Learning Non-Stationary Sampling Distributions for Motion Planning

Brian Okorn, Rogerio Bonatti, Ratnesh Madaan, Sam Zeng
Robotics Institute
Carnegie Mellon University
Email: {bokorn, rbonatti, ratneshm, szeng}@andrew.cmu.edu

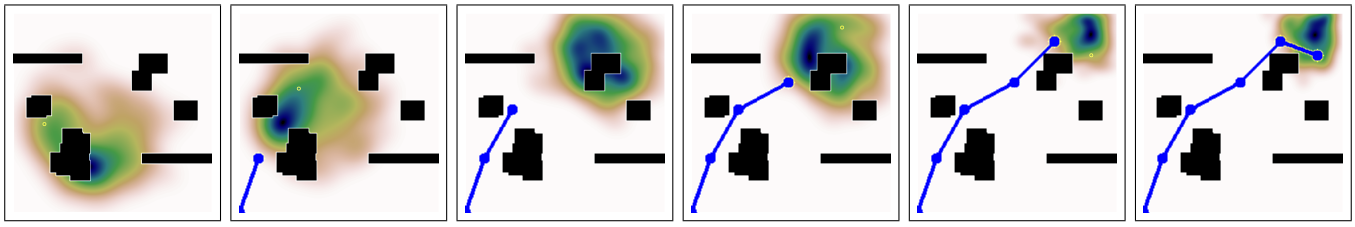


Fig. 1: RRT expansion based on learned sampling distribution. Start is the lower left corner and goal is upper right corner. Kernel density estimate fit to 200 samples with blue as high probability and pink as low. Next sample checked in yellow.

***Abstract*—Sampling based motion planning has been shown to be effective across various domain of problems, however the bottleneck of collision checks in high dimensional problems still remains. In this work, we propose to learn the distribution which is used by a planner to sample the configure space. Specifically, we extend previous work by using a conditional varitional auto-encoder to learn such a distribution in an online fashion, where the conditioning variables are both the instantaneous planning graph and the workspace. We demonstrate our method for a 2D holonomic point robot over a dataset of synthetic environment of multiple types, and use Rapidly exploring Random Trees (RRTs) as our path planner. Our results show more efficient sampling distributions and reach the goal using significantly fewer collision checks than vanilla RRT.**

I. Introduction

Sampling-based motion planning algorithms are powerful tools for solving high-dimensional problems quickly, and are also probabilistically complete, meaning that the algorithm will converge to a solution as the number of samples increases, if one exists. Sampling-based planners rely on an implicit representation of the configuration space, drawing samples from a distribution across the domain, and building a search tree that ultimately links the start and goal state.

Traditional sampling-based planners use a uniform distribution of samples in the configuration space. This approach is inefficient due to three main factors: (1) not all samples improve the search to closer to the goal state, (2) narrow gaps in the environment may require finer sampling granularity to be surpassed, and (3) next best possible samples are dependent on the nodes of the current search tree. Previous work [1, 2, 3] addressed the first two points, showing that planners can significantly improve performance by biasing the sampling distribution using features of the environment. However, these works use a supervised approach and generate a static sampling distribution given a planning problem, which is not changed as the planning graph or tree is being built, and are inherently limited in their capabilities to reach the goal as fast as possible. Intuitively, in a case where the environment is composed of several gaps in sequence, once the planner found a solution through a gap, the optimal next samples should be taken close to the next gap. Similarly, if a lot of collision checks are failing indicating a cluttered environment, the distribution should be conservative, sampling states close to the nodes of the graph, whereas if the majority of the collision checks pass, then the distribution can afford to take risky samples. Hence, there's a need for an online paradigm in the space of methods which seek to learn sampling distribution, and the current search graph is a key component of the same.

In this work, we propose conditioning the sampling distribution jointly on the current search graph, in addition to the the fully observable environment. Thereby, we implicitly capture the time dimension of the search process, adapting the sampling distribution as the search progresses. Our algorithm uses a Conditional Variational Autoencoders (CVAE) to learn, using various expert samples, a sampling distribution during training time, using which samples in the configuration space can be drawn from at test time. We apply our sampling-based algorithm to a variety of 2D planning environments.

We learn a distribution in a supervised fashion offline, and then improve upon it in an online fashion, using different candidate sampling distributions as experts.

