2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Learning an Optimal Sampling Distribution for Efficient Motion Planning

Richard Cheng[1], Krishna Shankar[2], and Joel W. Burdick[1]

[1]California Institute of Technology
[2]Toyota Research Institute

***Abstract*— Sampling-based motion planners (SBMP) are commonly used to generate motion plans by incrementally constructing a search tree through a robot's configuration space. For high degree-of-freedom systems, sampling is often done in a lower-dimensional space, with a steering function responsible for local planning in the higher-dimensional configuration space. However, for highly-redundant sytems with complex kinematics, this approach is problematic due to the high computational cost of evaluating the steering function, especially in cluttered environments. Therefore, having an efficient, informed sampler becomes critical to online robot operation. In this study, we develop a learning-based approach with policy improvement to compute an optimal sampling distribution for use in SBMPs. Motivated by the challenge of whole-body planning for a 31 degree-of-freedom mobile robot built by the Toyota Research Institute, we combine our learning-based approach with classical graph-search to obtain a constrained sampling distribution. Over multiple learning iterations, the algorithm learns a probability distribution weighting areas of low-cost and high probability of success, which a graph search algorithm then uses to obtain an optimal sampling distribution for the robot. On challenging motion planning tasks for the robot, we observe significant computational speed-up, fewer edge evaluations, and more efficient paths with minimal computational overhead. We show the efficacy of our approach with a number of experiments in whole-body motion planning.**

Fig. 1: The general purpose mobile manipulation robot (described in [4]), which consists of a wheeled chassis, torso, left arm, right arm, and head, with a total of 31 DoF. The robot is shown here retrieving an item from a refrigerator, in a home.

I. Introduction

Sampling-based motion planners (SBMP) have been very successful in robotic systems, because they can provide collision free paths in complex environments. SBMPs (e.g. Probabilistic Roadmaps, RRT, etc.) operate directly on the continuous domain, and randomly sample points to find a path from start to goal [1]. These search algorithms are probabilistically complete (i.e. probability of successfully finding a path approaches one as number of samples approaches infinity – assuming a solution exists). However, the computational complexity of these algorithms grows greatly with the planning dimension, and problems become intractable for highly redundant systems with many degrees-of-freedom (DoF). Therefore, efficiently planning motion remains a challenging problem for high degree-of-freedom, highly redundant mobile robots.

For such robots, a standard approach is to plan the end-effector trajectory in lower-dimensional Cartesian end-effector space, and use an optimization-based dynamic planner as a steering function between sampled points [2]–[4]. In this way, the start/end position of the end-effector can be easily specified, and much of the work for the SBMP is offloaded to a local optimizer [5]. However, this also means that *edge evaluation* (planning between two sampled points through a steering function) in the SBMP becomes less predictable, and significantly more expensive; in most cases it is already the most expensive part of the motion planning process [6]. Thus, random sampling will make the planning problem intractable, and it is important to sample *intelligently and efficiently*.

In this work, we are motivated by whole-body motion planning for the highly redundant, 31 DoF robot shown in Fig. 1 [4]. To improve motion planning, we develop an algorithm that learns an optimal sampling distribution for an SBMP through policy improvement. This allows the robot to leverage prior experiences to guide sampling, such that improved search trees (e.g. RRT) can be constructed efficiently for path planning. Since we assume an optimization-based planner as the steering function in our SBMP, we consider the case where edge evaluation is extremely expensive and the shortest path (in Cartesian distance) is not necessarily optimal or feasible.

The contributions of this work include:

- Developing a learning algorithm that provides an implicit sampling distribution for efficiently sampling

978-1-7281-6211-9/20/$31.00 ©2020 IEEE

good, low-cost points,
- Proving convergence of the algorithm to a locally optimal sampling distribution through policy improvement as more training instances are gathered,
- Achieving significantly lower computation time and joint motion when utilizing the learned distribution on a 31 DoF, highly-redundant mobile robot.

We will refer to the implicit sampling distribution as the *heuristic*, and will use the two terms interchangeably. The learned heuristic is easily interpretable as it gives an intuitive mapping from regions in Cartesian space to path probabilities.

The paper is structured as follows. Section III examines our framework for learning a baseline heuristic and implementing policy improvement on that heuristic. Section IV demonstrates our approach on a 2D navigation problem. Section V then details how we combine our learned heuristic with an A* planner for utilization on the mobile robot.

II. Related Work

Graph search algorithms (e.g. A*) are guaranteed to find an optimal path from start to goal, relative to the edge weights in the graph, with a chosen discretization (assuming an admissible heuristic) [7]. However, the performance is limited by the discretization, *and* obtaining the right edge weights may be unintuitive depending on the search space. Some works have looked at transferring concepts from graph search algorithms to sampling-based motion planners [8], [9].

SBMPs operate directly on the continuous domain and find a path from start to goal by randomly sampling points and constructing a search tree. Certain sampling-based planners (e.g. RRT*, BIT*) are asymptotically optimal, converging to the optimal solution (i.e. path) as the number of sampled points approach infinity [9], [10]. However, such convergence can be prohibitively slow, grows exponentially in the dimension, and depends heavily on the heuristic used to sample points [6]. The standard RRT uses a uniform sampling of the state space, while other works have used goal-biased heuristics and/or other sampling heuristics to improve solution quality and search speed [11]–[14]. Some works have shown efficient ways to improve paths by constraining the sampling space once an initial solution has been found [9], [15], and using local optimizers to guide edges [5]. Given the expensive local optimization in our problem, we require more carefully guided sampling, even before any initial solution has been found.

