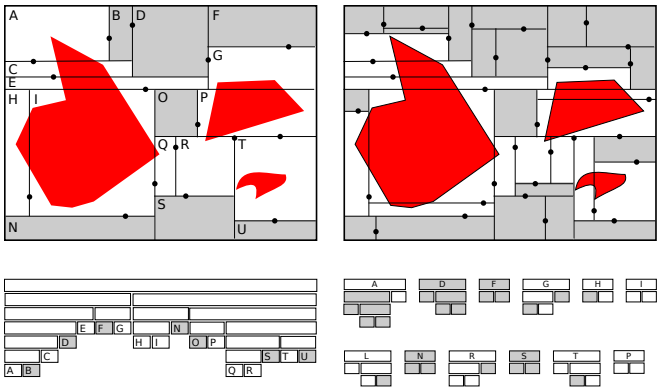


Fig. 7: The hyper-rectangles of leaf nodes (Top) from the corresponding kd-trees (Bottom). Letters show the correspondence between nodes and their hyper-rectangles. Left and Right show 28 and 41 points, respectively. Obstacle space is red. Free nodes are gray and mixed nodes are white. Letters show the correspondence. Descendants of a free node are always free. Mixed nodes eventually produce free node descendants (the probability that an obstacle node is produced is 0).

Proposition 2. For all v at depth $D > 1$ such that $\mathscr{L}(\mathbb{S}(v)) > 0$ there exists some $\delta > 0$ such that $v.F > \delta$.

Proof. All nodes at depth $D > 1$ have a parent $v.P$, which must have generated at least one sample in order to create v. Since $\mathscr{L}(\mathbb{S}(v)) > 0$, we know that $\mathscr{L}(\mathbb{S}(v.P)) > 0$. Therefore, by construction (lines 7, 11, 13) we know $v.F \geq \frac{\mathscr{L}(\mathbb{S}(v))}{\mathscr{L}(\mathbb{S}(v.P))} > 0$. Thus, the lemma is true for δ such that $0 < \delta < \frac{\mathscr{L}(\mathbb{S}(v))}{\mathscr{L}(\mathbb{S}(v.P))}$. □

Lemma 3. For a particular node v, let $N_n(v)$ be the number of times that a sample was generated from $\mathbb{S}(v)$ when the kd-tree has n nodes. Then, for all v such that $\mathscr{L}(\mathbb{S}(v)) > 0$, $\mathbb{P}(\lim_{n\to\infty} N_n(v) = +\infty) = 1$.

Proof. We begin by obtaining two intermediate results:

First, $v.F \leq v.T$ for all v by construction (lines 3, 5, 7, 11, 13). Thus, for all leaf nodes $v \in \mathcal{V}^{\mathrm{L}}$ it is guaranteed $v.M \leq \mathscr{L}(\mathbb{S}(v))$ by the definition of M (line 12). Recall that the set of leaf nodes covers the space $\mathbb{S}(\mathcal{V}^{\mathrm{L}}) = S$ and that the space in each leaf node is non-overlapping $\mathbb{S}(v_i) \cap \mathbb{S}(v_j) = \emptyset$ for all $v_i, v_j \in \mathcal{V}^{\mathrm{L}}$, $v_i \neq v_j$. Thus, we can sum over all leaf nodes to obtain the bound: $\sum_{v\in\mathcal{V}^{\mathrm{L}}} v.M \leq \sum_{v\in\mathcal{V}^{\mathrm{L}}} \mathscr{L}(\mathbb{S}(v)) = \mathscr{L}(S)$.

Second, using Proposition 2 we know that for any particular node v with positive measure $\mathscr{L}(\mathbb{S}(v)) > 0$ there exists some δ such that $v.F > \delta$.Thus, the following bound always holds: $v.M = \mathscr{L}(\mathbb{S}(v)) \frac{v.F}{v.T} \geq \mathscr{L}(\mathbb{S}(v)) \frac{\delta}{v.T}$ (where the first equality is by definition). Note this is the *worst case* situation in which node v always samples from obstacle space (and thus v remains a leaf node forever). Thus, $v.M \geq \mathscr{L}(\mathbb{S}(v)) \frac{\delta}{v.T}$.

