Learning and Using Abstractions for Robot Planning

Naman Shah,[1] **Abhyudaya Srinet,** [1] **Siddharth Srivastava** [1]

[1] Autonomous Agents and Intelligent Robots Lab,
School of Computing, Informatics, and Decision Systems Engineering,
Arizona State University,
Tempe, AZ, USA
namanshah@asu.edu, asrinet1@asu.edu, siddharths@asu.edu

Abstract

Robot motion planning involves computing a sequence of valid robot configurations that take the robot from its initial state to a goal state. Solving a motion planning problem optimally using analytical methods is proven to be *PSPACE-Hard*. Sampling-based approaches have tried to approximate the optimal solution efficiently. Generally, sampling-based planners use uniform samplers to cover the entire state space. In this paper, we propose a deep-learning-based framework that identifies robot configurations in the environment that are important to solve the given motion planning problem. These states are used to bias the sampling distribution in order to reduce the planning time. Our approach works with a unified network and generates domain-dependent network parameters based on the environment and the robot. We evaluate our approach with *Learn and Link planner* in three different settings. Results show significant improvement in motion planning times when compared with current sampling-based motion planners.

1 INTRODUCTION

Robot motion planning (MP) aims to compute a collision-free sequence of robot configurations that take the robot from an initial configuration to an intended goal configuration. The simplest motion planning problems are also proven to be *PSPACE-Hard* (Reif 1979; Canny and Reif 1987). Sampling-based motion planning techniques such as rapidly-exploring random trees (RRT) (LaValle 1998), probabilistic road maps (PRM) (Kavraki et al. 1996), fast matching tree (FMT) (Janson and Pavone 2013), batch informed trees (BIT*) (Menglu Lan et al. 2016) efficiently compute an approximate to the optimal solution. Such planners are known to be *probabilistic complete* i.e, if the solution exists, the probability of finding a solution reaches 1.0 as the number of samples approaches infinity. Most sampling-based motion planners (SBMPs) use uniform sampling to cover the entire state space of the planning task. Such methods of sampling fail to efficiently provide samples from cluttered or narrow regions. For example, consider the environments shown in Fig. 1. Green and blue regions in the images show initial and goal locations respectively. The robot has to pass through the narrow channel to reach the goal state. In such scenarios, SBMPs equipped with uniform samplers would struggle to generate samples from the narrow channel due to
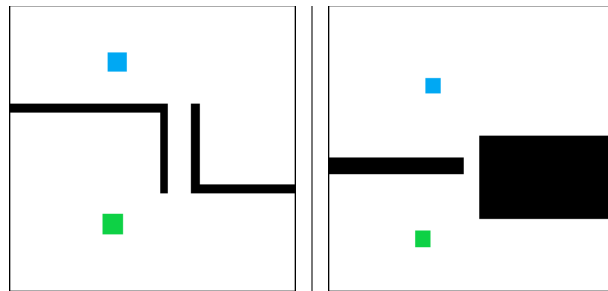

Figure 1: The green area represents the initial location of the robot and the blue area represents the goal location. Black regions show obstacles. The robot has to pass through the narrow passage to reach the goal location. Uniform samplers struggle in such scenarios due to reduced probability of locations in the narrow passage.

reduced probability. The problem becomes more challenging when the robot has a higher degree of freedom (e.g. a hinge joint or an arm manipulator) due to the dimensionality of the configuration space (*C-space*).

This paper focuses on speeding up sampling-based motion planning by identifying regions that are important to solve the given problem. These regions (sets of robot configurations) or configurations are the ones that the robot must achieve to solve the posed motion planning problem. Such regions are analogous to the concept of *landmarks* in automated task planning. Molina, Kumar, and Srivastava (2020) defines such sets as *critical regions* and use them to improve efficiency of path planning problems in *2D spaces*. Such critical regions can also be used to facilitate a hierarchical algorithm that first computes a path over the identified critical regions and use SBMP over these regions. In this paper, we propose a deep-learning-based framework that identifies such critical regions for any given arbitrary motion planning problem and uses them to bias the sampling distribution for an SBMP. With this biased distribution, the sampling-based motion planning algorithms are more likely to sample states from these critical regions and that should lead to reduced planning times.