II. Related work

The use of heuristics for classic deterministic planners such as A^* has been greatly explored in literature [4] and significantly improves planning time performance. Recent work from

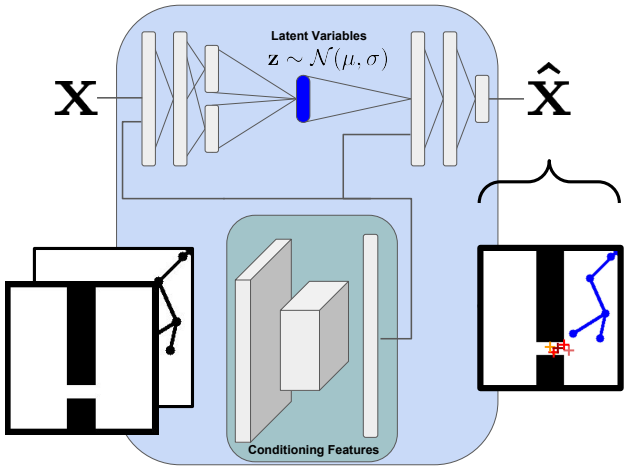


Fig. 3: Our Sampling Network architecture. Graph and Environment are encoded via the Conditioning Features, shown in green. The CVAE uses these features to replicate the precomputed high-utility samples $\mathbf{x}$, with its sample distribution $\hat{\mathbf{x}}$.

Bhardwaj et al. [5] explores use a clairvoyant oracle to learn heuristics for search-based planners based on features of the obstacle map. Simlar to our approach, a conditional variational autoencoder is used to mimic the oracles samples, conditioned on a flattened representation the environment, the start, and the goal. This lead to samples near the optimal distribution, but the samples might not actually progress the tree toward the goal in a timely fashion.

Similarly, we expect the use of rules to also improve performance of sampling-based algorithms. We can think of different objectives for biasing samples. For example, in the works of Arlsan [3] and Pan [6], the objective is to learn a sample rejection algorithm to learn collision free points in local searches, and at the same lower cost-to-go to the goal.

Closer to our approach, in the work of Zucker [2], the objective was to bias the sampling distribution using manually-defined features of the fully observable environment. This idea was used again by Ichter [1], who used a CVAE to learn sampling distributions given successful trajectories for a given environment.

One important issue when biasing distributions is how to analyze the utility of a given sample, and possible trade-offs between utility [7] and exploration of the environment. Suppose we have access to a clairvoyant oracle that knows the optimal path to a problem. If we trained the distribution bias only with examples from the optimal path, we would be maximizing utility, but at the same time we would not emphasize exploration. Exploration is particularly important for environments where the optimal solution lies in small gaps, hard to be sampled from. In a way, Ichter addresses this problem by alternating between completely random and biased samples.

In terms of biasing the sampling distribution based on current progress of the search tree, Gammell [8] uses the distance from start to goal on the current tree to limit searches to an ellipse from start to goal. In our work it is arguable that we aim to implicitly learn such a strategy based on the current search tree, but even for cases when we haven't reached the goal yet.

III. Approach

We introduce a method to learn sampling distribution for traditional sampling based motion planning algorithms, which is conditioned both on the environment or workspace, and the instantaneous planning graph. Specifically, we consider the motion planning problem from a start point to a goal region for a 2D holonomic point robot using the RRT algorithm. We use a conditional variational autoencoder (CVAE) [9] with the workspace and the planning graph as the conditioning variables. Both are represented as images, as depicted in Fig. 3 and are concatenated before feeding them into the CVAE, and the network is "sampled" at test time from inside the RRT algorithm. We now explain our modules one by one :

A. Rapidly-expanding Random Trees

RRTs [10] fall under the sampling based motion planning paradigm, where the key idea is to bypass the computationally intensive process of building a (high-dimensional) configuration space, by sampling random configurations and incrementally adding them to the current planning graph if they are valid - collision-free and kinematically or dynamically feasible. RRT is an efficient, incremental tree building algorithm devised to solve single query planning problems, and it works as follows. In the beginning, the start configuration as added to the tree. Next, a random sample is drawn and an attempt is made to connect to the current tree by finding the nearest node in the tree. If the connection is feasible, a new node is added according to a greediness or "growth-factor", ϵ. The previous two steps are repeated until the goal region is reached. This vanilla RRT algorithm can be expedited by multi-directional trees, biased sampling, changing the greediness, and also can be made optimal by local re-wiring of the tree as in RRT* [11]

B. Conditional Variational Autoencoder (CVAE)

Similar to Ichter et al. [1], we train a conditional variational autoencoder to mimic the sampling distribution of good samples. However, we conditioned on both the environment and as well as the current search tree. This lead to a non-stationary distribution that evolves over time, leading the search tree along the optimal path, generating the samples at the appropriate time with respect to the planner's progress. This makes sense because in order to sample a path from the start to the goal, the planner must not only sample a collision free series of points connecting the start to the goal, but it must also sample them in the correct order.

To produce the desired sample distribution, our CVAE is trained on high-utility samples, described in Section III-C.

At test time, the latent space is sampled and decoded in combination with the conditionally variables to produce high-utility sample. Our loss function is the standard combination of mean squared error as the reconstruction loss and KL-divergence as the distribution loss.

