USING DEEP LEARNING TO BOOTSTRAP ABSTRACTIONS FOR ROBOT PLANNING

Naman Shah, Siddharth Srivastava
School of Computing and Augmented Intelligence
Arizona State University
Tempe, AZ, 85281, ASU
{shah.naman, siddharths}@asu.edu

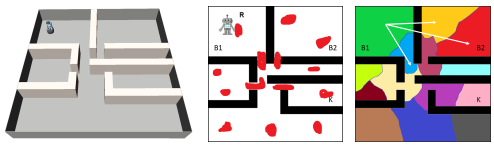
ABSTRACT

This paper addresses the problem of learning abstractions that boost robot planning performance while providing strong guarantees of reliability. Although state-of-the-art hierarchical robot planning algorithms allow robots to efficiently compute long-horizon motion plans for achieving user desired tasks, these methods typically rely upon environment-dependent state and action abstractions that need to be hand-designed by experts.

We present a new approach for bootstrapping the entire hierarchical planning process. This allows us to compute abstract states and actions for new environments automatically using the critical regions predicted by a deep neural network with an auto-generated robot-specific architecture. We show that the learned abstractions can be used with a novel multi-source bi-directional hierarchical robot planning algorithm that is sound and probabilistically complete. An extensive empirical evaluation on twenty different settings using holonomic and non-holonomic robots shows that (a) our learned abstractions provide the information necessary for efficient multi-source hierarchical planning; and that (b) this approach of learning, abstractions, and planning outperforms state-of-the-art baselines by nearly a factor of ten in terms of planning time on test environments not seen during training.

1 INTRODUCTION

Autonomous robots need to be able to efficiently compute long-horizon motion plans for achieving user desired tasks (Lo et al., 2018; Duckworth et al., 2016; Luperto et al., 2021). E.g., consider a scenario where a robot R in a household environment is tasked to reach kitchen K from its current location $B1$ (Fig. 1). State-of-the-art motion planning algorithms such as PRM (Kavraki et al., 1996) and RRT (LaValle, 1998) use random sampling of low-level configurations to compute a path from the robot's current location $B1$ to its target location K. Such sampling-based methods fail to efficiently sample configurations from confined spaces such as doorways and corridors under uniform sampling (Molina et al., 2020).



(a) (b) (c)

Figure 1: (a) An illustrative environment for a motion planning problem. The robot (R) is tasked to reach the kitchen (K). (b) Red blobs show a set of candidate critical regions in the environment. (c) an example of state abstraction. Each colored cell represents an abstract state. White arrows show a few abstract actions that takes the robot from one abstract state to another abstract states.

On the other hand, humans tend to reason using abstract, high-level actions. E.g., in the same scenario, we would use high-level (abstract) actions such as *"go out of the room B1"*, *"pass through the corridor"*, and *"enter the kitchen K"*. These abstract actions allow us to reason over a long-horizon easily. Currently, domain experts need to create such abstractions by hand. This limits the scope and scalability of approaches for hierarchical planning (e.g., (Garrett et al., 2020; Shah et al., 2020)) to situations and domains where experts are available and able to correctly intuit the required abstractions.

This paper shows that the required abstractions can be learned by identifying regions in the environment that are important to solve motion planning problems (loosely similar to landmarks in task planning). Molina et al. (2020) define such regions as *critical regions*. Critical regions are analogous to *landmarks* in automated planning but unlike landmarks, critical regions do not necessarily have to be reached to achieve the goal. Fig. 1(b) shows a few candidate critical regions for the environment in Fig. 1(a).

In this paper, we investigate two major questions: 1) Can we use deep learning to automatically generate such abstractions for new environments? And 2) how can we use them to enable safe and more efficient planning algorithms? The main contributions of this paper are: a formal foundation for hierarchical (state and action) abstractions based on the critical regions (similar to Fig. 1(c)) and a novel algorithm for on-the-fly construction of such hierarchical abstraction using predicted critical regions. Abstractions computed using learning in this manner can be difficult to use. In fact, our preliminary experiments showed that they do not help in standard single-source goal-directed search due to collision with obstacles in the environment.

An additional contribution of this work is the finding that these abstractions empower effective multi-source, multi-directional search algorithms. In general, such search algorithms can be difficult to use due to the absence of information about states that may lie close to a path to the goal. However, we found that when used with our auto-discovered abstract states, these algorithms significantly outperform existing baselines.

Our formal framework provides a way to generate sound abstractions that satisfy *downward refinement property* (Bacchus & Yang, 1991) for holonomic robots along with probabilistic completeness in the general case. Our exhaustive empirical evaluation shows that our approach outperforms state-of-the-art sampling and learning-based motion planners and requires significantly less time. This evaluation includes evaluation on a total of twenty different settings with four different robots, which include holonomic and non-holonomic robots.