In this paper, the proposed approach uses demonstrations to learn *critical regions* for the given motion planning task. Our approach utilizes a unified network. It uses the *C-space* to compute network architecture and domain-dependent pa-
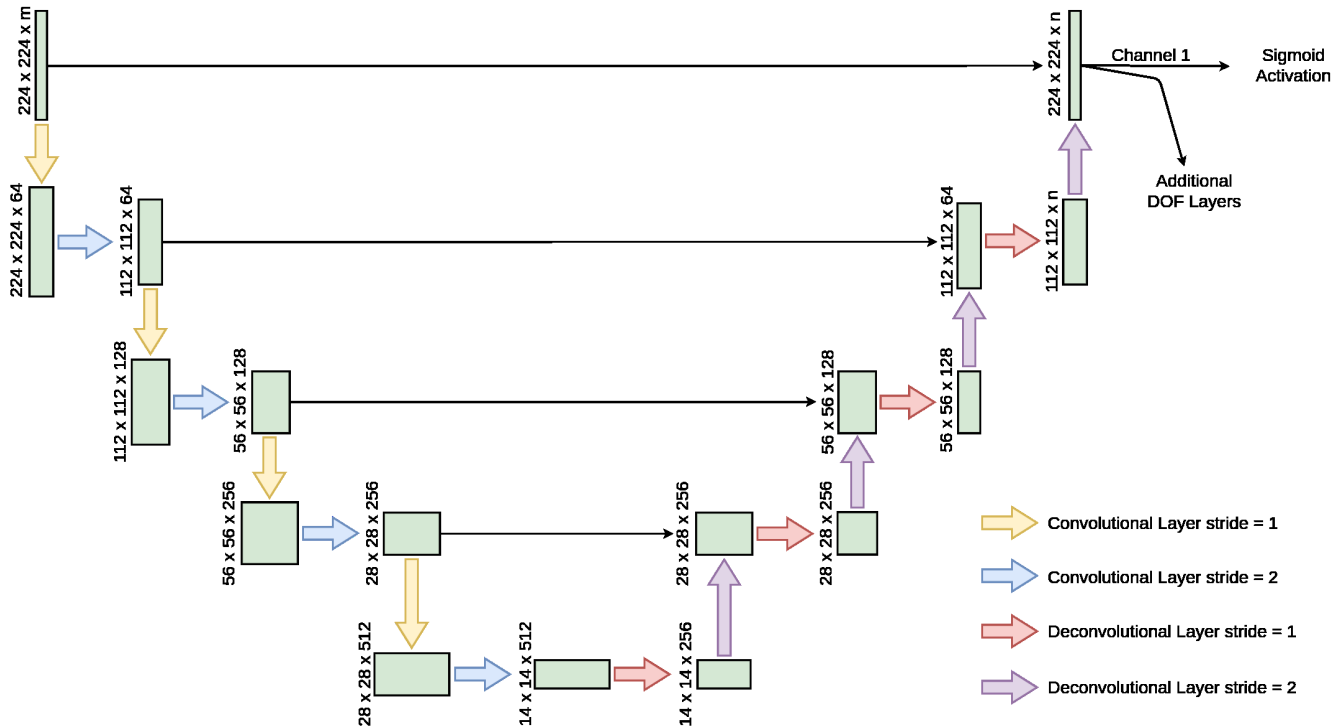
Figure 2: Network Architecture. Number of input channels m and number of output channels n depends on the environment and the robot.

rameters of the network and utilize it to learn the distribution of critical regions for the given *C-space*. We evaluate our approach in three different domains with three different robots. Our empirical evaluation shows that the model is able to capture important information such as environment and robot geometry, critical states, and goal states. Results show significantly reduced planning times compared to recently developed SBMPs that use uniform sampling.

To summarize the main contribution of our approach, our approach is able to use the number of degrees of freedom of the robot and the dimensionality of *C-space* to generate a robot-specific network architecture to learn the distribution of critical configurations. It uses the learned model to predict critical regions for the new motion planning problem, which is then used with a sampling-based motion planner to efficiently compute a motion plan.

The paper is organized as follows: section 2 reviews recent work that addresses similar problems. Section 3 provides the detailed understanding of our approach, section 4 discusses the empirical evaluation of our approach and compares it with recent motion planning algorithms, and section 5 discusses various approaches tried before the presented approach.

2 RELATED WORK

There has been a lot of research on guiding SBMPs to improve the speed and efficiency of motion planning. Heuristically guided RRT (Urmson and Simmons 2003) efficiently uses heuristics to guide the search towards the goal. Though

this approach is useful to compute motion plans with optimal cost, such an approach would be able to work only after sufficient samples are generated to cover the entire configuration space. Burns and Brock (2005) propose to use an adaptive sampling approach that estimates the model of the *C-space* and its connectivity. Havoutis and Ramamoorthy (2010) learn a non-linear manifold of the *C-space* to estimate accurate heuristic of the states in the *C-space*. Yuandong Yang and Brock (2014) use decomposition techniques to estimate the medial axis of the state-space model to bias the sampling distribution. While the heuristic obtained using the medial axis may be effective, it does not provide any guarantees of increased performance. Zhang, Huh, and Lee (2018) use rejection sampling to reject unrelated samples to speed up the motion planning problem. Zhang, Huh, and Lee (2018) use reinforcement learning to learn a policy that decides to accept or reject a new sample to expand the search tree. While it reduces the search space to compute the path, it may suffer from higher sample rejections in environments that have narrow passages which are not easily sampled by uniform samplers. While all of these approaches try to estimate the heuristic for improving the quality of motion plans and reducing planning time, they still require an efficient sampler to provide the samples. Our approach aims to learn such a sampler that can be used to efficiently generate samples that are required to solve the problem.