C. Data Generation

In order to generate sufficient and labeled training data, we adopted a sampling based approach to data generation. Specifically, in order to generate labels for the training data, we run RRT for a random number of iterations to generate a partially solved planning graph. From this graph, we examined a set of n valid future nodes, shown in red in Figure 4, and select a subset of of good nodes locations based on a utility function $g(G, X)$. The details of this are described in section III-D. This subset is used as the label for the next best sample given G and E. Then we can form a training dataset of (E, G, l_i) where l_i is a node location (x_i, y_i) from the subset of good future node positions which is used as the label. We repeat this over all k graphs for the environment and all environments in the dataset to form our total training dataset.

D. Sample Utility

Given an environment and the current planning graph, it is not entirely obvious as to which point should be sampled next and depends on whether we would like to sample as close to the optimal path as possible or to generate a valid path as fast as possible. In our initial work, we are focusing on the later case. As a result, we defined a heuristic, $g(G, X)$, for measuring the utility of adding a sample to the current graph. We chose a utility function based on the path length from a node on the current graph to the sample, and then from the sample to the goal. Specifically, we use the 8 connected distance of a path from the sample to the goal, shown varying from close (blue) to far (yellow) in Figure 4. Since all of the samples are pre-screened to be able to connect to the planning graph without collision, we do not worry about having to also check the distance from the graph to the sample as well. Figure 4 shows a visual representation of this process.

E. Online Learning

Initially, we train in a supervised fashion, using trees generated using RRT with a uniform distribution. However, this is not ideal as tree building is a sequential process and trees generated using solely our sampling policy vary greatly from the trees generated using a uniform distribution. Uniform distributions can differ from learned distributions in various ways such as significant branching shown in Figure 4. Additionally, in challenging environments, they might not be able to ever reach the goal in the training data as seen in the left figure in 5. As a result, the training data for this environment will be limited to areas around the start. Trees generated using the learned distribution, however, tend to progress further and be more linear as shown in the middle figure if 5. However, they fail to reach the goal since they are no longer similar to

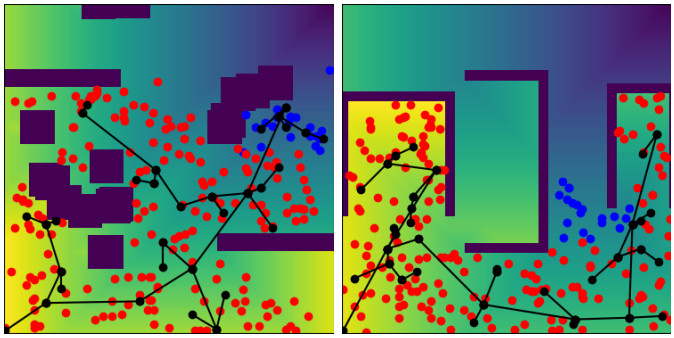


Fig. 4: Supervised samples, shown in red (top 20 in blue), expanded from graph, shown in black. The background color represents 8 point connected distance to the goal (blue is close, yellow is far)

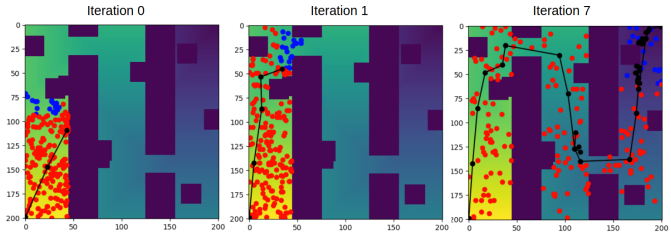


Fig. 5: Sample search tree generated using uniform distribution (left); Search tree after 100 epochs of supervised learning (middle); Search tree after 7 iterations of dataset aggregation (right).

the trees in the training dataset. We applied a similar technique to the methods found in DAgger [12] and inspired by the approach taken by Bhardwaj et al. [5] for heuristic learning we train later iterations on an aggregated dataset of trees generated using both uniform and learned distribution to produce the right most figure in 5.

IV. Experiments

A. 2D Holonomic Environment

Sampling distributions were learned using the planning datasets provided in Bhardwaj et al. [5]. This dataset consisted of eight environment types, including bugtraps (U-shape obstacles), forests, mazes, and gaps. A sampling distribution was learned from each of these environments using the top 20 the best sample, according to our utility function, of 100 uniformly generated samples.