The importance of a good sampling distribution (i.e. heuristic) is well-known, and recent work has studied different approaches for learning one [16]–[19]. [20], [21]. [22] directly generate samples end-to-end using neural networks (or recurrent neural network [23]). [24] learns a low-dimensional latent space (with a collision-checker) for high-dimensional planning problems to improve planning efficiency. However, these works learn from an expert demonstrator without improving upon the demonstrations. Rather than focusing on learning a sampling function, [25]–[27] examine the problem of determining which edges to evaluate in the tree to avoid expensive edge evaluations.

To allow learning of more optimal sampling distributions, [28], [29] use policy-gradient based reinforcement learning (RL) to learn an implicit sampling distribution (i.e. whether to accept or reject a sample) to guide an RRT. [30] similarly uses policy-gradient RL, but uses it to learn a local controller and a distance (i.e. cost) function between nodes. While policy-gradient RL methods have successfully solved many problems by modeling the learning problem as a Markov Decision Process, they suffer from significant learning variability and instability due to high variance in estimated policy gradients [31]. We formulate learning as a simpler classification problem, giving us greater stability (and in this case interpretability) in learning.

It is also worth mentioning a separate class of motion planners based on the notion of trajectory optimization. These planners are optimization-based (rather than sampling based), and are able to obtain smooth, low-cost trajectories, but they similarly face issues of scalability and susceptibility to local minima [32]–[35].

III. Learning a Sampling Distribution

In this section, we describe how we learn an optimal sampling function through iterative policy improvement. In Section III.A, we examine how to obtain a baseline sampling distribution from a collection of observed (suboptimal) paths. This is in the same vein as [20], though we learn an implicit sampling distribution rather than explicitly generating samples. In Section III.B, we describe the iterative policy improvement step, outline the learning algorithm, and prove that it converges to a locally optimal sampling distribution.

A. Supervised Learning of Baseline Sampling Distribution

Since the performance of sampling-based motion planners depends heavily on the sampling of the search space, our goal is to learn an implicit sampling distribution based on prior (simulated or real) experiences.

To do this, we first discretize the search space into voxels, each voxel holding a value between $(0, 1]$ representing the probability of that voxel being along a successful path. This provides us with a baseline heuristic (probability of accepting a sample within each voxel) to use for RRT exploration.

The heuristic is represented by a learned neural-network function $f_\theta : S \rightarrow A_{samp}$, which induces a distribution over paths. The input $s \in S$ represents the important aspects of the robot state and environment, and the "action" $a_{samp} \in A_{samp}$ is an m x m x m grid with each value representing the probability of accepting a sample from the given voxel. Our algorithm learns the function $a_{samp} = f_\theta(s)$ from the aggregated map data, which gives us our sampling heuristic for the RRT. Let us also define $a \in A$ (distinct from $a_{samp} \in A_{samp}$) which represents a continuous path from start to goal. Then we can say that the heuristic $a_{samp} = f_\theta(s)$ *induces a distribution over paths*, $\pi(a|s)$, from start to goal.

We use sigmoid cross-entropy loss to learn the sampling heuristic f_θ from our training dataset, which consists of RRT

paths (sampling in Cartesian space, and steering using the algorithm described in [4], [36]) in randomized representative environments. Minimizing this loss is equivalent to minimizing the $KL-$divergence between our probabilistic sampling distribution, and the paths in the training set.

Intuitively, the heuristic looks at the paths that have succeeded previously to inform it on the probability of new paths to succeed. Therefore, it improves RRT search speeds by guiding sampling towards high-success regions. However, this heuristic does not take into account the cost of the paths – only their validity – and does not contain a mechanism for policy improvement. We will refer to this sampling distribution as the *baseline heuristic*.

B. Policy Improvement to Optimize Sampling Distribution

Suppose we obtain the baseline sampling distribution $\pi(a|s)$ (arising from $a_{samp} = f(s)$), which gives a distribution over paths, trained on successful paths it has seen (with no regard for the cost/reward of those paths). Let $h(r(a)|s)$ be any bounded function that is an increasing function of reward. Then we can prove the following lemma.

Lemma 1.

$$J(s) = \int_a r(a)\pi(a|s,\theta)\delta a \;\leq\; \int_a r(a)\frac{\pi(a|s,\theta)\cdot h(r(a)|s)}{Z(s)}\delta a \tag{1}$$

where $Z(s)$ is a normalization factor such that $\frac{\pi(a|s,\theta)\cdot h(r(a)|s)}{Z(s)}$ is a probability distribution.