Multiple approaches have tried using statistical learning to boost motion planning. Wang et al. (2021) present a comprehensive survey on methods that utilize a variety of
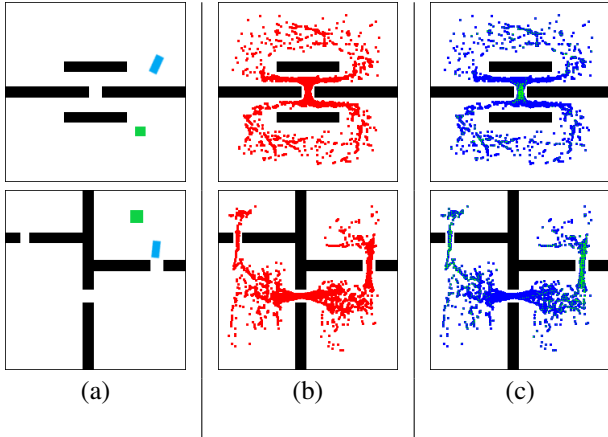
Figure 3: Training environments and labels for $\mathcal{SE}^2$ domain. (a) represents the input to the environment. Each label is generated by solving 50 motion planning problems. Red regions in (b) are critical locations of environments that are important to solve the problem. Critical regions are computed using Def. 1. (c) shows the distribution for the orientation of the robot. Blue regions show horizontal alignment of the robot and the green regions show vertical alignment of the robot. Blue rectangle in (a) shows the robot.
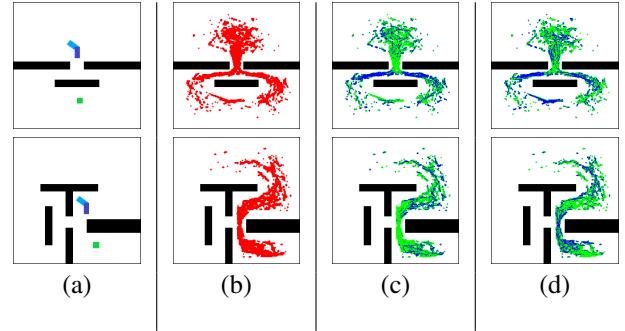

Figure 4: Training environments for *hinged robot* domain. (a) represents the input to the environment. Each label is generated by solving 50 motion planning problem. Red regions in (b) are critical locations of the environment. (c) shows the distribution of the orientation for the base link of the robot. Blue regions show the horizontal and green regions show the vertical alignment of the robot. (d) shows the distribution for the joint ω. Blue regions show that ω is closer to $180°$ and green regions show that ω is close to either $90°$ or $270°$. Blue rectangles in (a) show the robot with the hinged joint.

learning methods to improve the efficiency of the SBMPs. We discuss a few of these approaches that are relevant to our approach. Qureshi et al. (2020) use a convolutional neural network model using the data obtained from video streaming and expert demonstrations to guide the search exploration for path planning. Ichter, Harrison, and Pavone (2018) and Kumar et al. (2019) use a conditional variational autoencoder (CVAE) (Sohn, Lee, and Yan 2015) to learn the sampling distribution for the motion planning problem. Compared to the generative model presented by Ichter, Harrison, and Pavone (2018), our discriminative model requires less amount of data and provides a stable learning framework that requires less complex input representation. The simpler representation allows our approach to scaling to higher dimensional robots and reduces overall inference time. Molina, Kumar, and Srivastava (2020) use an image-based approach to learn and infer the sampling distribution using demonstrations. They use top view images of the environment with *critical regions* highlighted in the image to learn the distribution of critical regions. While their approach efficiently learns critical regions for simple navigational problems, the representation used for the labels restricts their approach to such simple navigational problems and doesn't allow them to work with arbitrary motion planning problems that have *C-spaces* of dimensions higher than two. Liu, Stadler, and Roy (2020) use semantic information to bias the sampling distribution for path planning tasks in partially known environments. Compared to Liu, Stadler, and Roy (2020), our approach is not limited to path planning problems and does not require semantic information explicitly but aims to learn such a notion in the form of critical regions.

3 METHODOLOGY

We use the notion of *critical regions* in an environment as defined by Molina, Kumar, and Srivastava (2020):

Definition 1 *Given a robot R, an environment E, and a class of motion planning problems M, the measure of criticality of a Lebesgue-measurable open set $r \subseteq \mathbb{C}$, $\mu(r)$, is defined as $\lim_{s_n \to^+ r} \frac{f(r)}{v(s_n)}$, where $f(s_n)$ is the fraction of observed motion plans solving tasks from M that pass through s_n, $v(s_n)$ is the measure of s_n under a reference density, and $\to^+$ denotes the limit from above along any sequence $\{s_n\}$ of sets containing r ($r \subseteq s_n, \forall n$).*

The above definition identifies regions that are important to solve a path planning problem. It assigns a higher score to the regions in the environment that have a higher fraction of motion plans passing through it. We use this notion of *critical regions* to compute sets of critical configuration for the given robot using motion planning demonstrations.

3.1 Learning Framework

We now explain the overall learning framework. We start by explaining how our network is able to generate a robot-specific architecture of the network and then discuss the data generation and network training.