The environment is represented as a 2D occupancy grid, allowing us to use a 2D convolutional neural network to featurize the environment information for conditioning the CVAE. Additionally, our configuration space is also two dimensional, allowing us to project the graph directly into the work space in the form of an image. This graph image and the occupancy grid are stacked and feed into a small convolutional network with two convolutions layers and a single fully connected

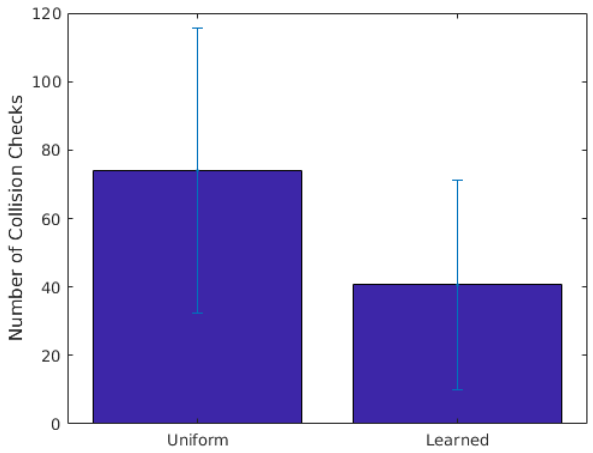


Fig. 6: Number of collision checks for RRT using uniform and learned distributions on the Forest environment.

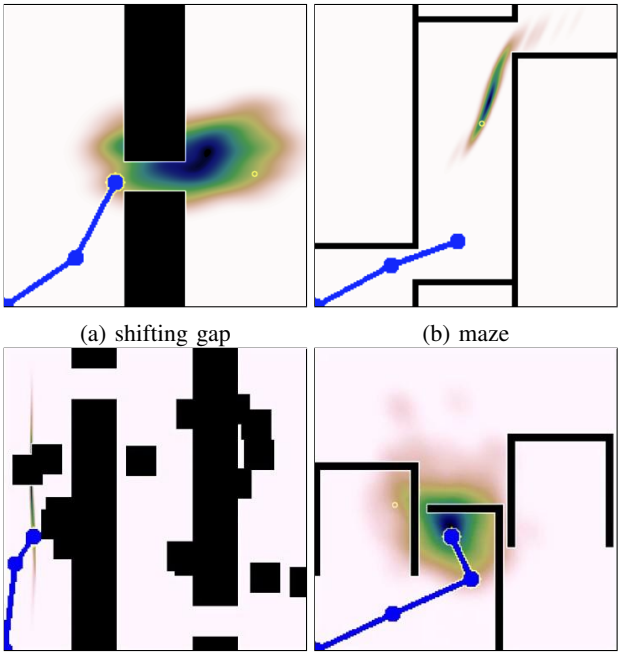
hidden layer, shown in green in Figure 3 to produce our CVAE's conditioning variables.

The results in each of the training environments can be seen in Figure 7. The distribution is visualized using a kernel density estimate fit to 200 random samples drawn from the learned distribution and example sample is shown as a yellow circle. The sequence of tree expansions using the learned distribution can be seen in Figure 1 for the Forest environment. Further sequences can be seen in Figures 10 and 11. We compared RRT using a uniform and the learned distribution on five instances of the forest environment, shown with error bars in Figure 6. The learned distribution found the goal using half the number of collision checks.

We tested the effect of training on a single best sample, like in Ichter et al. [1], verses a distribution of high utility samples. Since we are conditioning on the graph and the environment, there is only a one-to-one mapping between samples and conditioning variables, as opposed to many-to-one mapping you get when only conditioning on the environment (a sample for each vertex in the optimal path). This resulted in a extremely tight distribution that basically converges to a single point, seen on the left of Figure 8, where as using the top 20 produces a more varied distributions. While having an extremely tight distribution is useful when you perfectly select the optimal sample, it prevents the RRT from exploring when this sample is incorrectly predicted and may lead to the planned being unable to find the goal.

B. Implementation Details

We use Pytorch [13] to implement the CVAE. The image base feature network has a structure of C32-P2-C32-P2-FC1024 with relu activation functions and the CVAE has has two fully connected layers with 512 hidden units and latent dimension of 3. Open Motion Planning Library's [14] implementation of RRT is used to



(a) shifting gap (b) maze

(c) forest (d) multiple bugtraps

Fig. 7: Learned sampling distribution on variety of environments.

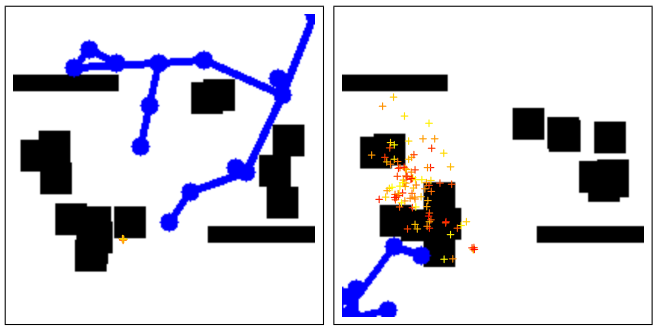


Fig. 8: Samples (red-yellow +) drawn from CVAE trained on single best sample, on left, and top 20 of 100 on right.

get the current planning graph and change the sample therein. Our code is available at https://github.com/madratman/learn2sample_motion_planning and https://github.com/madratman/ompl_learn2sample.