Proof: We use the following inequality, which holds if f and g are bounded, non-decreasing functions of the random variable r.

$$\mathbb{E}[f(r)g(r)] \geq \mathbb{E}[f(r)]\mathbb{E}[g(r)] \tag{2}$$

The policy $\pi(a|s)$ gives a distribution over paths, and there is a direct mapping from these paths, a to reward, r. Let $f := I$ be the identity function, and as discussed above, let $g := \frac{h(r(a)|s)}{Z(s)}$ be an increasing function of r. Then given the distribution $\pi(a|s)$, we have the following,

$$\begin{aligned}
\mathbb{E}_\pi[f(r)] &= \mathbb{E}_\pi[r] = \int r(a)\ \pi(a|s)\delta a \\
\mathbb{E}_\pi[g(r)] &= \int \frac{h(r(a)|s)}{Z(s)}\pi(a|s)\delta a = 1 \\
\mathbb{E}_\pi[f(r)g(r)] &= \int r(a)\ \frac{h(r(a)|s)}{Z(s)}\pi(a|s)\delta a
\end{aligned} \tag{3}$$

From Equation (2), it directly follows that the following inequality holds,

$$\int r(a)\ \frac{h(r(a)|s)}{Z(s)}\pi(a|s)\delta a \geq \int r(a)\ \pi(a|s)\delta a \tag{4}$$

□

Thus, given a baseline policy, we can increase the expected reward of the policy by modifying the distribution using the function h. We cannot say anything about how much we increase the expected reward. However, we can iteratively update our sampling distribution over learning episodes. Algorithm 1 illustrates how we can leverage $h(r|s)$ to learn an improving policy. In the algorithm, M, K are hyperparameters of the algorithm, N is the number of training episodes, and R is the sampled reward grid (i.e. the reward associated with each voxel). Note that we cannot just continually apply h *without running the RRT* to gather new training paths, because the new sampling distribution might not give us *valid* paths from start to goal.

Algorithm 1 Learning Heuristic with Policy Improvement

1: Import heuristic $f_0 : S \to A_{samp}$, which induces a path distribution $\pi_0(a|s)$
2: Collect the K datasets (i.e. groups of training examples) used to train the heuristic
3: **for** i **do** = 1,..,N
4: Use sampling heuristic $\frac{\pi_{i-1}(a|s)\cdot h(a|s)}{Z(s)}$ to run M iterations of the RRT
5: Store estimate of h in state-reward grid pairs (s, R)
6: Store the M state-path pairs (s, a) in a new group of training examples
7: Discard the oldest group of training examples (i.e. keep rolling record of improving paths)
8: Train heuristic $\pi_i(a|s)$ on last K training examples
9: **end for**

Theorem 1. *Given a bounded reward function, Algorithm 1 converges to a locally optimal sampling distribution.*

Proof: It follows directly from Equation 1 that $\mathbb{E}_{\pi_i}[r|s] \geq \mathbb{E}_{\pi_{i-1}}[r|s]$, since the former is trained also on paths from the modified distribution. By the Monotone Convergence theorem, if a monotone sequence of real numbers is bounded, then it has a finite limit. $\mathbb{E}[r|s]$ is clearly bounded, since we have defined the reward to be bounded. Therefore, we are guaranteed to converge to the following limit $\lim_{i\to\infty}\mathbb{E}_{\pi_i}[r|s] = r_{opt}(s)$.

□

Furthermore, since π_i is trained on only *successful* paths ρ_i from start to goal, it gives a distribution over successful paths. We will refer to the learned sampling distribution after policy improvement as the *reinforced heuristic*.

IV. Sampling Distribution in 2-Dimensional Domain

We first illustrate the method using a 2-dimensional problem where a point robot (no dynamics) uses an RRT to find a path from start to goal in a randomly generated obstacle map. In addition, we define a randomly generated, static cost map that induces a cost over every path.

The state s is an n x n occupancy grid capturing the obstacle map with the start/goal position annotated. The sampling heuristic f_{samp} is a ResNet-18 convolutional neural network [37] that takes this occupancy grid as its input, and outputs a probability heatmap over an m x m grid. The value of each voxel in the heatmap represents the probability of accepting a sample from that voxel.

Figure 2 shows the learned baseline heuristic, which is trained on paths generated by an RRT over several randomly

generated obstacle maps. The baseline heuristic encourages sampling in unobstructed regions along the path from start to goal. However, there is no sense of "cost" in this setting.

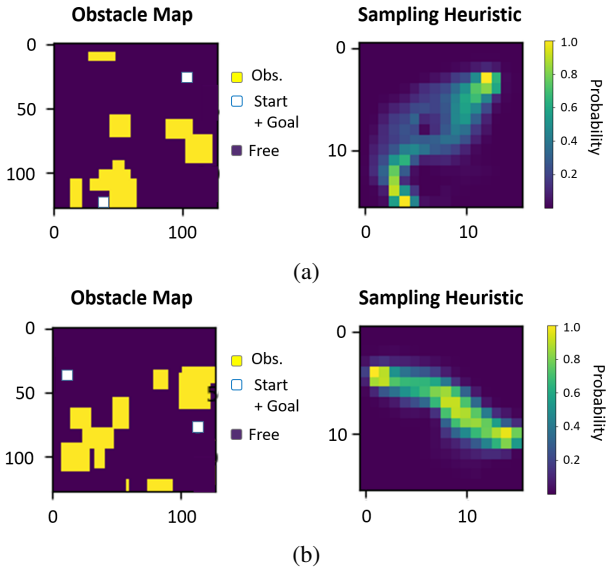

Fig. 2: Baseline sampling heuristic on two different example obstacle maps. The heuristic encourages sampling in unobstructed regions from the start to goal.

A sense of cost is introduced through the "Cost Map", as shown in Figure 3. Regions in yellow in the cost map are regions of high cost that the planner should learn to avoid, as described in the previous section. Algorithm 1 is run to continually improve the baseline heuristic such that it incorporates this cost landscape. Over time, the reinforced heuristic is successful in encouraging the motion planner to avoid high-cost regions.

It should be noted that by thresholding the sampling probability to a value greater than 0, we maintain the completeness property of the RRT. Therefore, in cases where the sampling distribution is not effective in providing a good bias towards low-cost, valid paths, we can still ensure that we will find a path in these cases (as the number of sampled points approaches infinity).