Deriving Robot Specific Network Architectures
In this section, we explain how our approach is able to generate robot-specific network architectures using the number of degrees of freedom and the robot geometry. We use these robot specific information to update the UNet (Ronneberger, P.Fischer, and Brox 2015) base architecture as shown in the Fig. 2. We start with a fully convolutional UNet architecture that consists of a total of sixteen layers with 3×3 filters. The first eight layers are convolutional layers that work as an encoder unit and the remaining eight layers are deconvolutional layers that work as a decoder unit. We use a single
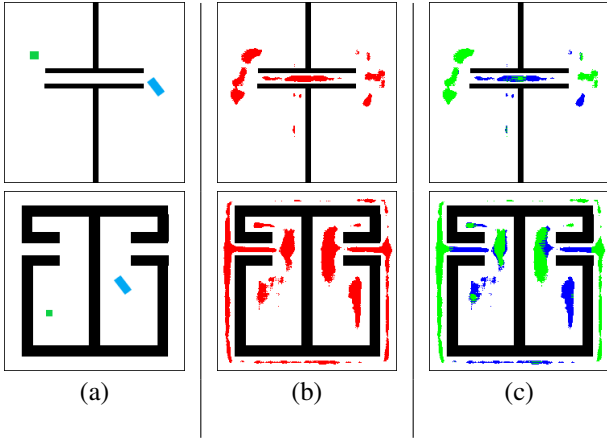
Figure 5: Test evaluations for $\mathcal{SE}^2$ domain. (a) represents the input to the environment. Top: Environment 3A, bottom: Environment 3B. Red regions in (b) are locations of environments that are predicted critical by the network. (c) shows the predicted orientation of the robot. Blue regions are locations where the network predicted robot to be horizontal and green regions are the regions where the network predicted the robot to be vertical.
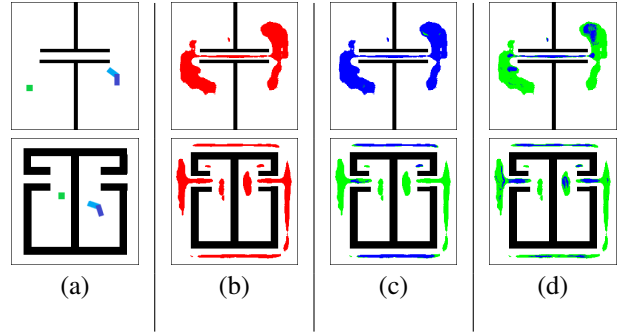

Figure 6: Test evaluations for *hinged robot* domain. (a) represents the input to the environment. Top: Environment 4A, bottom: Environment 4B. Red regions in (b) are locations of environments that are predicted critical by the network. (c) shows the predicted orientation for the base link of the robot. Blue regions are locations where the network predicted the base link to be horizontal and green regions are the regions where the network predicted the base link to be vertical. (d) shows the distribution for the joint ω. Blue regions show that the network predicted it to be closer to $180°$ and green regions show that the network predicted the angle ω close to $90°$ or $270°$.

channel of the last base layer to predict critical locations for the end-effector (base link for path planning) in the environment. We use additional k convolutional layers to predict the distribution for joint values. Here k represents the number of degrees of freedom the robot that is not captured by the location of the end-effector in the environment. For e.g., for a robot that has four degrees of freedom (x, y, θ, ω) where x and y represents the location of the robot in the environment, the network would have $k = 2$ additional convolutional layers - one convolutional layer for each remaining degree of freedom (θ and ω in this case). The dimensions of input to the network are computed using the dimensionality of the environment and the number of degrees of freedom of the robot. The first channel in the input represents the occupancy matrix of the environment and each additional channel represents the goal value for each degree of freedom of the robot. So for a robot with four degrees of freedom, the network will have five channels in the input layer.

The total number of channels in a label vector is also computed using the number of joints in the robot. We discretize the values of each joint of the robot in p discrete bins. We use a single channel to represent the distribution of critical regions for the end-effector. Additional p channels represent the distribution for values of each joint. So for a robot with four degrees of freedom (x, y, θ, ω), where x and y represents the location of the robot in the environment, the total number of output channels would be equal to 21, if $p = 10$ for θ and ω.

Training Data Generation For each instance of an environment, we generate a set of 100 motion planning tasks with randomly generated goal states. For each motion planning task, we generate 50 sub-tasks by generating 50 initial states of the robot randomly. We run an off-the-shelf motion planner on these subtasks to obtain demonstrations. We

combine the solutions for all subtasks to compute critical regions according to Def. 1 for the given environment E and goal g. For our experiments, we use OpenRAVE (Diankov 2010) robot simulator to simulate the motion planning problems and implementation of *OMPL's BiRRT* (Berenson et al. 2019) by https://ompl.kavrakilab.org to compute the motion plans.

To generate the input vector, we discretize the environment into n_d bins. We compute the occupancy matrix of the environment using the raster scan of the discretized environment. We append the goal value for each degree of freedom of the robot with the occupancy matrix to generate the input vector. The first channel of the label is generated by computing the criticality score for each discretized location of the environment using Def. 1. We use a 3×3 gaussian filter to smoothen the computed regions. Additional p channels are added to the label vector for each joint representing the distribution of its values for the given end-effector location.