V. Future Work

A. Training on larger state-spaces: 5 DOF manipulator

We are currently setting up a 5 DOF manipulator environment on OMPL to test our approach on larger state-spaces than 2D, and also to test if our approach works well in environments where the configuration space differs from the workspace.

So far we have already implemented an environment for generating training data in which we create an arbitrarily large forward tree with N nodes (from start to goal), and then generate K collision-free samples with respect to the current

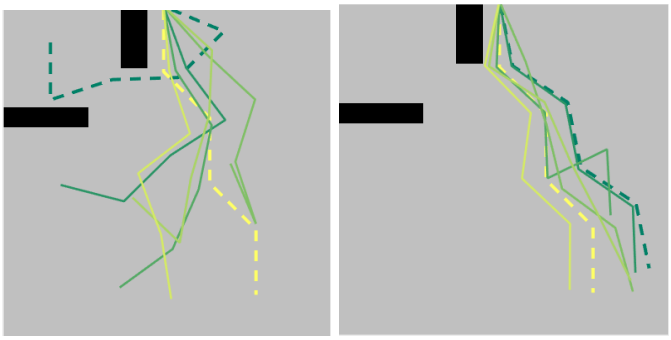


Fig. 9: Samples drawn from 5-DoF arm distribution. Start state is represented by the dashed yellow line and goal by the dashed green. The samples are colored by their distance to the goal (green is close, yellow is far), where the distance metric is calculated over the $SO(2)^5$ manifold.

tree, raking them according to their utility function. Figure 9 shows two examples of different samples ranked by utility (color variation), going from one start node to an end node (dashed lines).

The utility function described for the 2D point-robot problem, $g(G, X)$ is not applicable for RRT in higher dimensions, as we are not really operating on a grid, and our workspace differs from the configuration space. A utility function can be arbitrarily defined depending on the context of the planning problem. In our case, we decided to calculate our cost as the displacement of each joint, with equal weights. However, independently of how one decides to calculate costs, the ideal sample must be the one that minimizes the cost connecting the current graph with the goal configuration (so distance from start to closest node to the sample plus distance from node to sample, plus distance from sample to goal).

In an ideal training environment, if we have K samples, the best sample would be obtained by running an optimal planner on each sample until convergence, and ordering them by utility values. However, such an approach would prove too computationally costly in practice, in particular for high-dimensional environments. We opted therefore, for the following approach: (1) for a large amount of time (order of minutes), we run a planner such as RRT* forming a nearly optimal backwards tree from goal to start, (2) we find the nearest nodes to the current sample both in the forward and backwards trees, and (3) we sum the costs from start to sample on the forward tree and from sample to the goal via the backwards tree to obtain the utility function. This procedure assumes that the backwards tree is sufficiently dense across the configuration space, and greatly diminishes computational complexity. With varying greediness of these optimal samples. we expect the trained autoencoder model to take "risks" if for example, a lot of collision checks were free.

B. Graph convolutions

Our tests so far have been in simple, toy state spaces. When generating the conditioning variables in more complex state spaces, the graph will not adequately represented as a projection into a 2D image or 3D voxel representation. In these cases, the full graph is featurized using graph convolutions [15]. The graph nodes are described by their configuration space representation, as well as a their connectivity. Graph coarsening operations are used as pooling layers in graph space and allow the network to maintain a stable feature representation over different sizes of graph. The resulting feature representation will be combined with the environment description, a occupancy grid in either image or voxel map form, using convolutional layers, 2D or 3D respectively, and passed on the the variational autoencoder as the conditioning variable. A problem that will need to be addressed is the ordering of the vertices, while having no meaning to the graph, will have an effect on the linear layers of the featurizing network. This can be solved by randomizing this ordering or by using the

Graph convolutions require graphs of a fixed size. To facilitate this we "resize" the trees by padding trees that are to small with singleton vertices and use the graph pooling operation to shrink trees that are too large.

VI. Conclusion

Sampling based motion planning algorithms are highly effective as they do not rely on an explicit representation of configuration, but rather approximate it by sampling based motion planning has been shown to be effective across various domain of problems, however the bottleneck of collision checks in high dimensional problems still remains. In this work we learned the distribution which is used by a planner to sample the configure space. improving past work by using a conditional variational autoencoder to learn such a distribution in an online fashion, where the conditioning variables are both the instantaneous planning graph and the workspace. We demonstrated our method for a 2D holonomic point robot over a dataset of synthetic environment of multiple types, and use Rapidly exploring Random Trees (RRTs) as our path planner. Our results show more efficient sampling distributions and reach the goal using significantly fewer collision checks than vanilla RRT.