Combining the first and second results yields:

$$\mathbb{P}(X_n \in \mathbb{S}(v)) = \frac{v.M}{\sum_{v'\in\mathcal{V}^{\mathrm{L}}} v'.M} \geq \frac{\delta\mathscr{L}(\mathbb{S}(v))}{v.T\mathscr{L}(S)}$$

Where the left equality is by Proposition 1. By definition $\frac{\delta\mathscr{L}(\mathbb{S}(v))}{\mathscr{L}(S)} = k$ is a constant, and so $\mathbb{P}(X_n \in \mathbb{S}(v)) \geq \frac{k}{v.T}$. By definition, $v.T$ only increase when we draw a sample from $\mathbb{S}(v)$. Let $\hat{n}$ be the iteration at which the previous sample was generated from $\mathbb{S}(v)$. The probability that we *never again* generate a sample from $\mathbb{S}(v)$ is bounded:

$$\mathbb{P}\left(\lim_{n\to\infty} N_n(v) = N_{\hat{n}}(v)\right) \leq \lim_{n\to\infty} \prod_{i=\hat{n}}^{n} \left(1 - \frac{k}{v.T}\right) = 0$$

for all $v.T < \infty$ and $N_{\hat{n}}(v) < \infty$ (and thus $\hat{n} < \infty$). The rest of the proof follows from induction. □

Lemma 4. Let $\mathcal{V}_n^{\mathrm{F}}$ be the set of free nodes in the tree of n samples. Then, for all $x \in S_{\mathrm{F}}$, $\lim_{n\to\infty} \mathbb{P}\left(\exists v \in \mathcal{V}_n^{\mathrm{F}} \,|\, x \in \mathbb{S}(v)\right) = 1$

Proof. Let $\Xi_{\epsilon,x}$ be the open L1-ball with radius ϵ that is centered at point x. For all $x \in \mathrm{int}(S_{\mathrm{F}})$ there exists some $\epsilon > 0$ for which $\Xi_{\epsilon,x} \subset S_{\mathrm{F}}$. Therefore, it is sufficient to prove that for $x \in S_{\mathrm{F}}$, $\lim_{n\to\infty} \mathbb{P}\left(\exists v \in \mathcal{V}_n^{\mathrm{F}} \,|\, v \subset \Xi_{\epsilon,x}\right) = 1$.

Without loss of generality, we now consider a particular x. At any point during the run of the algorithm there is some leaf node $v_{\mathrm{L}} \,|\, v_{\mathrm{L}} \ni x$.

Lemma 3 guarantees that $v_{\mathrm{L}} \ni x$ will almost surely split into two children, one of which will also contain x, etc. Let $v_{D,x}$ represent the node at depth D that contains x. Let $X_D \in \mathbb{S}(v_{D,x})$ be the sample point that causes $v_{D,x}$ to split. Let $x[i]$ refer to the i-th coordinate of x. Thus, the splitting plane is normal to the $D \bmod d$-axis, and intersects that axis at $X_D[D \bmod d]$, where d is the dimensionality of the space.