Network Training The layer predicting critical locations of the environment, (CR_l), uses the sigmoid activation as the task is similar to element-wise classification. The layers predicting the distribution for individual joints (J_l) use the softmax activation as the task corresponds to multi-class classification. The loss function is defined as follows:

$$\mathcal{L} = \mathcal{L}_{CR} + \sum_{i=0}^{N_{dof}} \mathcal{L}_J^i \tag{1}$$

where,

$$\mathcal{L}_{CR} = (z-1) * log(1 - \sigma(CR_l)) - qz log(\sigma(CR_l)),$$

Here $\mathcal{L}_{CR}$ defines the weighted log loss for layers predicting the critical regions in the environment. It is important to use weighted log loss to tackle the class imbalance in the labels
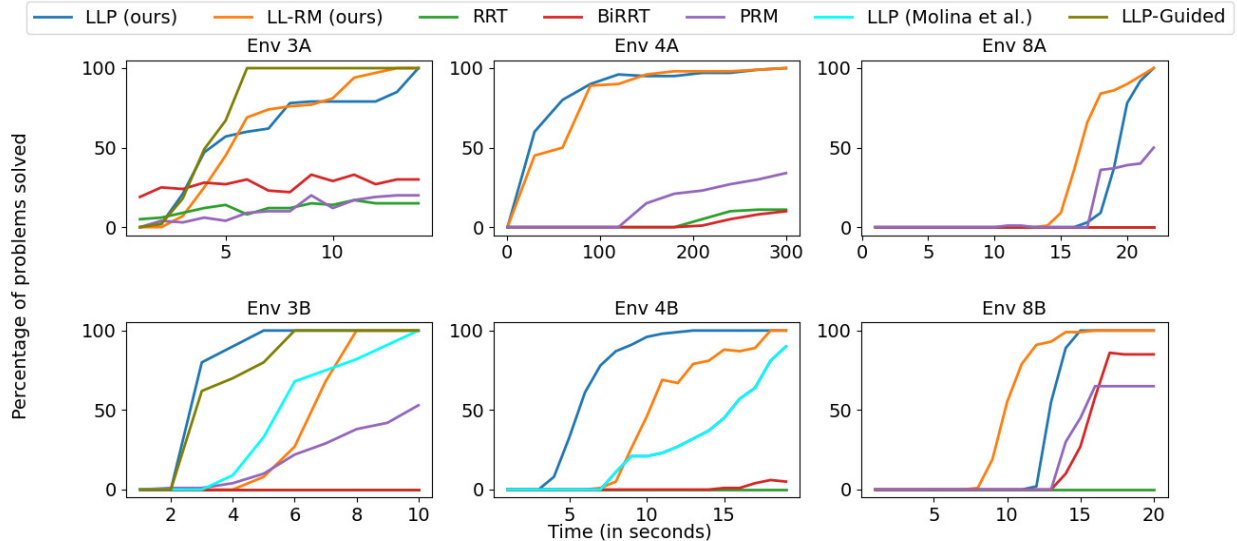
Figure 7: Plots show the fraction of motion planning tasks solved (y-axis) in the given time (x-axis) for the test environments in Figures 5, 6, and 9.

as most of the locations in the environment are not critical. σ represents the sigmoid function. q is the weight given to the positive samples. Also,

$$\mathcal{L}_J^i = -log\left(\frac{e^{z_k^i}}{\sum_m e^i}\right)$$

$\mathcal{L}_j^i$ represents the softmax cross entropy loss for i^{th} joint of the robot. We use *ADAM Optimizer* (Kingma and Ba 2014) with learning rate 10^{-4}. We implement the architecture shown in Fig. 2 using Tensorflow (Abadi et al. 2016). We train the network for $50,000$ epochs.

3.2 Motion Planning

While the biased distribution generated by our model can be used with any of the existing sampling-based motion planners, we use *Learn and Link planner* (Molina, Kumar, and Srivastava 2020) for our experiments. This planner generates multiple exploration trees rooted at multiple samples from the environment. It then tries to extend all the trees until they connect and form a single tree. Once a single tree is formed, the planner uses *Dijkstra's Algorithm* (Dijkstra 1959) to compute a path from the initial state to the goal state. For our experiments, we generate αN samples using the learned biased distribution and $(1 - \alpha)N$ samples uniformly. The planner provides two modes: 1) *LLP*: In this mode, the exploration tree is generated on the fly. 2) *LL-RM*: This mode is similar to *PRM* where the exploration graph is built initially and reused for every problem. In our evaluations, we compare results for both modes of the planner. For our experiments, we set $N = 500$ and $\alpha = 0.25$.