To provide a quantitative analysis, the table below shows the performance of the RRT using (a) uniform random sampling with goal bias, (b) the learned baseline heuristic (Section III.A), or (c) the reinforced heuristic (Section III.B). Results were averaged over 1000 trials in different randomized environments with randomized start/goal positions. For fair comparison, the same 1000 environments and start/goal positions were used for testing each of the three heuristics.

We note that the path cost was lowest using the reinforced heuristic, while the baseline heuristic and no heuristic cases performed similarly (this is expected since the baseline heuristic does not consider the cost landscape in its exploration). More importantly, we observe that using the reinforced heuristic or baseline heuristic leads to significantly fewer sampled points (and therefore fewer edge evaluations) before reaching the goal. Therefore, using the reinforced

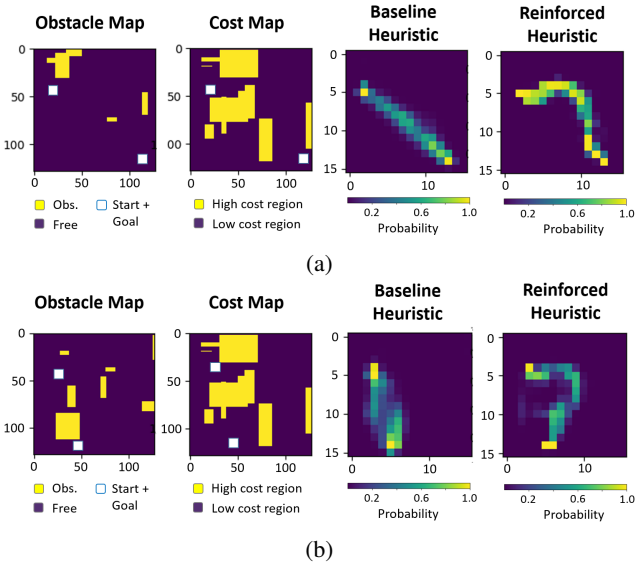

Fig. 3: Reinforced sampling heuristic (right) vs baseline sampling heuristic (second from right) after training with cost information (second from left). The cost map penalizes traversing certain regions with a high cost, and imposes a low cost on other regions. The reinforced sampling heuristic not only avoids obstacles, but encourages the motion planner to avoid high-cost regions of the map when doing so.

	RRT w/ goal bias	Baseline Heuristic	Reinforced Heuristic
Path Cost	1670	1640	**1540**
Edge Evals	173	118	**111**
Edge Evals (normalized)	0.85	0.62	**0.59**

TABLE I: Mean statistics of the RRT's performance over 1000 trials in different randomized environments. For fair comparison, each of the 3 methods was run across the same 1000 randomized environments. (Top) Average cost of each path found by the RRT. (Middle) Average edge evaluations until a path was found, (Bottom) Average edge evaluations, with each trial normalized to the max number of evaluations across the 3 methods (value of 1.0 implies that method had the most edge evaluations every time).

heuristic described in Section III.B gives us, on average, lower-cost paths with fewer edge evaluations compared to the baseline learned heuristic or no heuristic (with goal bias).

V. Learned Sampling Distribution on Robotic Platform

The system that motivates this study is more complex than that described in section IV – the 31 degree-of-freedom, highly-redundant two-armed robot shown in Figure 1 [4]. The objective is to plan whole-body motion of the robot to move the end-effector from a start to goal position specified