2 Related Work

Much of the prior work on the topic is focused on decomposing a motion planning problem into smaller subproblems to reduce its complexity. Several approaches have been proposed that use state decomposition to reduce the complexity of a motion planning problem. *Vertical cell decomposition* (Chazelle, 1985) partitions the state space into a collection of vertical cells and computes a roadmap that passes through all of these cells. Brock & Kavraki (2001) propose a hierarchical method that uses *wavefront expansion* to compute the decomposition of the state space. While these approaches establish the foundation of decomposition-based motion planning, partitions generated through such approaches are arbitrary and do not provide any guarantees of completeness.

Zhang et al. (2018) use rejection sampling to reject unrelated samples to speed up SBMPs.They use reinforcement learning to learn a policy that decides to accept or reject a new sample to expand the search tree. While their approach reduces the search space to compute the path, it still needs to process samples generated from regions that are irrelevant for the current problem. On the other hand, our hierarchical approach refines abstract plans into low-level motion plans which reduces the number of unnecessary samples. TogglePRM (Denny & Amato, 2013) maintains roadmaps for free space and obstacle space in the configuration space to estimate the narrow passages and sample points from these narrow passages. This approach works well for environments with α-ϵ-separable passages, even though it does not compute high-level abstractions.

Multiple approaches have used statistical learning to boost motion planning. Wang et al. (2021) present a comprehensive survey of methods that utilize a variety of learning methods to improve the efficiency of SBMPs. Multiple approaches discussed by Wang et al. (2021) use end-to-end deep learning to learn low-level reactive policies. End-to-end approaches are attractive given if they succeed, they can compute solutions much faster than traditional approaches, but it is not exactly clear under which conditions these algorithms would succeed. Formally, these end-to-end deep learning-based approaches lack the guarantees about completeness and soundness that our approach provides. Wang et al. (2021) also discuss approaches that use learning to aid sampling-based motion planning. We discuss a few of these approaches that are relevant to this work. Kurutach et al. (2018) uses *InfoGAN* (Chen et al., 2016) to learn state-space partitioning for simple SE^2 robots.

While their empirical evaluation shows promising results, similarly to previous decomposition-based approaches, they do not provide any proof of completeness. It is also not clear how their approach would scale to configuration spaces that had more than two dimensions. On the other hand, our approach provides formal guarantees of completeness and soundness (for holonomic robots) and scales to high-dimensional spaces.

Ichter et al. (2018) and Kumar et al. (2019) use a conditional variational autoencoder (CVAE) (Sohn et al., 2015) to learn sampling distributions for the motion planning problems. Ichter et al. (2020) use *betweenness centrality* to learn criticality score for low-level configurations. They uniformly sample a set of configurations from the environment and use configurations with higher criticality from this set to generate a roadmap. Their results show significant improvement over vanilla PRM but it is unclear how their approach would perform if the environment had regions that are important to compute motion plans yet difficult to sample under uniform sampling. On the other hand, our approach would identify such important regions to overcome these challenges. While these approaches (Ichter et al., 2018; Kumar et al., 2019; Ichter et al., 2020) focus on biasing the sampling distribution towards narrow areas in the environment, our approach aims to build more general high-level abstractions for the configuration space.

Molina et al. (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 to identify critical regions. While they develop a method for predicting critical regions and using them with a low-level motion planner, they do not use these critical regions for learning abstractions and performing hierarchical planning. Our empirical results (Sec. 4) show that our hierarchical approach is much more effective than their non-hierarchical approach and yields significantly better performance. Additionally, their approach is also restricted to navigational problems and does not scale to configuration spaces with more than two degrees of freedom (DOFs).

Deep learning has also been used for learning heuristics for high-level symbolic planning. Shen et al. (2020) use hypergraph networks for learning heuristics for symbolic planning in the form of delete-relaxation representation of the actual planning problems. Karia & Srivastava (2021) learn generalizable heuristics for high-level planning without explicit action representations in symbolic logic. In contrast, we focus on learning critical regions and creating high-level abstractions along with algorithms that work with these learned high-level abstractions.

Liu et al. (2020) use semantic information to bias the sampling distribution for navigational problems in partially known environments. Compared to it, our approach is not navigational problems and does not require semantic information explicitly but aims to learn such a notion in the form of critical regions. *SPARK* and *FLAME* (Chamzas et al., 2020) use state decomposition to store past experience and use it when queried for similar state decompositions. While their approach efficiently uses the experience from previous iterations, it requires carefully crafted state decompositions in order to cover a large number of scenarios, whereas our approach generates state abstraction automatically using the predicted critical regions.