4 EMPIRICAL EVALUATION

We evaluate our approach with three different types of motion planning problems. In the first set of problems, we attempt to solve path planning tasks for a rectangular robot in $\mathcal{SE}^2$ environment. For the second set of problems, we update the rectangular robot to have a hinge joint ω and solve path planning problems for the updated robot. For the last set of problems, we use a mobile manipulator called Fetch (Wise et al. 2016) to solve arm manipulation tasks in a cluttered environment. We compare our approach against state-of-the-art motion planners such as *RRT* (LaValle 1998), *PRM* (Kavraki et al. 1996), and *BiRRT* (Karaman et al. 2011). Both, *PRM* and *LL-RM* were given 1.0 seconds to build the roadmap. As the *Learn and Link Planner* is implemented using *Python*, we use the *Python* implementation of the above-stated algorithms available at https://ompl.kavrakilab.org/ to compare. Our training data, code, leaned model, and results are available at https://aair-lab.github.io/hdof_ll.html.

4.1 $\mathcal{SE}^2$ domain

In this domain, the objective is to solve path planning tasks for a rectangular robot involving three degrees of freedom (x, y, θ). We use a total of 20 environments to generate the demonstrations to train the model. Some of these environments with training labels are shown in Fig. 3. We generate the training data as explained in section 3.1 using $n_d = 224$ and $p = 10$. The input to the model is a *3D tensor* of size $224 \times 224 \times 4$. Label of each input is also a *3D tensor* of size $224 \times 224 \times 11$. We augment the generated data by rotating it by $90°, 180°$, and $270°$. In total, we obtain 8000 training samples from 20 training environments.
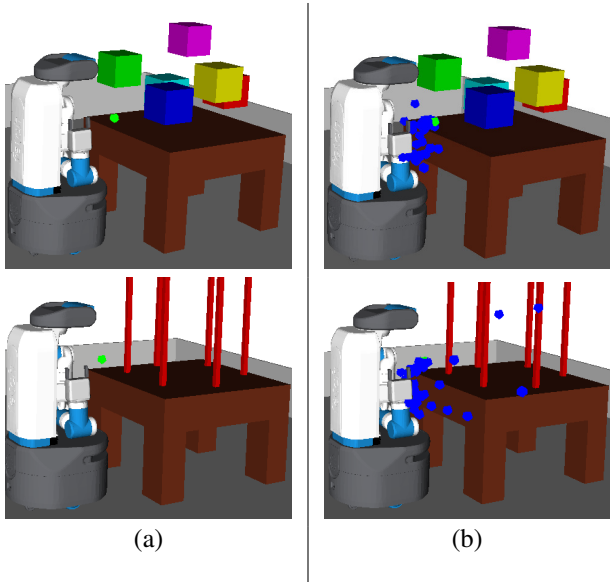
Figure 8: Training Environment for arm manipulation problems with Fetch. The green area shows the goal location for the end effector. Blue regions in (b) shows the critical regions for the end effector.
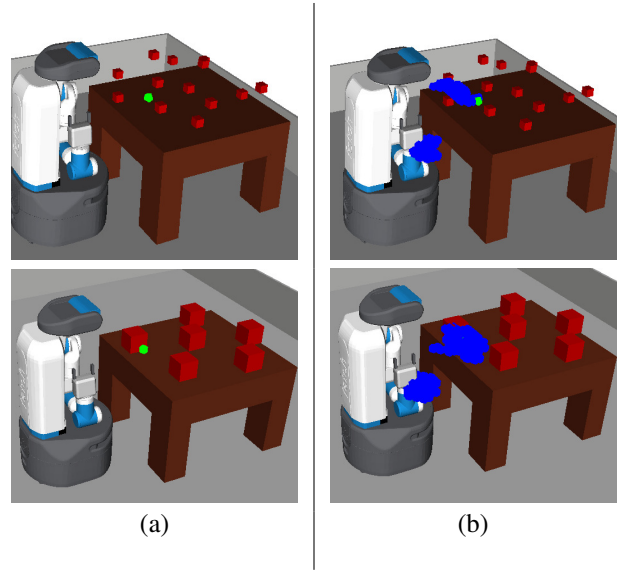

Figure 9: Test evaluations for arm manipulation tasks using Fetch. Top: Environment 8A and bottom Environment 8B. The green region shows the goal location for the end effector. Although the network predicts CRs for all the joints, only CRs for end-effector are shown in (b) in blue for clarity.

Result Analysis Fig. 5 shows the test environments and sampling distributions generated using our trained model for the current domain. Fig. 5(a) shows the environments, Env A and Env B, with the initial state (blue) and the goal state (green) which were a part of the input to the model and Fig. 5(b) and (c) shows how the network is able to learn the geometry of the environment and the robot to accurately predict the critical regions for the given problem. It can be seen from Fig. 5 that the model was able to identify the critical areas such as narrow passages and tunnels in the environment. It is also clear from Fig. 5(c) that the model was able to learn the geometrical constraints. The model has predicted horizontal orientation of the robot as needed in narrow tunnels in Env A (top) and vertical orientation for passages for Env B (bottom).