Each time the current leaf $v_{D,x} \ni x$ splits $\mathbb{P}(X_D \in \Xi_{\epsilon,x} \wedge X_D[i] < x[i]) = \frac{\mathscr{L}(\mathbb{S}(v_{D,x}) \cap \mathbb{S}(\Xi_{\epsilon,x}))}{2\mathscr{L}(\mathbb{S}(v_{D,x}) \cap S_{\mathrm{F}})} > 0$. By construction $\frac{\mathscr{L}(\mathbb{S}(v_{D+d,x}) \cap \mathbb{S}(\Xi_{\epsilon,x}))}{\mathscr{L}(\mathbb{S}(v_{D+d,x}) \cap S_{\mathrm{F}})} \geq \frac{\mathscr{L}(\mathbb{S}(v_{D,x}) \cap \mathbb{S}(\Xi_{\epsilon,x}))}{\mathscr{L}(\mathbb{S}(v_{D,x}) \cap S_{\mathrm{F}})}$ so $\lim_{D\to\infty} \mathbb{P}(\exists X_D \,|\, X_D \in \Xi_{\epsilon,x} \wedge X_D[D \bmod d] < x[d \bmod D]) = 1$. A similar argument can be made for $X[i] > x[i]$, $\lim_{D\to\infty} \mathbb{P}(\exists X_D \,|\, X_D \in \Xi_{\epsilon,x} \wedge X_D[D \bmod d] > x[D \bmod d]) = 1$. Thus, in the limit as $D \to \infty$, there will almost surely be a set of $2d$ points $\{X_{D_1}, \ldots, X_{D_{2d}}\}$ sampled at levels $D_1, \ldots, D_{2d}$, such that $X_{D_i} \in \Xi_{\epsilon,x}$ for $i = \{1, \ldots, 2d\}$, and $i = D_i \bmod d$ and $X_{D_i}[i] < X[i]$, and $i = D_{d+i} \bmod d$ and $X_{D_{d+i}}[i] > X[i]$. By construction, $X_{\max_i(D_i)}$ is on a splitting plane that borders a node v such that $\mathbb{S}(v) \ni x$ and $v \subset \Xi_{\epsilon,x}$ (and thus $v \in \mathcal{V}^{\mathrm{F}}$). Lemma 3 implies that $\mathbb{P}(\lim_{n\to\infty} D = \infty) = 1$ for $v_{D,x} \,|\, \mathbb{S}(v_{D,x}) \ni x$. □

Corollary 5. $\mathbb{P}\left(\lim_{n\to\infty} \mathscr{L}\left(\mathbb{S}\left(\mathcal{V}^{\mathrm{ML}}\right) \cap S_{\mathrm{F}}\right) = 0\right) = 1$.

Corollary 6. $\mathbb{P}\left(\lim_{n\to\infty} \mathscr{L}\left(S_{\mathrm{F}} \setminus \mathbb{S}\left(\mathcal{V}^{\mathrm{FL}}\right)\right) = 0\right) = 1$.

Corollary 7.
$\mathbb{P}\left(\lim_{n\to\infty} \sum_{v\in\mathcal{V}^{\mathrm{FL}}} \mathscr{L}(\mathbb{S}(v)) = \mathscr{L}(S_{\mathrm{F}})\right) = 1$.

Lemma 8. $\mathbb{P}(\lim_{n\to\infty} v.M = 0) = 1$ for all obstacle leaf nodes $v \in \mathcal{V}^{\mathrm{OL}}$.

Proof. There are two cases, one for $\mathscr{L}(\mathbb{S}(v)) = 0$ and another for $\mathscr{L}(\mathbb{S}(v)) > 0$. The first is immediate given $v.M \triangleq \frac{v.F}{v.T}\mathscr{L}(\mathbb{S}(v))$. For the second, we observe that $\mathbb{P}(\exists x \,|\, x \in \mathbb{S}(v) \wedge x \in S_{\mathrm{F}}) = 0$ by definition, and so $v.F$ will almost surely not change (and v will remain a leaf node almost surely). Thus, $\mathbb{P}(v.T = \infty) = 1$ by Lemma 3, and

so $\mathbb{P}\left(\lim_{n\to\infty} \frac{v.F}{v.T} = 0\right) = 1$. Using the definition of $v.M$ finishes the proof. □

Corollary 9. $\lim_{n\to\infty} \mathbb{P}\left(X_n \in \mathbb{S}\left(\mathcal{V}^{\mathrm{OL}}\right)\right) = 0.$

Lemma 10. $\mathbb{P}\left(\lim_{n\to\infty} v.M = 0\right) = 1$ for all mixed leaf nodes $v \in \mathcal{V}^{\mathrm{ML}}$.

Proof. $v.F$ will almost surely not change by Corollary 5. The rest of the proof is similar to Lemma 8. □

Corollary 11. $\lim_{n\to\infty} \mathbb{P}\left(X_n \in \mathbb{S}\left(\mathcal{V}^{\mathrm{ML}}\right)\right) = 0.$

Corollary 12.
$\lim_{n\to\infty} \mathbb{P}\left(X_n \in \left(\mathbb{S}\left(\mathcal{V}^{\mathrm{ML}}\right) \cup \mathbb{S}\left(\mathcal{V}^{\mathrm{OL}}\right)\right)\right) = 0$

We observe that this result does not conflict with Lemma 3. Each node with finite space is sampled an infinite number of times; however, the proportion of samples from obstacle nodes and mixed nodes approaches 0 in the limit as $n \to \infty$.

