Learning Adaptive Sampling Distributions for Motion Planning by Self-Imitation

Ratnesh Madaan*, Sam Zeng*, Brian Okorn, Sebastian Scherer

***Abstract*— Sampling based motion planning algorithms are widely used due to their effectiveness on problems with large state spaces by incremental tree growth in conjunction with uniform, random sampling. The major bottleneck in the performance of such algorithms is the amount of collision checks performed, which in turns depends on the sampling distribution itself. In this work, we present a framework to learn an *adaptive, non-stationary* sampling distribution which explicitly minimizes the search effort, given by the amount of collision checks performed. Our framework models the sequential nature of the problem by leveraging both the instantaneous search tree over the robot configuration space, as well as the workspace environment, by encoding them with a conditional variational auto-encoder, to learn a stochastic sampling policy. We encode the workspace environment with a convolutional network, and the configuration space planning tree with a recurrent neural network. We introduce an *approximate oracle* which can return multiple label samples for a partially solved planning problem, by forward simulating it. We use an imitation via iterative supervised learning framework to learn a stochastic sampling policy. We call this self-supervised imitation of an oracle generated by forward simulation as self-imitation. We validate our approach on a 4D kinodynamic helicopter planning problem with glideslope and curvature constraints, and a 2D holonomic problem.**

I. Introduction

Sampling based motion planning algorithms are widely used due to their effectiveness in problems with large state spaces, via incremental tree growth in conjunction with uniform, random sampling [1]. Previous work has focused on their worst case and asymptotic performance guarantees. However, real life problems are time-constrained, and often what is needed instead is good finite-time performance [2]. The major bottleneck in the performance of such algorithms is the amount of collision checks performed, which in turns depends on the sampling distribution itself. Biasing the sampling distribution with manually defined *stationary* heuristics helps alleviate this to some extent [3, 4]; however due to the sequential nature of the problem, we expect a non-stationary, *adaptive* sampling distribution as the search tree progresses to perform better intuitively. In this work, we present a framework to learn such an adaptive distribution which explicitly minimizes the search effort, given by the amount of collision checks performed for the feasible path planning problem.

We formulate the problem as a Markov Decision Process, where the state space is a concatenation of the workspace environment and the planning graph in the configuration space. The optimal sampling distribution is then a stochastic policy over the continuous configuration space of the robot, such that it minimizes the expected number of collision checks. For sample efficient training, we introduce an *approximate oracle* which returns multiple label samples given a partially solved planning problem, by forward simulating the problem. Notably, we show that we can improve a single-query motion planning algorithm, the Rapidly Exploring Random Trees [5] by forwarding-simulating RRT itself, akin to the self-supervision paradigm. We use a Conditional Variational Auto-Encoder (CVAE) [6] to learn this policy in an imitation learning framework in which we use both experience replay [7] and dataset aggregation [8]. Our framework is general, and can be applied to other domains to generate a stochastic oracle by forward simulating environment dynamics and returning a set of the most promising actions, and then imitating this oracle in a self-supervised manner. Hence, the name *self-imitation*. To imitate the oracle, our CVAE uses a 2D or 3D Convolutional Neural Network (CNN) to encode the workspace environment, and an RNN to encode the instantaneous tree at each time step. Although in this work, we focus on the feasible path problem by learning a sampling distributions for RRTs, such a learned distribution can be wrapped inside any optimal algorithm like RRT*, Batch Informed Trees (BIT*) or Fast Marching Trees (FMT*) [9–11].

II. Approach

A. Single-query Motion Planning

We start by formally defining the Single-query Motion Planning (SQMP) framework, as depicted in Alg. 1. Let $\mathcal{W} \subset \mathbb{R}^m$ $(m \in 2, 3)$ be the workspace environment in which the robot operates. Let $\mathcal{C} \subset \mathbb{R}^n$ be the configuration space of the robot, and $\mathcal{C}_{obs} \subset \mathcal{C}$ be the set of invalid states in collision. The free configuration space is then given by $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$. Let the start configuration and goal configurations be given by q_{start} and q_{goal}. Initially, the start configuration as added as the root vertex of the tree, $\mathcal{T}$ (Line 1). Then, a random sample q_{rand} is drawn from the configuration space (Line 4), and an attempt is made to add it to $\mathcal{T}$ by the `Extend` function. First, the nearest node to q_{rand} in $\mathcal{T}$, $q_{nearest}$ is found (Line 10). Then a local planner given by a `Steer` function finds a feasible connection moving from $q_{nearest}$ in the direction of q_{rand} while respecting the robot's constraints. If successful, a new configuration, q_{new} is added to $\mathcal{T}$, according to a greediness or growth-factor hyperparameter, ϵ. The previous two steps