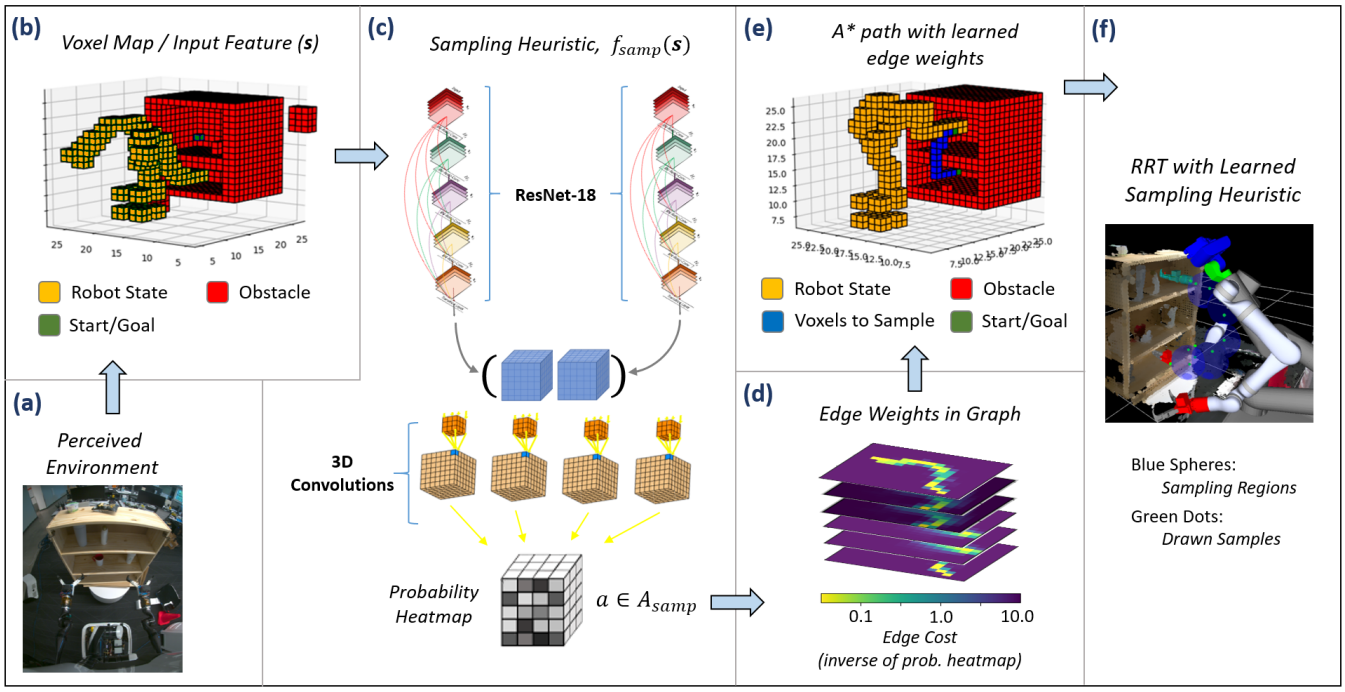

Fig. 4: Pipeline used to generate sampling heuristic. *(a)* The robot perceives the environment and constructs a voxel map. *(b)* Voxel map and sensors are used to get the robot state. *(c)* Robot state is fed through the trained neural network (described in Section V.*A*) to obtain the learned heuristic (i.e. probability heatmap) in the robot's end-effector space. *(d-e)* To further boost sample efficiency, the inverse of these probabilities are used as edge costs in a graph of the discretized end-effector space, and the A* search algorithm finds the lowest cost path in the graph from start to goal (i.e. blue voxels). *(f)* The RRT draws points from these sampling regions, with the green dots showing sampled points along a successful path.

in Cartesian, 6-DoF space. While planning the end-effector position, we must account for the kinematics/dynamics of the robot. The optimality/feasibility of a given path depends on not just distance in the end-effector space, but the path (arc-length) traveled by all 31 joints. Therefore, the objective is to learn a sampling distribution in the 3D end-effector space that implicitly accounts for the kinematics of the full robot as well as the motion of all its joints.

We utilize a sampling-based planning algorithm (QP-RRT) to plan a valid path for the end-effector *in 3D Cartesian space* from a start position to goal position [4]. Each edge between two points of the RRT is evaluated using a quadratic program (QP), which solves for the robot's inverse kinematics to move the end-effector with collision checking [36]. Once the RRT returns a valid path, the plan is executed by running the QP on each of the edges of the final path.

Because of the large number of collisions that must be checked and the high redundancy of the robot, each edge evaluation of the RRT is expensive (can take $\approx 0.5s$ for a single edge evaluation in cluttered environments). Therefore **intelligent sampling is crucial to minimize the number of edge evaluations** required to go from start to goal, since practically the RRT should use no more than 60 edge evaluations in its search.

The learned sampling distribution is not only necessary to efficiently find successful paths from start to goal, but also encourages greater optimality of discovered paths. Note that optimal SBMP algorithms (e.g. RRT*, BIT*) rely on additional edge evaluations to rewire the search graph. Because these edge evaluations are expensive, they further aggravate the existing issues of computation time and highlight the desire for optimal samples that will minimize joint motion.

A. Problem Setup

We define the input space $S := \mathbb{R}^{64\text{x}64\text{x}64\text{x}2}$, which represents a 3D voxel map with 2 channels. One channel encodes the state of the robot and the start/goal position of the end-effector, and the other channel represents an occupancy grid of the surrounding environment. Each input voxel measures 4cm^3. The output space is defined as $\mathcal{A}_{samp} := \mathbb{R}^{10\text{x}10\text{x}10}$, representing a voxel map that encodes the probability of a good solution path passing through each voxel. Each output voxel measures 16cm, for a total workspace measuring 1.6m. The goal of the trained neural network is to learn a mapping $f_{samp} : S \rightarrow \mathcal{A}_{samp}$ that will allow us to identify promising regions of the action space to sample.

Neural Network Architecture: The neural network f_{samp} takes the two channels of the 3D voxel map (each of dimension $\mathbb{R}^{64\text{x}64\text{x}64}$) and feeds each into a ResNet-18 network with ReLu activation functions [37] (stride lengths were reduced to obtain desired output dimensions). The output is two 3D feature maps, each of size $\mathbb{R}^{16\text{x}16\text{x}256}$. A max pooling layer over the channel depth is added before the two channels are concatenated, resulting in a combined feature