3 Learning Abstractions and Planning

Our approach uses a learned deep neural network to identify critical regions in a given environment and use these identified critical regions to generate hierarchical state and action abstractions. The learned neural network predicts critical regions for each joint of the robot. This requires a robot-specific architecture for the robot. Our approach uses UNet (Ronneberger et al., 2015) as the base deep network and automatically updates it using geometry and degrees of freedom of the robot to learn to predict critical regions for the given configuration space. We include a detailed discussion on updating network architecture and training the network in the appendix. Now, we first discuss our approach for generating and using abstractions using critical regions.

3.1 Generating and Using Abstractions

In this section, we describe our approach -- ***H**ierarchical **A**bstraction-guided **R**obot **P**lanner (HARP)* -- for generating abstract states and actions and using them to efficiently perform hierarchical planning. We assume standard notion of motion planning (LaValle, 2006) problem. We also define critical regions in the same way defined by Molina et al. (2020). We grow Voronoi cells around

each predicted critical regions to generate state and action abstractions. We define a labelling function ℓ that maps each Voronoi cell to an abstract state $s \in \mathcal{S}$. We define abstract actions as transitions between these abstract states. More details and theoretical guarantees about the leaned abstractions are present in Appendix B.

A naíve approach would be to generate a complete Voronoi diagram and then extract abstract states and actions from it. This would require iterating over all configurations in the configuration space and computing a large number of motion plans to identify executable abstract actions. This is expensive (and practically infeasible) for continuous low-level configuration spaces. Instead, we use the RBVD as an implicit concept. We generate abstractions on-the-fly by computing membership of low-level configurations in abstract states only when needed.

Algorithm 1: Hierarchical Abstraction-guided Robot Planner (HARP)

Input: Configuration space $\mathcal{X}$, a region predictor Φ, an initial configuration $x_0 \in \mathcal{X}$, goal configuration $x_g \in \mathcal{X}$, a custom heuristic h, low-level sampling-based motion planner *MP*

Output: A motion plan τ

1 $\rho \leftarrow$ predict_critical_regions(Φ, $\mathcal{X}$, x_0, x_g)
2 $\mathcal{S}, \mathcal{A} \leftarrow$ generate_state_action_abstractions(ρ, $\mathcal{X}$)
3 $s_0, s_g \leftarrow$ get_HL_state($\mathcal{S}$, ρ, x_0), get_HL_state($\mathcal{S}$, ρ, x_g)
4 $\mathcal{P} \leftarrow$ multi-source_bi-directional_beam_search($\mathcal{S}$, $\mathcal{A}$, s_0, s_g)
5 $\tau \leftarrow$ refine_path($\mathcal{P}$, *MP*)
6 $h \leftarrow$ update_heuristic(τ, $\mathcal{S}$)
7 return τ

Algorithm 1 describes our approach for generating and using hierarchical abstractions. Given the configuration space $\mathcal{X}$ and initial and goal configurations (x_0 and x_g) of the robot R, HARP uses a learned DNN Φ to generate a set of critical regions ρ (we include details about learning this DNN in Appendix E) (line 1). The remainder of Alg. 1 can be broken down into three important steps: 1) computing a set of candidate high-level plans, 2) refining candidate high-level plans into a low-level trajectory, and 3) updating the heuristic for abstract states. We now explain each of these steps in detail.

Computing High-Level Plans To compute high-level plans that reach the goal configuration x_g from the initial configuration x_0, first we determine abstract initial and goal states s_0 and s_g corresponding to the initial and goal configurations x_0 and x_g (line 3). To do this efficiently, we store sampled points from each critical region in a k-d-tree and query it to determine the abstract state for the given low-level configurations without explicitly constructing the complete Voronoi diagram. This allows us to dynamically and efficiently determine high-level states for low-level configurations.

Once we determine initial and goal states s_0 and s_g, we use our high-level planner to compute a set of candidate high-level plans going from s_0 to s_g (line 4). To compute these candidate high-level plans, we develop a multi-source bi-directional variant of beam search (Lowerre, 1976) that yields multiple high-level candidate plans.

We call this *multi-source bi-directional beam search*. We include the pseudocode for vanilla and multi-source bi-directional beam search in Appendix D.

Intuitively, this algorithm works as follows: we use a priority queue to maintain a fringe to keep track of the current state of the search. We initialize this fringe with multiple randomly sampled abstract states to allow the beam search to start from multiple sources. We select and expand these nodes in a specific order to compute paths that reach from the initial state to the goal state.