References

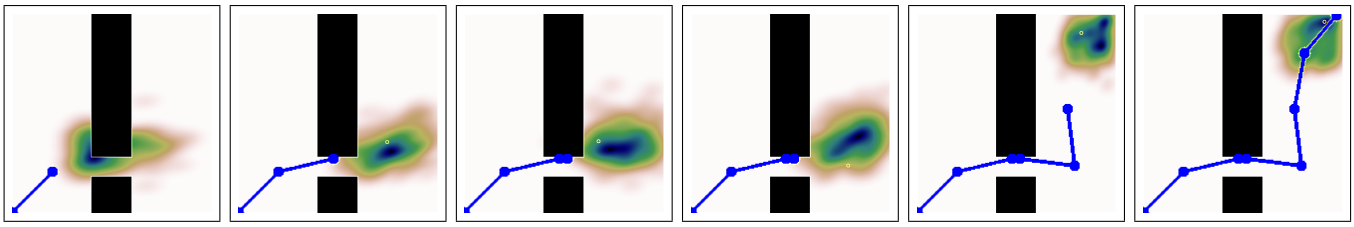
[1] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," *arXiv preprint arXiv:1709.05448*, 2017.

[2] 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*. IEEE, 2008, pp. 3757–3762.

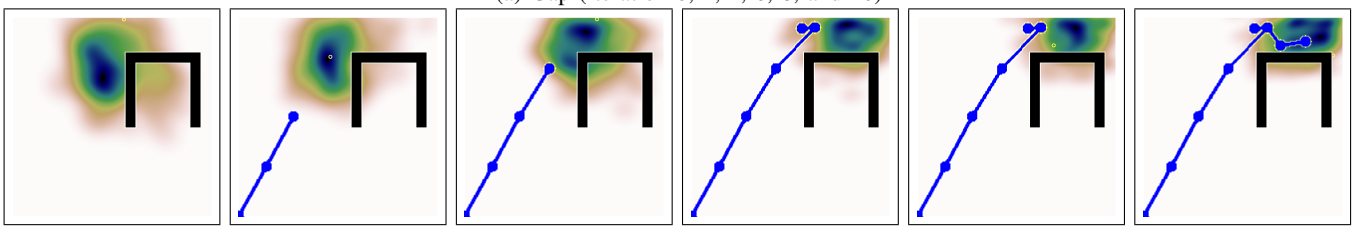
[3] O. Arslan and P. Tsiotras, "Dynamic programming guided exploration for sampling-based motion planning algorithms," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 4819–4826.

[4] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.
[5] M. Bhardwaj, S. Choudhury, and S. Scherer, "Learning heuristic search via imitation," *arXiv preprint arXiv:1707.03034*, 2017.
[6] J. Pan, S. Chitta, and D. Manocha, "Faster sample-based motion planning using instance-based learning," in *Algorithmic Foundations of Robotics X*. Springer, 2013, pp. 381–396.
[7] B. Burns and O. Brock, "Toward optimal configuration space sampling." in *Robotics: Science and Systems*, 2005, pp. 105–112.
[8] 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.
[9] 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.
[10] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.
[11] 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.
[12] S. Ross, G. J. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *International Conference on Artificial Intelligence and Statistics*, 2011, pp. 627–635.
[13] A. Paszke, S. Gross, S. Chintala, and G. Chanan, "Pytorch," https://github.com/pytorch/pytorch.
[14] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, http://ompl.kavrakilab.org.
[15] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," *CoRR*, vol. abs/1606.09375, 2016. [Online]. Available: http://arxiv.org/abs/1606.09375