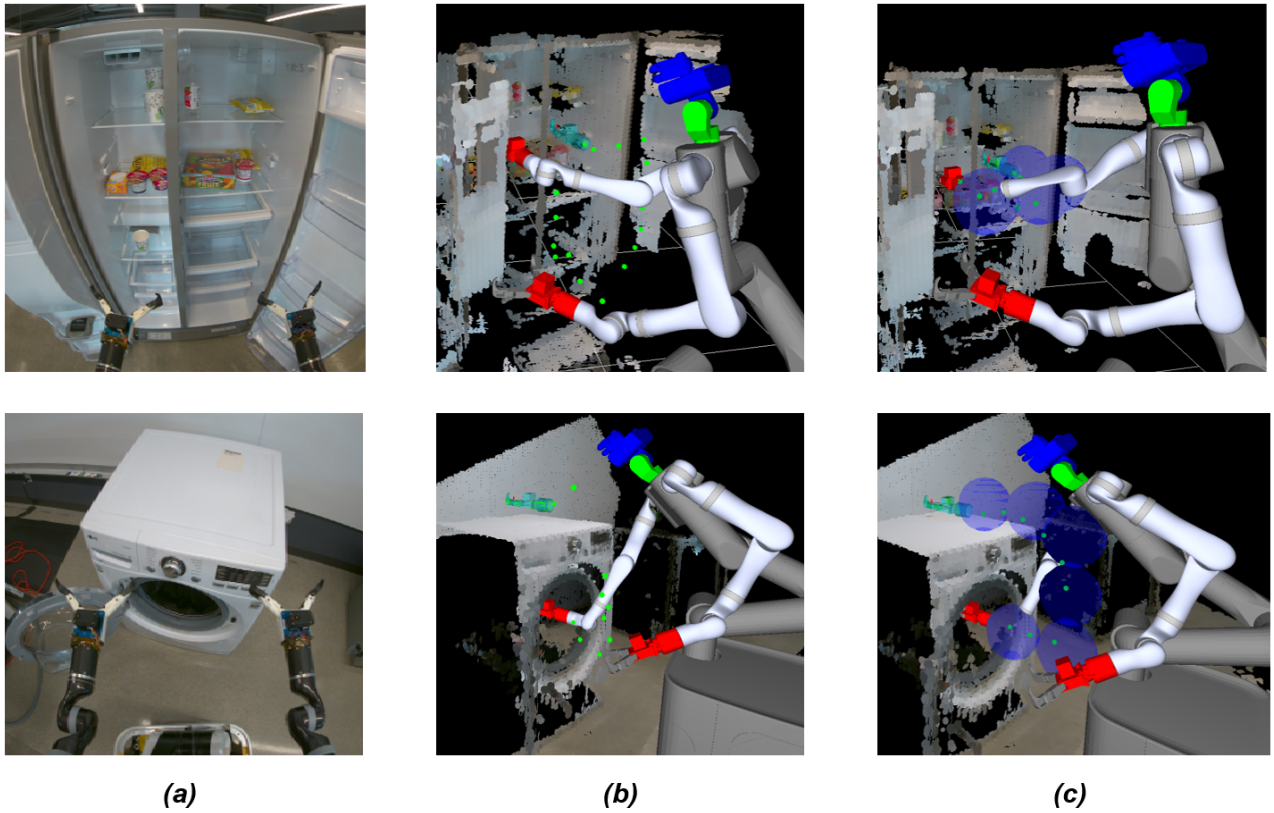

Fig. 5: **Visualization of the robot planning with and without the learned heuristic**. The top row shows visualizations for planning in the Refrigerator environment, while the bottom row shows visualizations for planning in the Laundry environment. *(a)* Image of the planning environment that the robot sees. *(b)* Path found by the RRT (green dots) without the learned heuristic. *(c)* Path found by the RRT with the learned heuristic. Green dots represent the RRT path, while blue spheres represent the regions that the RRT is constrained to sample within.

map of size $\mathbb{R}^{16\text{x}16\text{x}64\text{x}2}$. This feature map is then fed through four 3D convolutional layers with linear activation functions (i.e. pass-through). The first 3D convolutional layers utilizes a 5x5x5 kernel with a $[1, 1, 4]$ stride; the last three 3D convolutional layers utilize a 3x3x3 kernel with no edge padding. Therefore, the final convolutional layer provides the desired output $a \in \mathcal{A}_{samp} = \mathbb{R}^{10\text{x}10\text{x}10}$. Figure 4(c) provides a simplified illustration of this architecture. This network architecture enables fast inference over the high-dimensional state space (≈ 50 ms inference).

We gather training data by generating both purely random environments and random representative environments (e.g. random cabinets and tables), and then running the QP-RRT with the simulated robot on those environments. Note that time is *not* encoded in the output, so the resulting plan does not consider this.

Algorithm 2 describes the learning algorithm for the robot motion planning system. It differs from Algorithm 1 mainly in that it considers that we do *not* have access to $h(a|s)$ and therefore cannot leverage it to improve exploration. For the baseline heuristic, we set $j = 1$ and disregard r.

B. Combining Learned Probabilities with A* Search

Sampling directly using the learned probabilities allows us to maintain completeness and is ideal in many cases. However, because of the very limited sample budget in our target application, sampling directly from the learned distribution is *still* not efficient enough.

We must further narrow down the search space, which is achieved by using our learned sampling distribution (giving us probabilities for successful paths), and utilizing the inverse of the sampling probabilities as edge costs in a graph. Transitions to occupied voxels are given infinite cost in this graph. Once we construct a graph using edge weights from the neural network, we then run a graph search algorithm (i.e. A*) on the constructed graph. This gives us the highest probability voxel path from the start to goal. *Sampling from the voxels in this path gives us the highest probability of finding a low-cost path under our sampling heuristic.*

While restricting the sampling area of the RRT to these areas eliminates the completeness property of the RRT, it is a necessary trade-off to remain within our limited sample budget. The next subsection will show that this tradeoff leads to significantly better performance. Figure 4 summarizes the

Algorithm 2 Learning Heuristic for Robot Sampling

Import heuristic $f_0 : S \rightarrow A_{samp}$, which induces a path distribution $\pi_0(a|s)$
for i = 0,...,N **do**
  Initialize robot/environment and start/goal ($s_i \sim S$)
  Populate robot state in voxel map s_i
  Initialize RRT with start position x_0 of the end-effector
  for j = 1,...,M **do**
    while Solution not found **do**
      Sample point x_p using heuristic $f_i(s_i)$
      Look for solution from current tree τ to x_p by looking at closest K points ($y_{p,k}$) between τ and x_p and evaluating QP between them.
      Compute $r_{p,k} = \|\Delta\theta\|_2$
    end while
    Create output grid a_j^{samp} based on resulting path
    Compute reward of the resulting path, r_j^p
  end for
  Store the best state-output pairs (S, a_i^{samp}, r_i^p) based on the highest reward r^p.
  Train heuristic $f_i(a|s)$ on the last K stored records
end for

pipeline used to generate our sampling heuristic once it has been learned.