Formally, the nodes in the fringe are expanded as follows: Let n be a node in the fringe. We select the node to expand using $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of the path heading to n (unit cost for each action) and $h(n)$ is a custom heuristic that is defined as follows. Let m be the parent of n and let s_n and s_m be the abstract states corresponding to the nodes n and m. The heuristic $h(n)$ is computed as:

$$h(n) = h'(s_m, s_n) + min\{h'(s_n, s_i), h'(s_n, s_g)\}$$

Here, $h'(s_1, s_2) = \epsilon_{12} d^r(r_1, r_2)$ defines the estimated distance between abstract high-level states s_1 and s_2 with corresponding critical regions r_1 and r_2 respectively. s_i is the initial abstract state and s_g is the goal abstract state.

$\epsilon_{ij} \in (0, 1]$ is a constant that accounts for imprecise abstract actions. Alg. 1 dynamically changes it to update the heuristic function (line 6) and making it more accurate. Initially, ϵ_{ij} is set to 1 for all i and j. Once a low-level trajectory τ is computed (explained later), we compute the abstraction $\bar{\tau}$ of this trajectory. For each consecutive pair of abstract states $\langle s_i, s_j \rangle$ in $\bar{\tau}$, we decrease the value of ϵ_{ij} by $\epsilon_{ij}/2$. This allows HARP to use experience from previously computed trajectories to prioritize abstract actions that have low-level refinements to compute accurate high-level plans.

After an abstract state s_j is selected from the fringe, its successors are created by applying abstract actions on it and adding these successors to the fringe. This process continues until k high-level plans are found or the fringe is empty.

Once we generate a set of candidate high-level plans from multi-source bi-directional beam search, we use a low-level planner to refine these plans into a low-level collision-free trajectory from initial low-level configuration x_0 to goal configuration x_g (line 5).

Refining High-level Plans While any *probabilistically complete* motion planner can be used to refine the computed high-level plans into a low-level trajectory between given two configurations, we use *Learn and Link Planner (LLP)* (Molina et al., 2020) as a low-level planner in HARP (*MP* in Alg. 1) as it allows us to easily use the abstract states and their critical regions to efficiently compute motion plans. LLP is a sampling-based motion planner that initializes exploration trees rooted at N samples from the configuration space and extends these exploration 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.

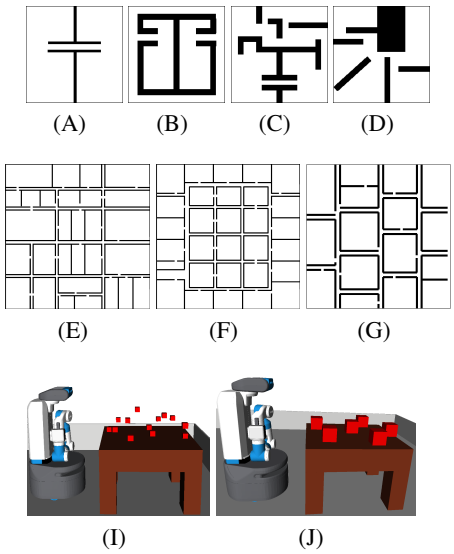


Figure 2: Test environments for our approach. Dimensions of environments (A)-(D) are $\mathbf{5m} \times \mathbf{5m}$ and dimensions of environments (E)-(G) are $\mathbf{25m} \times \mathbf{25m}$. (A)-(G) are used for navigational problems while (H) and (I) are used for manipulation problems.

In order to use LLP to refine a set of candidate high-level plans simultaneously, we first select a subset of critical regions $\bar{\rho} \subseteq \rho$ that includes all critical regions corresponding to all high-level states in candidate plans. We use this subset of critical regions $\bar{\rho}$ to provide initial samples to initialize exploration trees of LLP. In this work, we generate M samples from the set of critical regions $\bar{\rho}$ and generate the rest of the $N - M$ samples using uniform random sampling. Similarly, to expand these exploration trees, we generate a fixed number of samples from the set critical regions ρ and then continue with uniform sampling.

We use these characteristics of our algorithm to show that our approach is *probabilistic complete*.

Theorem 3.1. *If the low-level motion planner (*MP *in Alg. 1) is probabilistically complete, then HARP is probabilistically complete.*

Proof. (Sketch) While refining high-level plans to a low-level motion plan (line 5 in Alg. 1), HARP uses a fixed number of samples from the critical regions along the high-level plans to initialize a low-level motion planner. This does not reduce the set of support (regions with a non-zero probability of being sampled) of the sampling distribution being used by the motion planner. □

4 Empirical Evaluation

We extensively evaluate our approach in twenty different scenarios with four different robots. All experiments were conducted on a system running *Ubuntu 18.04* with *8-core i9* processor, *32 GB*