*Equal Contribution. All authors are with the Robotics Institute at Carnegie Mellon University, Pittsburgh, Pennsylvania, USA. [ratneshmadaan, samlzeng]@gmail.com, [bokorn, basti]@andrew.cmu.edu

are repeated until the goal configuration is reached. Alg 1 essentially defines how RRTs [5] works. Most of the other single query algorithms are extensions and modifications of the same, for example including bidirectional search (RRT-Connect) [12] and optimality by rewiring the tree (RRT*) [9].

B. SQMP as a Markov Decision Process

The problem of biasing the sampling distribution to minimize the number of expected collision checks can be interpreted as a sequential decision making problem. The associated Markov Decision Process (MDP) can be defined over the intersection of the space of workspace environments and the space of trees built in the configuration space using the corresponding planning algorithm.

Formally, at any time step t, the state $s_t \in \mathcal{S}$ is given by the tuple of the workspace environment $\mathcal{W}$ (assumed to be static) and the current planning tree $\mathcal{T}_t$: $s_t = \{\mathcal{T}_t, \mathcal{W}\}$. $\mathcal{T}_t$ itself is defined by its vertices (nodes) and edge sets: $\mathcal{T}_t = \{V, E\}$. Each node's value is given by $v_i \in \mathcal{C}_{free}$, and each edge means that there is a feasible connection from the corresponding parent node to its child node. The action space $\mathcal{A}$ is continuous and is given by the configuration space itself, $\mathcal{C}$. At any time-step t, the action is given by a sample: $a_t = q_{sample} \in \mathcal{C}$. A stochastic policy can then be defined: $\pi(a_t|s_t) = \pi(q_{sample}|\{\mathcal{T}_t, \mathcal{W}\})$. The transition dynamics are defined by the `Extend` function (Line 9, Alg 1): $P(s_{t+1}|s_t, a_t) = P(\{\mathcal{T}_{t+1}, \mathcal{W}\}|\{\mathcal{T}_t, \mathcal{W}\}, q_{sample})$. If a collision free extension is possible while moving from $q_{nearest}$ to q_{sample}, a q_{new} is added to $\mathcal{T}_t$ (Line 13). Otherwise, $\mathcal{T}_t$ does not change. If $q_{sample} \subset \mathcal{C}_{obs}$, we declare the action to be invalid and ignore it, moving to the next iteration (Line 5). It should be noted that one would assign a high negative reward for such an invalid sample in a reinforcement learning setting, but in our imitation learning framework, we ignore negative samples.

Our goal is to find an optimal stochastic sampling policy π^*, which minimizes the expected number of collision checks, over a dataset of planning problems with similar workspaces and the same robot dynamics. We can formalize this by defining a one-step cost over our MDP, $c_t(s_t, a_t) = c_{collision}(\{\mathcal{T}_t, \mathcal{W}\}, q_{sample})$. $c_{collision}$ is defined as the number of collision checks performed by the `Extend` function while trying to extend $q_{nearest}$ to q_{new}, both of which are defined w.r.t the chosen q_{sample}. If a valid q_{new} is not found or if the tree does not move to promising directions in the configuration space, the search effort is essentially wasted, which is exactly what we want to minimize. The policy can be then evaluated by the cumulative cost over one episode. Episodes are ended if goal region is reached $q_{new} \subset q_{goal}$, or the maximum planning time T has elapsed. Note that even though we use the number of collision checks as the cost, our formulation is general. One could replace the one step cost by other metrics, or a combination of thereof. For instance, another cost worth optimizing is the number of optimal boundary value problems solved in a kinodynamic problem by the `Steer` function [13].