4.2 Hinged Robot

In this domain, the objective is to solve path planning problems for a hinged robot that has two rectangular links connected by a rotational joint ω. The robot has four degrees of freedom (x, y, θ, ω). The joint ω can rotate in the range $[-\pi/2, \pi/2]$. Similar to section 4.1, we use a total of 20 environments to generate the demonstrations to train the model. Fig. 4 shows some of the environments used to generate the training data. We generate training data according to section 3.1 using $n_d = 224$ and $p = 10$. The input is a *3D tesnor* of size $224 \times 224 \times 5$. The label is also a *3D tensor* of shape $224 \times 224 \times 21$. We augment the environments by $90°, 180°,$ and $270°$ to generate the variants of the input environments. We have a total of 8000 demonstrations including the augmented data.

Result Analysis Fig. 6 shows the test environments and sampling distributions generated using our trained model for the *hinged robot*. Fig 6(a) shows the environments Env A and Env B, with the goal state (green) which was given to the model as input. Similar to the previous results, it can be seen that the model was able to learn the required distribution of critical regions as well as the geometrical constraints of the robot and the environment. Red regions in Fig. 6 accurately capture the critical locations in the environment. Fig. 6(c) and (d) shows that the model was able to learn geometrical constraints for the problem by predicting horizontal orientation for the robot (blue regions) for narrow horizontal tunnels in Env A (top) and vertical orientation for the robot (green regions) for the narrow vertical passages in Env B (bottom). It is clear from Fig. 6(d) that the model was successfully able to generalize to able predict an accurate distribution for joint ω. Fig. 6(d) shows that the model was able to predict a *line shape* for the narrow channels and an *L shape* for the corners.

4.3 Arm Manipulation: Fetch robot

For the last set of problems, we use a mobile manipulator named *Fetch* (Wise et al. 2016) with *8 DOFs* to perform arm manipulation. The goal of these experiments is to show that our approach is not limited to the path planning tasks and is generalizable to any arbitrary motion planning task. Fig. 8 shows two of the eight training environments used to generate demonstrations. We generate a total of 800 training samples according to section 3.1 for eight training environments using $n_d = 64$ and $p = 10$. While the previous experiments involved robot movements along only x and y axis, these experiments included arm movements along all three x, y,

and z axis. We update the architecture shown in Fig. 2 by replacing convolutional and deconvolutional layers with *3D* convolutional and *3D* deconvolutional layers. We also reduce the number of discretized locations in the environment as the reach of the robot is much lesser for arm manipulation compared to path planning problems. The input to the model is a *4D tensor* of size $64 \times 64 \times 11$. The first channel consists of the distribution for critical locations in the environment. Label for each input is also a *4D tensor* of size $64 \times 64 \times 64 \times 81$.

Result Analysis Fig. 9 shows the test environments and sampling distributions predicted by our model. Fig. 9(a) shows the input environment to the model. Fig. 9(b) shows the critical locations predicted by the model. It can be seen from Fig. 9(b) that the model was able to predict critical locations around the goal. For simplicity, we do not show distributions for each joint, but our results show that the model was able to learn these distributions and improve the efficiency of motion planning.

4.4 Comparison

Fig. 7 shows the comparison of the number of motion planning problems solved by our approach in a given time with the available SBMPs. The *x-axis* shows the time (in seconds) and the *y-axis* shows the fraction of motion planning problems solved (out of 100) in the given time. While sampling-based motion planners such as *RRT*, *BiRRT*, and *PRM* struggle to compute motion plans due to the inability to sample critical configurations of the environment, our biased sampling distribution helps the *Learn and Link planner* to efficiently sample such configurations by using the learned model. It can be seen from Fig. 7 that our approach was able to solve a significantly higher number of problems in less than 20% time than the available sampling-based motion planners with uniform samplers. We also compare our approach in the domains 4.1 and 4.2 with the approach developed by Molina, Kumar, and Srivastava (2020). While their approach only predicts critical regions for *2D spaces*, we uniformly sample remaining joint values for the robot. Results in Fig. 7 for environments *3A* and *3B* show that the approach developed by Molina, Kumar, and Srivastava (2020) was able to perform better than the current sampling-based approaches like *RRT* and *BiRRT*, but our approach was able to outperform it significantly due to the availability of critical regions for all the degrees of freedom compared to just for the base location.

We also implemented *Guided-LLP* mode that uses identified critical regions to formulate a high-level search problem. We convert the critical regions to high-level states and construct a high-level search space. For the basic implementation, we use the minimum distance between two regions to compute the connectivity between high-level states. For our experiments, we set this distance to $0.2m$. For the given initial and goal state, a high-level plan is computed by searching the shortest path in the high-level search space that is then used to direct the low-level *LLP*. We evaluate this mode in environments *3A* and *3B*. Fig. 7 shows that for environment *3A*, *Guided-LLP* was able to perform slightly

better compared basic variant. For environment *3B*, its performance was slightly inferior compared to the basic version. This performance can be improved by using a more intelligent approach for establishing connectivity between high-level states.

5 ABLATION STUDY

Apart from the presented approach that uses UNet architecture shown in Fig. 2, we mainly tried two other approaches. As part of the first approaches, we implemented a generative network known as *conditional generative adversarial network* (CGAN) (Mirza and Osindero 2014) to predict the criticality score for all the states in an environment. The approach was not able to scale due to the dimensionality of the model. For the second approach, we tried to learn the distribution of each joint independently without identifying critical locations for the end-effector (base link for path planning) of the environment. While the approach was able to learn distributions, it failed to improve the efficiency of the planner due to the correlation between the values of each joint.