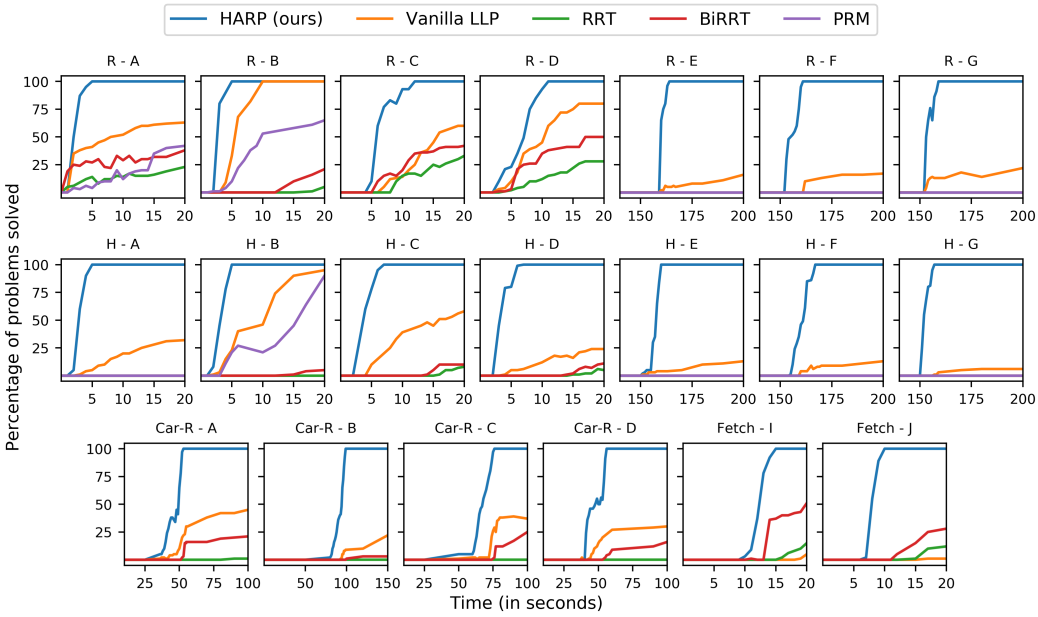


Figure 3: Each plot shows the fraction of **100** independently generated motion planning tasks solved (y-axis) in the given time (x-axis) for all the test environments and robots. The title of each subplot represents the robot and the environment. E.g., "R - A" stands for rectangular robot in environment A (Fig. 2(A)).

RAM, and an *Nvidia 2060* GPU (our approach uses only a single core) OpenRAVE robot simulator (Diankov, 2010). We compare our approach with state-of-the-art motion planners such as *RRT* (LaValle, 1998), *PRM* (Kavraki et al., 1996), and *BiRRT* (Kuffner & LaValle, 2000). As LLP is implemented using *Python*, we use the *Python* implementation of the baseline algorithms available at `https://ompl.kavrakilab.org/` for comparison. Our training data, code, trained model, and results are available at [link removed for submission].

We evaluate our approach with four different robots. We use two holonomic robots: a 3-DOF rectangular robot (R) and a 4-DOF hinged robot (H). Both robots can move along X and Y axes and rotate its body around Z axis. The hinged robot also contains an additional hinged joint that can rotate in the range $[-\frac{\pi}{2}, \frac{\pi}{2}]$. We also evaluate our approach with a non-holonomic rectangular car robot (Car) that has 3-DOFs. Controls available to operate the robot were linear velocity $v \in [-0.2, 0.2]$ and the steering angle $\theta \in [-\frac{\pi}{4}, \frac{\pi}{4}]$. Lastly, we evaluated our approach with a mobile manipulator Fetch (Wise et al., 2016) to perform arm manipulation in order to evaluate scalability of our approach to high-dimensional configuration spaces.

Figures 2 and 5 show the test environments (unseen by the model while training) for our system. Environments shown in Fig. 2 are inspired by the indoor office and household environments. Our training data consisted of 20 environments similar to the ones shown in Fig. 2(A)-(D) with dimensions $5m \times 5m$. We investigate the scalability of our approach by conducting experiments in environments shown in Fig. 2(E)-(G) with dimensions $25m \times 25m$ (much larger than training environments). To handle such large environments without making any changes to the DNN, we use the standard approach of sliding windows with stride equal to window-width (Tang et al., 2020; Birgui Sekou et al., 2018; Lu et al., 2015; Hou et al., 2016). This crops the larger environment into pieces of the size of the training environments. Individual predictions are then combined to generate a set of critical regions for arbitrary large environments. We also evaluate the applicability of our approach to non-holonomic robots in environments shown in Fig. 2(A)-(D).

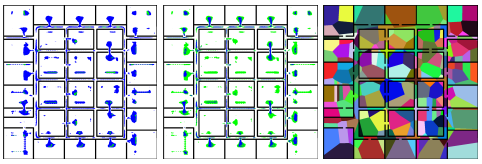
4.1 Analysis of the Results

As discussed in the introduction (Sec. 1), our objective is to show whether 1) state and action abstractions can be derived automatically and 2) whether auto-generated state and action abstractions

can be efficiently used in a hierarchical planning algorithm. Additionally, we also investigate 3) does dynamically updating the heuristic function (line 6 in Alg. 1) improve Alg. 1's efficiency?