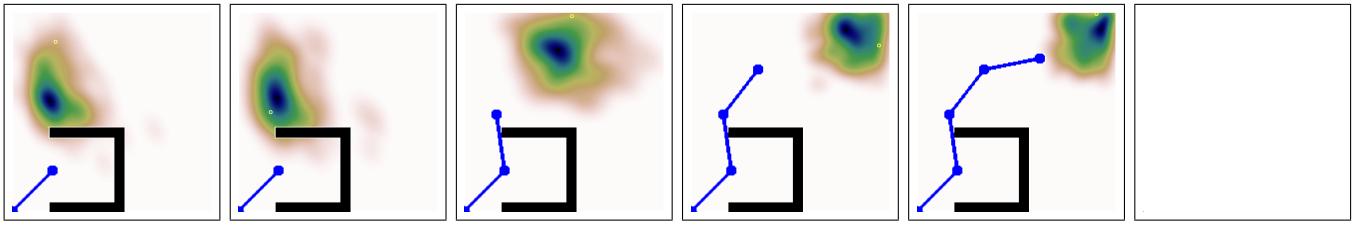
APPENDIX



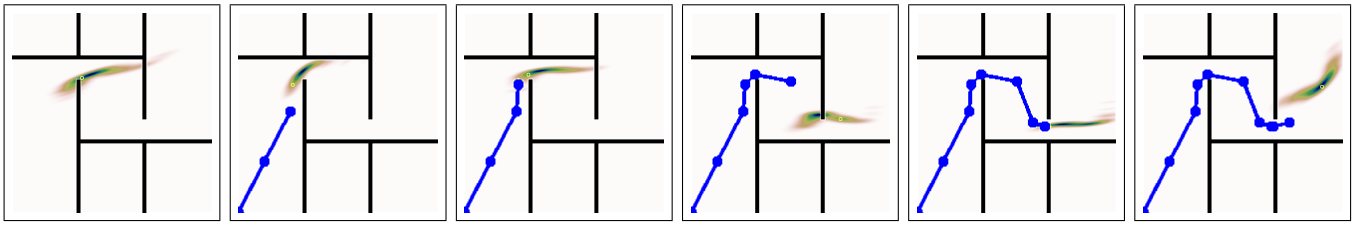
(a) Gap (Iteration 0, 2, 4, 6, 8, and 10)



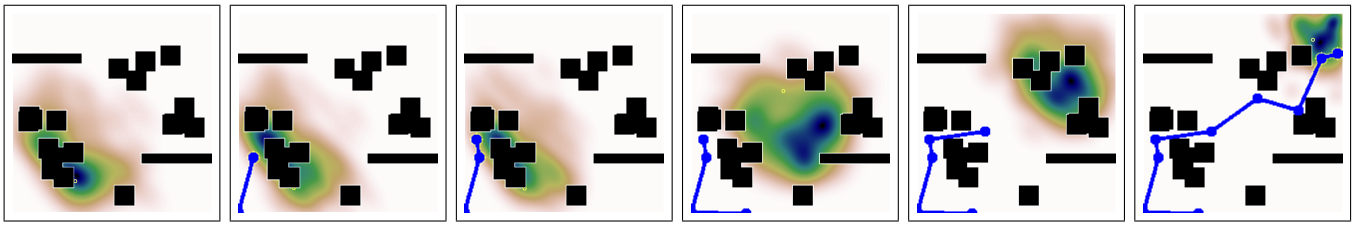
(b) Bugtrap 1 (Iteration 0, 2, 3, 5, 6, and 8)



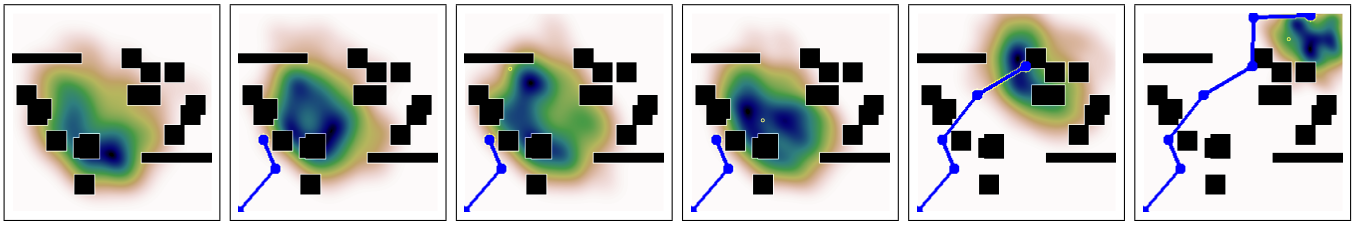
(c) Bugtrap 2 (Iteration 0, 1, 2, 3, and 4)



(d) Maze (Iteration 0, 3, 6, 9, 12, and 14)

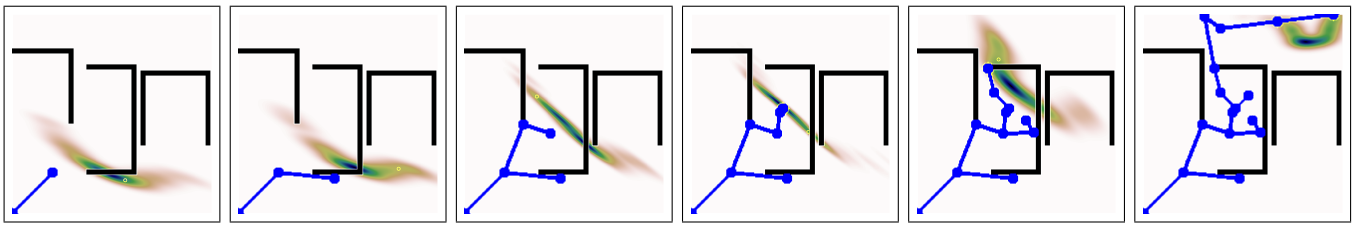


(e) Forest 1 (Iteration 0, 6, 12, 18, 24, and 29)