6 CONCLUSION

In this paper, we present an approach to learn biased sampling distribution that works with any available sampling-based motion planner for any arbitrary motion planning problem. Our model uses demonstrations to learn critical states of the given environment and sampling distribution for each joint of the robot. Through our experiments, we show that our approach is robust and can be scaled to robots having higher degrees of freedom.

Acknowledgements

References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. 2016. Tensorflow: A system for large-scale machine learning. In *Proc. OSDI, 2016*.

Berenson, D.; Srinivasa, S. S.; Ferguson, D.; and Kuffner, J. J. 2019. Manipulation planning on constraint manifolds. In *Proc. ICRA, 2009*.

Burns, B.; and Brock, O. 2005. Toward Optimal Configuration Space Sampling. In *Proc. RSS, 2005*.

Canny, J.; and Reif, J. 1987. New lower bound techniques for robot motion planning problems. ISBN 0-8186-0807-2.

Diankov, R. 2010. *Automated Construction of Robotic Manipulation Programs*. Ph.D. thesis, Carnegie Mellon University.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1(1): 269–271.

Havoutis, I.; and Ramamoorthy, S. 2010. Motion Synthesis through Randomized Exploration on Submanifolds of Configuration Space. In Baltes, J.; Lagoudakis, M. G.; Naruse,

T.; and Ghidary, S. S., eds., *RoboCup 2009: Robot Soccer World Cup XIII*, 92–103. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-11876-0.

Ichter, B.; Harrison, J.; and Pavone, M. 2018. Learning sampling distributions for robot motion planning. In *Proc. ICRA, 2018*.

Janson, L.; and Pavone, M. 2013. Fast Marching Trees: a Fast Marching Sampling-Based Method for Optimal Motion Planning in Many Dimensions - Extended Version. *CoRR* abs/1306.3532.

Karaman, S.; Walter, M.; Perez, A.; Frazzoli, E.; and Teller, S. 2011. Anytime Motion Planning using the RRT*.

Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation* 12(4): 566–580.

Kingma, D.; and Ba, J. 2014. Adam: A Method for Stochastic Optimization. *Proc. ICLR, 2014* .

Kumar, R.; Mandalika, A.; Choudhury, S.; and Srinivasa, S. 2019. LEGO: Leveraging Experience in Roadmap Generation for Sampling-Based Planning. *Proc. IROS, 2019* .

LaValle, S. M. 1998. Rapidly-exploring random trees: A new tool for path planning .

Liu, K.; Stadler, M.; and Roy, N. 2020. Learned Sampling Distributions for Efficient Planning in Hybrid Geometric and Object-Level Representations. In *Proc. ICRA, 2020*.

Menglu Lan; Shupeng Lai; Yingcai Bi; Hailong Qin; Jiaxin Li; Feng Lin; and Chen, B. M. 2016. BIT*-based path planning for micro aerial vehicles. In *Proc. IECON 2016*.

Mirza, M.; and Osindero, S. 2014. Conditional Generative Adversarial Nets. *CoRR* abs/1411.1784.

Molina, D.; Kumar, K.; and Srivastava, S. 2020. Identifying Critical Regions for Motion Planning using Auto-Generated Saliency Labels with Convolutional Neural Networks. *Proc. ICRA, 2020* .

Qureshi, A. H.; Miao, Y.; Simeonov, A.; and Yip, M. C. 2020. Motion planning networks: Bridging the gap between learning-based and classical motion planners. *IEEE Transactions on Robotics* .

Reif, J. H. 1979. Complexity of the mover's problem and generalizations. In *Proc. SCFS 1979*.

Ronneberger, O.; P.Fischer; and Brox, T. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, 234–241. Springer. URL http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a. (available on arXiv:1505.04597 [cs.CV]).

Sohn, K.; Lee, H.; and Yan, X. 2015. Learning Structured Output Representation using Deep Conditional Generative Models. In *Proc. NIPS, 2015*.

Urmson, C.; and Simmons, R. 2003. Approaches for heuristically biasing RRT growth. In *Proc. IROS, 2003*.

Wang, J.; Zhang, T.; Ma, N.; Li, Z.; Ma, H.; Meng, F.; and Meng, M. Q.-H. 2021. A survey of learning-based robot motion planning. *IET Cyber-Systems and Robotics* .

Wise, M.; Ferguson, M.; King, D.; Diehr, E.; and Dymesich, D. 2016. Fetch and freight: Standard platforms for service robot applications. In *Workshop on Autonomous Mobile Service Robots*.

Yuandong Yang; and Brock, O. 2014. Adapting the sampling distribution in PRM planners based on an approximated medial axis. In *Proc. ICRA, 2014*.

Zhang, C.; Huh, J.; and Lee, D. D. 2018. Learning implicit sampling distributions for motion planning. In *Proc. IROS, 2018*.