1) CAN WE LEARN STATE AND ACTION ABSTRACTIONS?

Our approach learns critical regions for each DOF of the robot. Fig. 4 shows critical regions predicted by our learned model for the hinged robot H. We can see that our model was able to identify critical regions in the environment such as doorways and narrow hallways. Fig. 4(a) shows critical regions for orientation of the robot's base link (captured by DOF θ). The blue regions in the figure represent the horizontal orientation of the robot and the green regions represent the vertical orientation of the robot. Fig. 4(b) shows critical regions for the hinge joint ω for the robot H. Here, blue regions show that the network predicted the hinge joint to be flat (close to $0°$) and green regions represent configurations where the model predicted "L" configurations of the robot (ω close to $90°$ or $270°$). Fig. 4 shows that our approach was able to predict the correct orientation of the robot accurately most of the time. Our approach was able to scale to robots with a high number of degrees of freedom. Fig. 5(a) shows one of the test environments used for these experiments and Fig. 5(b) shows the predicted critical regions. This shows that our model was able to learn critical regions in the environment that can be used to generate efficient abstractions. We include similar results for other environments in Appendix F.



(a) (b) (c)

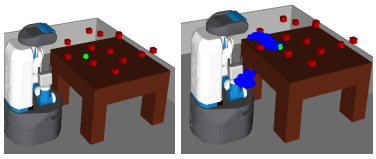
Figure 4: Critical regions and generated abstraction for a **4**-DOF hinged robot. (a) and (b) Critical regions predicted by the model. Blue regions in (a) show that model predicted the robot's base link to be horizontal while green regions show that the model predicted the robot's base link to be vertical. Blue regions in (b) show that the network predicted the hinge to be closer to $\mathbf{180}°$ and green regions show that the network predicted it to be closer to $\mathbf{90}°$ or $\mathbf{270}°$. (c) State abstraction generated by our approach though our approach does not need to explicitly generate these abstractions.

Now we answer the second question on whether using abstractions to compute motion plans helps improve the planner's efficiency by qualitatively comparing our approach with a few existing sampling-based motion planners.

b) CAN LEARNED ABSTRACTIONS BE USED EFFICIENTLY FOR HIERARCHICAL PLANNING?

We compare our approach against widely used SBMPs such as RRT (LaValle, 1998), PRM (Kavraki et al., 1996), and BiRRT (Kuffner & LaValle, 2000).

Fig. 3 shows the comparison of our approach with other sampling-based motion planners. The x-axis shows the time limit in seconds and the y-axis shows the percentage of motion planning problems solved in that time limit. For each time limit on x-axis, we randomly generate 100 new motion planning problems to thoroughly test our approach and reduce statistical inconsistencies. Fig. 3 shows that our approach significantly outperforms all of the existing sampling-based motion planners. Specifically for environments C, D, and E, uniform sampling-based approaches were not able to solve a single problem in a time threshold of $600s$.



(a) (b)

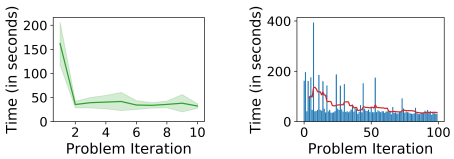
Figure 5: Critical regions for arm manipulation task using an **8**-DOF Fetch. (a) The green region shows the goal location for the end effector. (b) Critical regions generated by the learned model for end-effector's location in workspace. Although the network predicts CRs for all the joints, only critical regions for end-effector's location are shown.

Our approach also outperforms the learning-based planner LLP (Molina et al., 2020) (Fig. 3), which uses learned critical regions but does not use state and action abstractions and does not perform

hierarchical planning. This illustrates the value of learning abstractions and using them efficiently for hierarchical planning.

Similarly, we also evaluate our approach against TogglePRM (Denny & Amato, 2013). TogglePRM is written in C++ and accepts only discrete SE^2 configuration space for a simple dot robot. We created discrete variants of environments shown in Fig. 2(A) and 2(B) with a total of 50176 states and compare the total number of nodes sampled. For 100 random trials, on an average our approach generated 631 ± 278 and 496 ± 175 states compared to TogglePRM which generated 4234 ± 532 and 19234 ± 4345 states for discrete variants of the environments shown in Fig. 2(A) and 2(B) respectively. Our approach was able to outperform TogglePRM since these environments do not have α-ϵ-separable passages (Denny & Amato, 2013).

c) Does Dynamically Updating Heurstic Function Improve Efficiency?