C. Experiments

To test the robot, we drive the robot to different household objects (e.g. desk, cabinet, etc...) extract the state of the robot and the voxel map [4], and construct the state representation to feed to the neural network. We then compute the sampling heuristic and simulate ten robot motion plans (i.e. running the QP-RRT ten times with and without the learned heuristic). Because of the random nature of the RRT, results will be different across trials. Statistics across ten trials for five different environments (50 trials total) are shown in the tables below. We see significant improvements in computation time and decrease in joint-space arc-length across scenarios.

Furthermore, arc-length and compute time statistics are shown across ten *successful* trials whereas failures (i.e. timeouts) are ignored in the statistics. These failures due to timeout are common when *not* using the learned heuristic as seen in the last table below, which shows the number of timeouts experienced before achieving 10 successful trials. Therefore, not only does the learned heuristic improve performance when comparing only successful cases, it also achieves significantly greater success rates than the original QP-RRT.

Note that improvements are smaller on the *Table* environment. This is because the table task has few obstacles, and it is much easier for the base RRT to plan in.

Figures 5 visualize the points in the RRT path both with and without the learned heuristic in two real environments. The top image in each figure shows the true environment, the middle image shows the baseline RRT-planned path (in green dots) without the learned heuristic, and the bottom image shows the RRT-planned path with the learned heuristic (blue spheres represent the sampling regions for the RRT). We see that without the heuristic, the RRT often takes odd paths because the search is more random through the end-effector space, whereas the heuristic constrains sampling along a more natural path from the start to goal. Note that the learned heuristic could similarly be used in conjunction with optimal variants that do some kind of pruning (e.g. RRT*), though this would impose *significant* cost in terms of additional edge evaluations.

Environment	Compute time (Learned Heuristic)	Compute time (No Heuristic)
Desk	$\mathbf{8.7 \pm 2.2}$ **s**	33.0 ± 9.0 s
Cabinet	$\mathbf{9.5 \pm 2.8}$ **s**	13.7 ± 6.8 s
Laundry	$\mathbf{8.9 \pm 2.6}$ **s**	34.5 ± 5.9 s
Refrigerator	$\mathbf{5.8 \pm 1.9}$ **s**	23.1 ± 8.0 s
Table	3.0 ± 1.7 s	4.0 ± 0.4 s

Environment	Arc-Length (Learned Heuristic)	Arc-Length (No Heuristic)
Desk	$\mathbf{12.4 \pm 1.1}$ **rad**	21.0 ± 4.6 rad
Cabinet	17.7 ± 2.7 rad	15.7 ± 2.0 rad
Laundry	$\mathbf{15.5 \pm 1.3}$ **rad**	20.6 ± 4.2 rad
Refrigerator	$\mathbf{11.5 \pm 1.7}$ **rad**	22.7 ± 3.6 rad
Table	$\mathbf{18.3 \pm 0.8}$ **rad**	23.8 ± 1.5 rad

Environment	Failures (Learned Heuristic)	Failures (No Heuristic)
Desk	**0**	9
Cabinet	**1**	4
Laundry	**1**	18
Refrigerator	**1**	12
Table	0	0

TABLE II: Performance statistics for the RRT, with and without the learned heuristic, across 10 repeated trials in 5 different environments. *(Top Table)* Mean±SD computation time before RRT returned a solution, given that it succeeded. *(Middle Table)* Average joint space path length of the returned solution. Denotes the total radians traveled by all the joints. *(Bottom Table)* Number of times the RRT failed before completing 10 successful trials.

VI. Conclusion

Whole-body motion planning is a challenging problem for high degree-of-freedom, redundant mobile manipulators in the real-world. Sampling-based motion planners with complex steering functions can often fail for such robots due to the expensive computations required for RRT edge evaluations. Therefore, it is important to efficiently utilize the limited number of nodes that can be explored. The learning-based sampling distribution presented here allows us to plan whole-body motions for a high degree-of-freedom mobile robot in challenging environments consistently with significantly lower computation time and minimal joint motion. Furthermore, the learned distribution is easily interpretable as it gives an intuitive mapping from regions in space to success probabilities. We intend to extend this work by exploring alternative input modalities (e.g. stereo imagery instead of voxel map), and closed-loop execution to handle dynamic obstacles.

References

[1] S. M. LaValle, *Planning algorithms*, 2006.

[2] B. W. Satzinger, J. I. Reid, M. Bajracharya, P. Hebert, and K. Byl, "More solutions means more problems: Resolving kinematic redundancy in robot locomotion on complex terrain," in *IEEE International Conference on Intelligent Robots and Systems*, 2014.