Lemma 13. $\lim_{n\to\infty} \mathbb{P}\left(X_n \in S_{\mathrm{O}}\right) = 0$

Proof. This follows from Corollary 12 and the fact that $\mathscr{L}\left(S_{\mathrm{O}} \setminus \left(\mathbb{S}\left(\mathcal{V}^{\mathrm{ML}}\right) \cup \mathbb{S}\left(\mathcal{V}^{\mathrm{OL}}\right)\right)\right) = 0.$ □

Lemma 14. $\mathbb{P}\left(\lim_{n\to\infty} v.M = \mathscr{L}\left(\mathbb{S}\left(v\right)\right)\right) = 1$, for all free nodes $v \in \mathcal{V}^{\mathrm{F}}$.

Proof. There are two cases, one for when $\mathscr{L}\left(\mathbb{S}\left(v\right)\right) = 0$ and another for when $\mathscr{L}\left(\mathbb{S}\left(v\right)\right) > 0$. The former is immediate given the definition of $v.M$, and so we focus on the latter. When a new free node $v_D \in \mathcal{V}^{\mathrm{F}}$ is created at depth $D > 1$ of the tree it initializes $v_D.F > 0$ and $v_D.T > 0$ based on similar values contained in its parent (and wighted by the relative measures of v_D vs. its parent). By Lemma 3 we know that v_D will almost surely generate two children $v_{D+1,0}$ and $v_{D+1,1}$. By construction (lines 11-13), they will be initialized with $v_{D+1,j}.F = (v_D.F + 1)\frac{\mathscr{L}(\mathbb{S}(v_{D+1,j}))}{\mathscr{L}(\mathbb{S}(v_D))}$ and $v_{D+1,j}.T = (v_D.T + 1)\frac{\mathscr{L}(\mathbb{S}(v_{D+1,j}))}{\mathscr{L}(\mathbb{S}(v_D))}$, for $j \in \{0, 1\}$. These children will also generate their own children almost surely, etc. Because v_D is a free node, all samples from its subtree will result in more *free node* descendants being created almost surely. Let $\hat{C}_n$ be the set containing all leaf node descendants of v_D at iteration n. By construction (line 24), as soon as $|\hat{C}_n| \geq 1$, then $v_D.M = \sum_{v\in\hat{C}_n} v.M$. We now examine a single term of the latter summation, i.e., the term for node v_{D+k} at depth $D + k$. In particular. $v_{D+k}.M = \frac{v_{D+k}.F}{v_{D+k}.T}\mathscr{L}\left(\mathbb{S}\left(v_{D+k}\right)\right)$. For the remainder of this proof we will abuse our notation and let $\|\cdot\| = \mathscr{L}\left(\mathbb{S}\left(\cdot\right)\right)$ to make the following equations more readable. Unrolling the recurrence relation for $v_{D+k}.F$ gives:

$$v_{D+k}.F = \frac{\|v_{D+k}\|}{\|v_{D+k-1}\|}\left(\ldots\frac{\|v_{D+2}\|}{\|v_{D+1}\|}\left(\frac{\|v_{D+1}\|}{\|v_D\|}\left(v_D.F + 1\right) + 1\right)\ldots + 1\right)$$

where $v_{D+k-1}, \ldots, v_{D+2}, v_{D+1}, v_D$, are the ancestors of v_{D+k} going up the tree to v_D. This can be rearranged:

$$v_{D+k}.F = \frac{\|v_{D+k}\|}{\|v_D\|}v_D.F + \frac{\|v_{D+k}\|}{\|v_D\|} + \frac{\|v_{D+k}\|}{\|v_{D+1}\|} + \ldots + \frac{\|v_{D+k}\|}{\|v_{D+k-1}\|}$$

Similarly, the $v_{D+k}.T$ recurrence relation is:

$$v_{D+k}.T = \frac{\|v_{D+k}\|}{\|v_D\|}v_D.T + \frac{\|v_{D+k}\|}{\|v_D\|} + \frac{\|v_{D+k}\|}{\|v_{D+1}\|} + \ldots + \frac{\|v_{D+k}\|}{\|v_{D+k-1}\|}$$

$\lim_{k\to\infty} \frac{\|v_{D+k}\|}{\|v_D\|} = 0$, also $\mathbb{P}\left(\frac{\|v_{D+k}\|}{\|v_{D+k-1}\|} = 0\right) = 0$ given $\mathscr{L}\left(\mathbb{S}\left(v_D\right)\right) > 0$, where we resume our normal notation. Thus, $\mathbb{P}\left(\lim_{k\to\infty} v_{D+k}.M = \mathscr{L}\left(\mathbb{S}\left(v_{D+k}\right)\right)\right) = 1$.