(a) (b)

Figure 6: (a) Solving **20** randomly generated problems repeatedly 1**10** times. x-axis show the problem iteration and y-axis shows the average time over **20** problem instances. (b) Time taken to solve **100** randomly generated problem instances. X-axis shows the problem number and y-axis shows the taken to solve each problem.

We carried out two sets of experiments. In the first set of experiments, we generated 20 random motion planning problems and solved each problem repeatedly for 10 times while updating the heuristic function. We maintained separate copies of high-level heuristic functions for each problem. Fig. 6(a) shows the results for this set of experiments in the environment E (Fig. 2(E)) with the 4-DOF hinged robot. The x-axis shows the planning iteration and the y-axis shows the average time over randomly generated 20 problem instances. We can see how planning time reduces drastically once costs for abstract actions are updated.

In the second set of experiments, we generated 100 random pairs of initial and goal states and computed motion plans for each of them. This time, we maintained a single heuristic function across all problems and updated it after each motion planning query. Fig. 6(b) shows the result of the experiment in the environment E (Fig. 2(E)) with the 4-DOF hinged robot H. The x-axis shows the problem number and the y-axis shows the time taken by our approach to compute a solution. The red line in the plot shows the moving average of planning time. Fig. 6(b) shwows that dynamically updating the heuristic function for high-level planning helps to increase the efficiency of HARP and decrease motion planning times.

Empirical evaluation using these experiments validates our hypothesis that learning abstractions and effectively using them improves motion planning efficiency.

5 Conclusion

In this paper, we presented a probabilistically complete approach HARP, that uses deep learning to identify abstractions for the input configuration space. It learns state and action abstractions in a bottom-up fashion and uses them to perform efficient hierarchical robot planning. We developed a new multi-source bi-directional planning algorithm that uses learned state and action abstractions along with a custom dynamically maintained cost function to generate candidate high-level plans. A low-level motion planner refines these high-level plans into a trajectory that achieves the goal configuration from the initial configuration.

Our formal framework provides a way to generate sound abstractions that satisfy the downward refinement property for holonomic robots. Our empirical evaluation on a large variety of problem settings shows that our approach is able to significantly outperform state-of-the-art sampling and learning-based motion planners. Through our empirical evaluation, we show that our approach is robust and can be scaled to large environments and to robots that have high degrees of freedom. Our work presents a foundation for learning high-level, abstractions from low-level trajectories. Currently, our approach works for deterministic robot planning problems. We aim to extend our approach to support stochastic settings and learn abstractions for task and motion planning problems.

ACKNOWLEDGMENTS

We thank Abhyudaya Srinet for his help in implementing a primitive version of the presented work. We also thank Kyle Atkinson for his help with creating test environments. This work is supported in part by the NSF under grants 1909370 and 1942856.

REFERENCES

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *Proc. OSDI, 2016*, 2016.

Fahiem Bacchus and Qiang Yang. The downward refinement property. In *Proc. IJCAI, 1991*, 1991.

Taibou Birgui Sekou, Moncef Hidane, Julien Olivier, and Hubert Cardot. Retinal blood vessel segmentation using a fully convolutional network – transfer learning from patch- to image-level. In *Proc. MLMI, 2018*, 2018.

Oliver Brock and Lydia E. Kavraki. Decomposition-based motion planning: A framework for real-time motion planning in high-dimensional configuration spaces. In *Proc. ICRA, 2001*, 2001.

Constantinos Chamzas, Zachary Kingston, Carlos Quintero-Peña, Anshumali Shrivastava, and Lydia E Kavraki. Learning sampling distributions using local 3d workspace decompositions for motion planning in high dimensions. 2020.

Bernard Chazelle. Approximation and decomposition of shapes. *Algorithmic and Geometric Aspects of Robotics*, 1:145–185, 1985.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. In *Proc. NeurIPS, 2016*, 2016.

Jory Denny and Nancy M. Amato. Toggle prm: A coordinated mapping of c-free and c-obstacle in arbitrary dimension. volume 86, pp. 297–312. Springer, Berlin, Heidelberg., 2013.

Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, 2010.

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

Paul Duckworth, Yiannis Gatsoulis, Ferdian Jovan, Nick Hawes, David C Hogg, and Anthony G Cohn. Unsupervised learning of qualitative motion behaviours by a mobile robot. In *Proc. AAMAS, 2016*, 2016.

Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proc. ICAPS, 2020*, 2020.

Le Hou, Dimitris Samaras, Tahsin M Kurc, Yi Gao, James E Davis, and Joel H Saltz. Patch-based convolutional neural network for whole slide tissue image classification. In *Proc. CVPR, 2016*, 2016.