Algorithm 1 Single-query Motion Planning

1: $\mathcal{T}$.`init`(q_{start})
2: `done` = `false`
3: **while** `done` $\neq$ `Reached` **do**
4: $q_{rand} \leftarrow$ `Random_Config()`
5: **if** $q_{rand} \in \mathcal{C}_{obs}$ **then**
6: `continue`
7: $done$ = `Extend`($\mathcal{T}, q_{rand}$)
8: `Return` {$\mathcal{T}$, `path`(q_{start}, q_{goal}) }
9: **procedure** `EXTEND`($\mathcal{T}, q$)
10: $q_{nearest} \leftarrow$ `Nearest`($q, \mathcal{T}$)
11: $q_{new} \leftarrow$ `Steer`($q_{nearest}, q_{rand}, \epsilon$)
12: **if** `ObstacleFree`($q_{nearest}, q_{new}$) **then**
13: $\mathcal{T}$.`add_vertex`(q_{new})
14: $\mathcal{T}$.`add_edge`({$q_{nearest}, q_{new}$})
15: **if** ($q_{new} = q_{goal}$) **then**
16: Return `Reached`
17: **else**
18: Return `Advanced`

Given a prior distribution over environments: $P(\mathcal{W})$, and start and goal states $P(q_{start}, q_{goal})$, we can evaluate our policy in a manner similar to [14]:

$$J(\pi) = \mathbb{E}_{\substack{(q_s,q_g)\sim P(q_s,q_g) \\ \mathcal{W}\sim P(\mathcal{W})}} \left[\sum_{t=1}^{T} \mathbb{E}_{P(s_{t+1}|s_t,\pi(s_t))} \left[c_{collision}\left(s_t, \pi\left(s_t\right)\right)\right]\right] \tag{1}$$

$$\pi^* = \arg\min_{\pi\in\Pi} J(\pi) \tag{2}$$

Algorithm 2 LaSD

Initialize replay memory/dataset aggregator $\mathcal{D}$ to capacity N
Initialize policy to uniform random: $\hat{\pi}_1 = \pi_{random}, \beta_1 = 1$
Define *Approx Oracle*'s parameters: n_{free}, n_{labels}
Initialize π^*_{AO} = `AlmostOracle`(n_{free}, n_{labels})
for episode $i = 1$ to M **do**
  Sample random workspace environment, $\mathcal{W} \sim P(\mathcal{W})$
  Sample start and goal states, $(q_{start}, q_{goal}) \sim P(q_{start}, q_{goal})$
  Decay β_i
  Set current mixture policy, $\pi_i = \beta_i \pi^*_{AO} + (1 - \beta_i)\pi_i$
  Initialize planning tree, $\mathcal{T}_1 = \mathcal{T}$.init($q_{start}$)
  for $k = 1$ to T **do**
    Get sample from current policy, $q_{sample} \leftarrow \pi_i(\mathcal{T}_k, \mathcal{W})$
    Extend current tree with sample, `Extend`($\mathcal{T}_k, q_{sample}$)
    Invoke *AO*, $\mathbf{q}^*_\mathbf{k} = \pi^*_{AO}$.`GetLabelSamples`($\mathcal{T}_k, \mathcal{W}, q_{goal}$)
    Add to replay memory, $\mathcal{D} = \mathcal{D} \cup (\{\mathcal{T}_k, \mathcal{W}\}, \mathbf{q}^*_\mathbf{k})$
    Sample minibatch, $\phi_j = (\{\mathcal{T}_j, \mathcal{W}\}, q^*_j)$ from $\mathcal{D}$
    Train policy π_i on ϕ_j
return Best $\hat{\pi}_i$ on validation dataset.

C. LaSD: Learning Adaptive Sampling Distributions

After formalizing the problem, it is easy to see that one may use an appropriate model-free RL algorithm like DDPG [15]. However, such algorithms are known to be highly sample-inefficient. To address this, previous work has shown that given an oracle, such algorithms can be reduced to iterative supervised learning [8, 16]. This then begets two questions which we address in the next two subsections: what is the oracle for our problem setting, and if there is one, how can we imitate it.