[3] P. Hebert, M. Bajracharya, J. Ma, N. Hudson, A. Aydemir, J. Reid, C. Bergh, J. Borders, M. Frost, M. Hagman, J. Leichty, P. Backes, B. Kennedy, P. Karplus, B. Satzinger, K. Byl, K. Shankar, and J. Burdick, "Mobile manipulation and mobility as manipulation - Design and algorithms of RoboSimian," *Journal of Field Robotics*, 2015.

[4] M. Bajracharya, J. Borders, D. Helmick, T. Kollar, M. Laskey, J. Leichty, U. Ma, Jeremy Nagarajan, A. Ochiai, J. Petersen, K. Shankar, K. Stone, and Y. Takaoka, "A Mobile Manipulation System for One-Shot Teaching of ComplexTasks in Homes," in *IEEE International Conference on Robotics and Automation*, 2020.

[5] S. Choudhury, J. D. Gammell, T. D. Barfoot, S. S. Srinivasa, and S. Scherer, "Regionally accelerated batch informed trees (RABIT): A framework to integrate local information into optimal path planning," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2016.

[6] K. Hauser, "Lazy collision checking in asymptotically-optimal motion planning," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2015.

[7] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, 1968.

[8] S. M. Persson and I. Sharf, "Sampling-based A algorithm for robot path-planning," *International Journal of Robotics Research*, 2014.

[9] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "BIT *: Batch Informed Trees for Optimal Sampling-based Planning via Dynamic Programming on Implicit Random Geometric Graphs," *CoRR*, 2015.

[10] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," in *International Journal of Robotics Research*, 2011.

[11] S. M. LaValle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," *In*, 1998.

[12] J. J. Kuffner and S. M. La Valle, "RRT-connect: an efficient approach to single-query path planning," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2000.

[13] C. Urmson and R. Simmons, "Approaches for Heuristically Biasing RRT Growth," in *IEEE International Conference on Intelligent Robots and Systems*, 2003.

[14] A. H. Qureshi, S. Mumtaz, K. F. Iqbal, B. Ali, Y. Ayaz, F. Ahmed, M. S. Muhammad, O. Hasan, W. Y. Kim, and M. Ra, "Adaptive Potential guided directional-RRT," in *2013 IEEE International Conference on Robotics and Biomimetics, ROBIO 2013*, 2013.

[15] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *IEEE International Conference on Intelligent Robots and Systems*, 2014.

[16] Y. Li, R. Cui, Z. Li, and D. Xu, "Neural Network Approximation Based Near-Optimal Motion Planning with Kinodynamic Constraints Using RRT," *IEEE Transactions on Industrial Electronics*, 2018.

[17] P. Lehner and A. Albu-Schaffer, "The Repetition Roadmap for Repetitive Constrained Motion Planning," *IEEE Robotics and Automation Letters*, 2018.

[18] M. Bhardwaj, S. Choudhury, and S. Scherer, "Learning heuristic search via imitation," in *International Conference on Robot Learning*, 2017.

[19] T. Takahashi, H. Sun, D. Tian, and Y. Wang, "Learning Heuristic Functions for Mobile Robot PathPlanning Using Deep Neural Networks," in *International Conference on Automated Planning and Scheduling*, 2019.

[20] A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip, "Motion planning networks," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2019.

[21] A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip, "Motion Planning Networks: Bridging the Gap Between Learning-based and Classical Motion Planners," *arXiv*, 2019.

[22] B. Ichter, J. Harrison, and M. Pavone, "Learning Sampling Distributions for Robot Motion Planning," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2018.

[23] M. J. Bency, A. H. Qureshi, and M. C. Yip, "Neural Path Planning: Fixed Time, Near-Optimal Path Generation via Oracle Imitation," 2020.

[24] B. Ichter and M. Pavone, "Robot Motion Planning in Learned Latent Spaces," *IEEE Robotics and Automation Letters*, 2019.

[25] C. M. Dellin and S. S. Srinivasa, "A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors," in *Proceedings International Conference on Automated Planning and Scheduling, ICAPS*, 2016.

[26] N. Haghtalab, S. Mackenzie, A. D. Procaccia, O. Salzman, and S. S. Srinivasa, "The provable virtue of laziness in motion planning," in *Proceedings International Conference on Automated Planning and Scheduling, ICAPS*, 2018.

[27] M. Bhardwaj, S. Choudhury, B. Boots, and S. S. Srinivasa, "Leveraging Experience in Lazy Search," *arXiv*, 2019.

[28] M. Zucker, J. Kuffner, and J. A. Bagnell, "Adaptive workspace biasing for sampling-based planners," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2008.

[29] C. Zhang, J. Huh, and D. D. Lee, "Learning Implicit Sampling Distributions for Motion Planning," in *IEEE International Conference on Intelligent Robots and Systems*, 2018.

[30] H. T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, "RL-RRT: Kinodynamic Motion Planning via Learning Reachability Estimators from RL Policies," *IEEE Robotics and Automation Letters*, 2019.

[31] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 2018.

[32] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," 2009.

[33] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2011.

[34] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "CHOMP: Covariant Hamiltonian optimization for motion planning," *International Journal of Robotics Research*, 2013.

[35] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *International Journal of Robotics Research*, 2014.

[36] K. Shankar, J. W. Burdick, and N. H. Hudson, "A quadratic programming approach to quasi-static whole-body manipulation," in *Springer Tracts in Advanced Robotics*, 2015.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016.