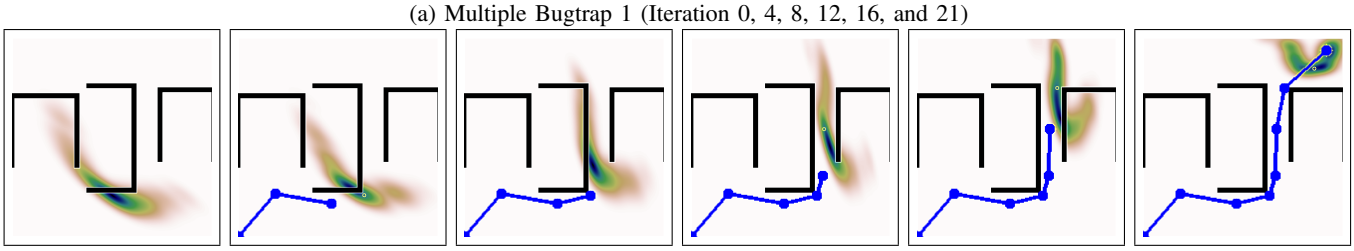


(f) Forest 2 (Iteration 0, 3, 6, 9, 12, and 16)

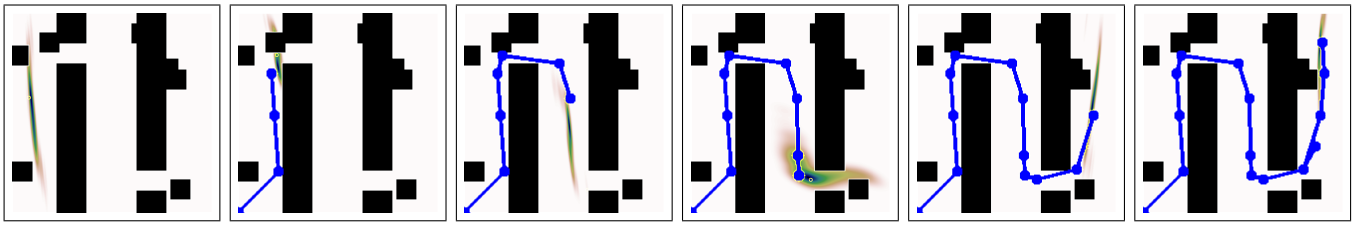
Fig. 10: Learned sampling distribution on variety of environments. KDE fit to 200 samples (blue high, red low). Next sample checked in yellow.



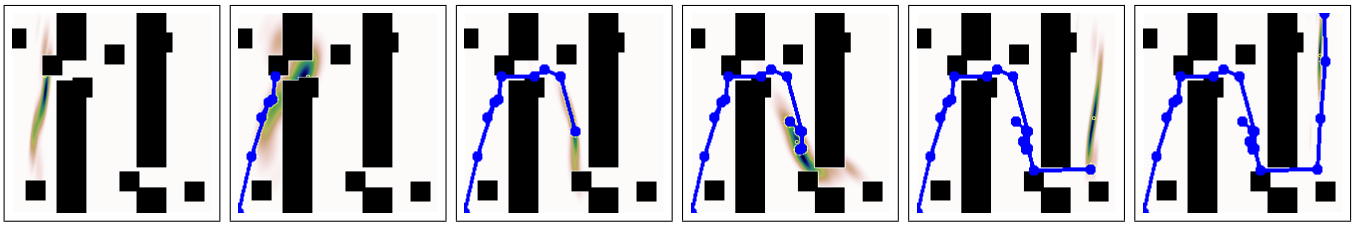
(a) Multiple Bugtrap 1 (Iteration 0, 4, 8, 12, 16, and 21)



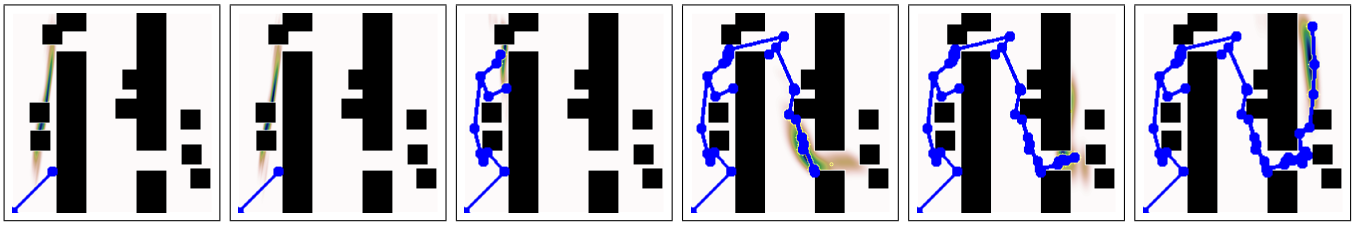
(b) Multiple Bugtrap 2 (Iteration 0, 2, 3, 5, 6, and 8)



(c) Gap Forest 1 (Iteration 0, 3, 6, 9, 12, and 15)



(d) Gap Forest 2 (Iteration 0, 5, 10, 15, 20, and 23)



(e) Gap Forest 3 (Iteration 0, 17, 34, 51, 68, and 84)

Fig. 11: Learned sampling distribution on more complicated environments. KDE fit to 200 samples (blue high, red low). Next sample checked in yellow.