Algorithm 3 Approx Oracle

Initialize : n_{free}, n_{labels}
procedure GETLABELSAMPLES($\mathcal{T}, \mathcal{W}, q_{goal}$)
  Initialize `priority_queue, label_samples` to empty
  while $i < n_{free}$ **do**
    $q_{rand} \leftarrow$ `RandomConfig()`
    if $q_{rand} \in \mathcal{C}_{free}$ **then**
      $q_{nearest} \leftarrow$ `Nearest`$(q, \mathcal{T})$
      $q_{new} \leftarrow$ `Steer`$(q_{nearest}, q_{rand}, \epsilon)$
      if `ObstacleFree`$(q_{nearest}, q_{new})$ & `GotSolvedWithinBudget(RRT(`q_{rand}, q_{goal}`))` **then**
        $C_{collision}$ = `RRT.get_last_cost()`
        `priority_queue.push(`$q_{rand}, C_{collision}$`)`
        i++
  $j = 0$
  while $j < n_{labels}$ **do**
    `label_samples.push(priority_queue.top())`
    `priority_queue.pop()`
    j++
  return `label_samples`

1) Defining an Oracle: To preserve the nature of the stochastic searched in randomized sampling based algorithm, our oracle should return labels which are multi-modal in nature, thereby spanning promising regions of $\mathcal{C}$. We now briefly highlight two relevant previous works. SAIL (Search as Imitation Learning) [14] learn heuristics for search based planners, by learning a deterministic policy over the space of lists to decide which node to expand, by *imitating Q-values* and arg-min regression. However, in our case, we need to learn stochastic policy imitating multimodal oracle samples over a continuous action space, a problem to which the arg-min framework of [14] does not scale. Another relevant work is [4], who learn a static distribution for single-query sampling based planning algorithms. However, they get oracle samples by using the resulting plans of optimal planners. We argue that this does not leverage the powerful randomized feature of sampling based planning, and sampling the optimal path directly can be infeasible in practice, especially for environments with high clutter.

For our problem, an oracle can be obtained by a visibility graph over the configuration space. However, the building the visibility graph in a high dimensional configuration space is extremely expensive. Instead, we define an oracle which returns *approximately* optimal samples, given a partially solved planning problem, in the form of the instantaneous tree and workspace environment $\{\mathcal{T}, \mathcal{W}\}$.

This *Approx Oracle (AO)* is defined in Alg. 3, and leverages the planning algorithm itself by forward simulating it. First, it finds n_{free} number of candidate samples which have valid, collision free extension to the given $\mathcal{T}$, which can also solve the forward problem within a fixed time budget, t_{AO}. It maintains these candidate samples in a priority queue, where they are ordered by the number of collision checks it takes the planning algorithm to solve the forward problem: reaching q_{goal}. *AO* then returns a list of the top n_{label} label sample configurations $\mathbf{q}^*$, which took minimum time while solving the forward problem. Fig. 1 shows label samples for an example helicopter planning problems. This is a top down view of a hilly environment (green denotes regions with high elevation). The start state is midpoint of bottom edge of image, while the goal state is midpoint of top edge. We can see as the tree (in blue) changes, the samples (in white) change their modes accordingly.

It is worth noting that [17] (Section 8.3) suggest that the approach taken by SAIL [14] can be extended to learn sampling distributions by using a backward tree as an oracle, however we note that the problem is not symmetric. Obtaining label samples by forward simulating the partially solved tree would be more accurate than choosing them from a backward tree constructed from q_{goal} to q_{start}.

2) Imitating the Oracle: We learn this policy by aggregating the dataset in an online fashion and decaying a mixture policy following Dataset Aggregation [8], as shown in Alg. 2. We combine the idea of dataset aggregation with experience replay [7] by maintaining a rolling memory buffer of state-action tuples from π^*_{AO}, and random sampling minibatches from it to train our policy to decorrelate the chronological memory. The state-action tuples in our case is the instantaneous planning tree, the occupancy grid of the environment, and the corresponding target sample location.

We now discuss how we encode the tree and workspace in our CVAE.

3) Encoding the Tree and Workspace: We follow and extend the framework proposed by [4] and use a conditional variational autoencoder [6] to learn a non-stationary policy that can map from planner state to samples, as shown in Fig. 3. Here, the planner state is both the environment $\mathcal{W}$ and the instantaneous search tree $\mathcal{T}_t$. For $\mathcal{W}$, we use a 2D or a 3D convolutional encoder, where we feed in a binary occupancy grid of the appropriate size.