Lemma 3 guarantees that $\mathbb{P}\left(\lim_{n\to\infty} k = \infty\right) = 1$ for all v_{D+k} such that $\hat{C}_n \ni v_{D+k}$. Thus, by summing over the members of $\hat{C}_n$ we get: $\mathbb{P}\left(\lim_{n\to\infty} \sum_{v\in\hat{C}_n} v.M = \sum_{v\in\hat{C}_n} \mathscr{L}\left(\mathbb{S}\left(v\right)\right)\right) = 1$. $v_D.M = \sum_{v\in\hat{C}_n} v.M$ by definition. Also by definition $\mathbb{S}\left(v_D\right) = \bigcup_{v\in\hat{C}_n} \mathbb{S}\left(v\right)$ and $v_i \cap v_j = \emptyset$ for all $v_i, v_j \in \hat{C}_n$ such that $i \neq j$; therefore, $\mathscr{L}\left(\mathbb{S}\left(v_D\right)\right) = \sum_{v\in\hat{C}_n} \mathscr{L}\left(\mathbb{S}\left(v\right)\right)$. Substitution finishes the proof. □

Note, Corollary 15 depends on Lemma 14 and Corollary 6:

Corollary 15.
$\mathbb{P}\left(\lim_{n\to\infty} \sum_{v\in\mathcal{V}^{\mathrm{FL}}} v.M = \mathscr{L}\left(S_{\mathrm{F}}\right)\right) = 1.$

Lemma 16. $\mathbb{P}\left(\lim_{n\to\infty} f_n\left(x\right) = c\right) = 1$ for all x and B such that $x \in B \subset S_{\mathrm{F}}$ and $\mathscr{L}\left(B\right) > 0$

Proof. By Proposition 1 and Lemmas 14 and Corollary 15 we know that $\lim_{n\to\infty} \mathbb{P}\left(x_n \in \mathbb{S}\left(v\right)\right) = \frac{\mathscr{L}(\mathbb{S}(v))}{\mathscr{L}(S_{\mathrm{F}})}$ for all free nodes $v \in \mathcal{V}^{\mathrm{F}}$ almost surely. By construction (line 2) once a leaf node $v \in \mathcal{V}^{\mathrm{FL}}$ is reached, samples are drawn uniformly from within $\mathbb{S}\left(v\right)$. Thus, the uniform probability density of drawing $x_n \in \mathbb{S}\left(v\right)$, given that the algorithm has decided to draw from within $\mathbb{S}\left(v\right)$, is $f_n\left(x_n | x_n \in \mathbb{S}\left(v\right)\right) = \frac{1}{\mathscr{L}(\mathbb{S}(v))}$. Therefore, the posterior probability density $\lim_{n\to\infty} f_n\left(x_n\right) = \lim_{n\to\infty} f_n\left(x_n | x_n \in \mathbb{S}\left(v\right)\right)\mathbb{P}\left(x_n \in \mathbb{S}\left(v\right)\right) = \frac{1}{\mathscr{L}(S_{\mathrm{F}})}$ almost surely, which is constant and *independent* of $v \in \mathcal{V}^{\mathrm{FL}}$, and thus holds almost everywhere in $\bigcup_{v\in\mathcal{V}^{\mathrm{FL}}} \mathbb{S}\left(v\right)$—and thus almost everywhere in S_{F} (by Corollary 6). □

Theorem 17. $\mathbb{P}\left(\lim_{n\to\infty} f_n\left(x\right) = f_{\mathrm{F}}\left(x\right)\right) = 1.$

Proof. This is proved by combining Lemmas 13 and 16. □