The next issue lies in encoding the burgeoning tree which has a variable number of vertices and edges. This is an open problem being actively worked upon [18–23]. We use an architecture inspired by GraphRNN [23] which learn generative models of graphs. Keeping this in mind, we feed the tree as a sequence of edges into what we call an *Edge-RNN*. We represent each node with its n-dimensional state vector. Then we couple each child node with its parent node, as a $2 \times n$ array (each node has only a single parent as it is a tree). We feed these edges in the order in which the child nodes were added to the tree, as visualized in Fig. 2. We use Gated Recurrent Units [24] with a fixed-size memory vector, we take the output from the RNN as the encoded graph feature vector after feeding in all the child-parent edge pairs. For the root node, we mimic the values for its parent. This dense graph feature vector is concatenated with the workspace feature vector from the convolutional workspace encoder, followed by a dense layer to get the latent space of the CVAE.

Note: (1) We suspect that tree-LSTMs from the NLP literature [25], which use LSTM modules chained recursively in a tree-like topology, when constructed in a breadth-wise fashion, would perform better than our current approach. (2) Currently, we use only the node's state values as the feature

vector. However, these node features can be supplemented by goal-driven heuristics to improve the sampling distribution.

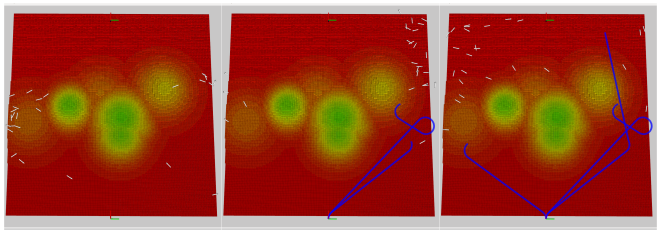

Fig. 1: *Approximate Oracle:* The start state is the center bottom of the image, while the goal is at the top. The approximately correct label sample for the instantaneous planning tree (shown in blue) are visualized as arrows. Arrow heads depict the of the state.

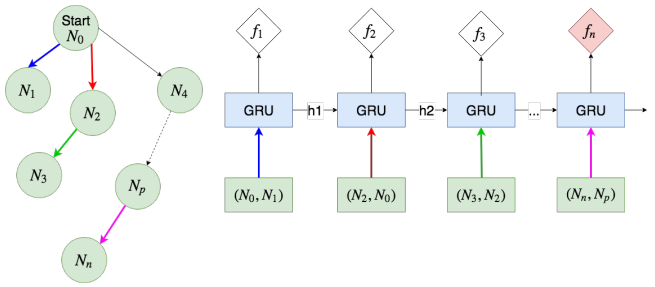

Fig. 2: Encoding graph edges into RNN network. On the left is the planning graph. The figure on the right demonstrates how edges are feed into a RNN network which encodes our graph into the final feature vector f_n

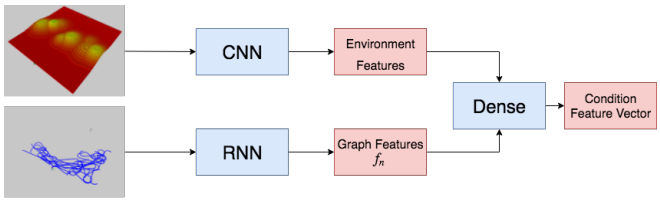

Fig. 3: Model used to featurize the conditioning variables for the CVAE.

III. Experiments and Results

We conduct experiments on datasets from [14, 17]:

1) 2D Holonomic: We validate our method on the dataset provided by [14], which consists of multiple 2D environments like single and multiple bugtraps, forest, mazes, etc. Qualitative results can be inspected in Figure 4 and in videos available here.

2) 4D Kinodynamic Helicopter: We test our framework on a helicopter problem with glideslope and curvature constraints in the dynamics, as outlined in [26]. The state space is given by translation and heading: $\{x, y, z, \psi\}$. Here, we use two types of procedurally generated environments drawn from some parameterized distributions each: canyons and hills. We use the environment generation code provided by [17].

We use [4] as a baseline, by only using the convolution workspace encoder. And similarly, we introduce another baseline by using only the Edge-RNN encoder. The results on test data are shown in Table I. All evaluation metrics are obtained by averaging the result of running each baseline 5 times on a fixed test set of 100 randomly generated environments, after training each baseline on a training dataset of 100 environments. While we are only explicitly optimizing for the number of collision checks, we also evaluate the final path length and the number of samples used to obtain a feasible path.

Conditioning on only $\mathcal{T}$ reduces the number of collision checks, while conditioning on only on $\mathcal{W}$ as done by [4] reduces the overall path length. Using both $\mathcal{T}$ and $\mathcal{W}$ further reduces the number of collision checks as well as path length compared as compared to uniform sampling. !

IV. Supplementary Material

Animated versions of the figures in this document are available here.

References

[1] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *Ieee access*, vol. 2, pp. 56–77, 2014. I

[2] S. Choudhury, "Adaptive motion planning," 2018. I

[3] M. Zucker, J. Kuffner, and J. A. Bagnell, "Adaptive workspace biasing for sampling-based planners," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, 2008, pp. 3757–3762. I

[4] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," *arXiv preprint arXiv:1709.05448*, 2017. I, II-C.1, II-C.3, III-.2, I

[5] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998. I, II-A

[6] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," in *Advances in Neural Information Processing Systems*, 2015, pp. 3483–3491. I, II-C.3

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015. I, II-C.2

[8] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 627–635. I, II-C, II-C.2

[9] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011. I, II-A

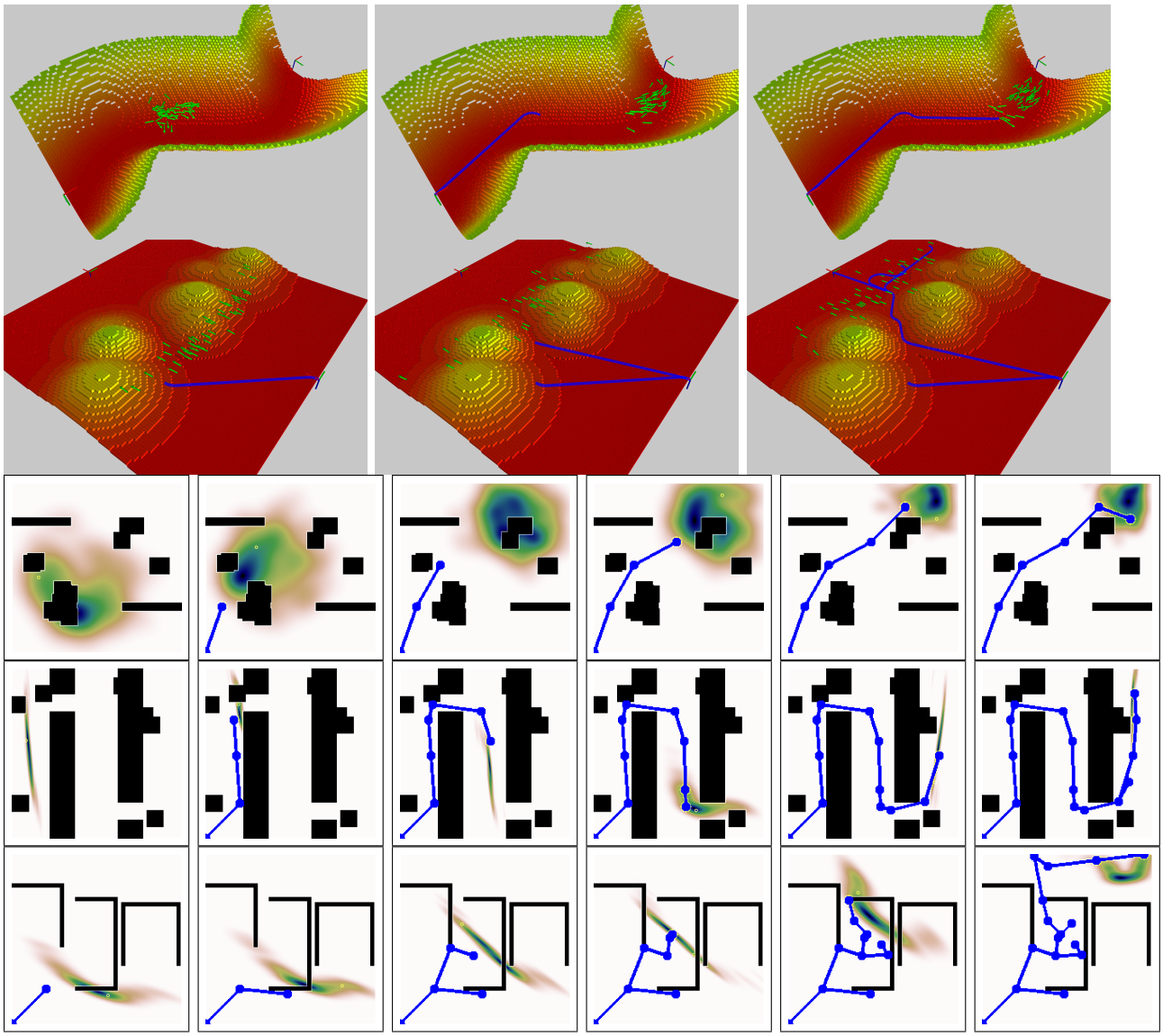

Fig. 4: Qualitative results of our learned distribution. Top two rows show the result on a helicopter planning problem, trained on a dataset of random canyons(Row 1) and hills (Row 2). Tree is shown in blue. The 50 random samples generated by our policy are shown as green arrows. The goal state is shown at the end of the canyon in Row 1, and in Row 2, the goal state is on the midpoint of the opposite edge of the start state. The bottom three rows shows qualitative results on the dataset generated by [14]. Start point is always on bottom left of image, and goal state is on top right.

TABLE I: Comparison of Conditioning on Different Aspects of Planning Problem vs Uniform Sampling for Kinodynamic Helicopter

Env	Metric	Graph and Environment	Graph Only	Environment Only [4]	Uniform Sampling
Hills	Collision Checks	**996.0 ± 1128.7**	1257.01 ± 874.7	2846.25 ± 4146	2627.76 ± 2080.0
	Solution Length	**350.4± 85.45**	384.8 ± 110.3	361.4 ± 77.9	436.3 ± 131.6
	Number of Samples	50.5 ± 17.9	**40.6 ± 47.1**	87.6 ± 180.1	66.5 ± 73.7
Canyon	Collision Checks	**801.6 ± 646.7**	1125.66 ± 1125.7	1807.18 ± 3144.6	1809.15 ± 1445.9
	Solution Length	323.4 ± 58.9	337.348 ± 74.3	**299.856 ± 62.0**	365.4 ± 90.6
	Number of Samples	49.5 ± 11.9	**31.7 ± 30.1**	68.5 ± 16.9	67.7 ± 64.9

[10] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 3067–3074.

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

[12] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2. IEEE, 2000, pp. 995–1001. II-A

[13] R. Allen and M. Pavone, "Toward a real-time framework for solving the kinodynamic motion planning problem," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 928–934. II-B

[14] M. Bhardwaj, S. Choudhury, and S. Scherer, "Learning heuristic search via imitation," *arXiv preprint arXiv:1707.03034*, 2017. II-B, II-C.1, III, III-.1, 4

[15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015. II-C

[16] S. Ross and J. A. Bagnell, "Reinforcement and imitation learning via interactive no-regret learning," *arXiv preprint arXiv:1406.5979*, 2014. II-C

[17] S. Choudhury, M. Bhardwaj, S. Arora, A. Kapoor, G. Ranade, S. Scherer, and D. Dey, "Data-driven planning via imitation learning," *arXiv preprint arXiv:1711.06391*, 2017. II-C.1, III, III-.2

[18] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016. II-C.3

[19] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[20] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.

[21] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *International conference on machine learning*, 2016, pp. 2014–2023.

[22] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *Proc. CVPR*, vol. 1, no. 2, 2017, p. 3.

[23] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, "Graphrnn: A deep generative model for graphs," *arXiv preprint arXiv:1802.08773*, 2018. II-C.3

[24] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014. II-C.3

[25] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," *arXiv preprint arXiv:1503.00075*, 2015. II-C.3

[26] S. Choudhury, S. Arora, and S. Scherer, "The planner ensemble and trajectory executive: A high performance motion planning system with guaranteed safety," in *AHS 70th Annual Forum, Montreal, Quebec, Canada*, vol. 1, no. 2, 2014, pp. 3–1. III-.2