Brian Ichter, James Harrison, and Marco Pavone. Learning sampling distributions for robot motion planning. In *Proc. ICRA, 2018*, 2018.

Brian Ichter, Edward Schmerling, Tsang-Wei Edward Lee, and Aleksandra Faust. Learned critical probabilistic roadmaps for robotic motion planning. In *Proc. ICRA, 2020*, 2020.

Rushang Karia and Siddharth Srivastava. Learning generalized relational heuristic metworks for model-agnostic planning. In *Proc. AAAI, 2021*, 2021.

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

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Proc. ICLR, 2014*, 2014.

James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proc. ICRA, 2000.*, 2000.

Rahul Kumar, Aditya Mandalika, Sanjiban Choudhury, and Siddhartha S. Srinivasa. Lego: Leveraging experience in roadmap generation for sampling-based planning. In *Proc. IROS, 2019*, 2019.

Thanard Kurutach, Aviv Tamar, Ge Yang, Stuart Russell, and Pieter Abbeel. Learning plannable representations with causal InfoGAN. 2018.

Steven . M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

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

Katherine Liu, Maritna Stadler, and Nicholas Roy. Learned sampling distributions for efficient planning in hybrid geometric and object-level representations. In *Proc. ICRA, 2020*, 2020.

Shih-Yun Lo, Shiqi Zhang, and Peter Stone. Petlon: Planning efficiently for task-level-optimal navigation. In *Proc. AAMAS, 2018*, 2018.

Bruce T Lowerre. *The Harpy Speech Recognition System.* Carnegie Mellon University, 1976.

Xin Lu, Zhe Lin, Xiaohui Shen, Radomir Mech, and James Z Wang. Deep multi-patch aggregation network for image style, aesthetics, and quality estimation. In *Proc. ICCV, 2015*, 2015.

Matteo Luperto, Luca Fochetta, and Francesco Amigoni. Exploration of indoor environments predicting the layout of partially observed rooms. In *Proc. AAMAS, 2021*, 2021.

Daniel Molina, Kislay Kumar, and Siddharth Srivastava. Learn and link: Learning critical regions for efficient planning. 2020.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Proc. MICCAI, 2015*, 2015.

Naman Shah, Deepak Kala Vasudevan, Kislay Kumar, Pranav Kamojjhala, and Siddharth Srivastava. Anytime integrated task and motion policies for stochastic environments. In *Proc. ICRA, 2020*, 2020.

William Shen, Felipe Trevizan, and Sylvie Thiébaux. Learning domain-independent planning heuristics with hypergraph networks. In *Proc. ICAPS, 2020*, 2020.

Kihyuk Sohn, H. Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *Proc. NIPS, 2015*, 2015.

Chaohui Tang, Qingxin Zhu, Wenjun Wu, Wenlin Huang, Chaoqun Hong, and Xinzheng Niu. Planet: Improved convolutional neural networks with image enhancement for image classification. *Mathematical Problems in Engineering*, 2020, 2020.

Jiankun Wang, Tianyi Zhang, Nachuan Ma, Zhaoting Li, Han Ma, Fei Meng, and Max Q-H Meng. A survey of learning-based robot motion planning. *IET Cyber-Systems and Robotics*, 2021.

Melonee Wise, Michael Ferguson, Derek King, Eric Diehr, and David Dymesich. Fetch and freight: Standard platforms for service robot applications. In *Workshop on Autonomous Mobile Service Robots*, 2016.

Clark Zhang, Jinwook Huh, and Daniel D Lee. Learning implicit sampling distributions for motion planning. In *Proc. IROS, 2018*, 2018.

APPENDIX

A PRELIMINARIES

Motion Planning Problem Let $\mathcal{X} = \mathcal{X}_{\text{free}} \cup \mathcal{X}_{\text{obs}}$ be the configuration space of a given robot (LaValle, 2006). Here $\mathcal{X}_{\text{free}}$ represents the set of configurations where the robot is not in collision with any obstacle and $\mathcal{X}_{\text{obs}}$ represents configurations in collision with an obstacle. Let $x_i \in \mathcal{X}_{\text{free}}$ and $x_g \in \mathcal{X}_{\text{free}}$ be the initial and goal configurations of the robot. A motion planning problem is defined as follows:

Definition 1. *A motion planning problem $\mathcal{M}$ is defined as a 4-tuple $\langle \mathcal{X}, u, x_i, x_g \rangle$. where $\mathcal{X} = \mathcal{X}_{free} \cup \mathcal{X}_{obs}$ is the configuration space and $x_i, x_g \in \mathcal{X}_{free}$ are the robot's initial and goal configurations. $u : \mathcal{X} \to \{0, 1\}$ is a collision function that determines collisions for configurations: $u(x) = 1$ iff $x \in \mathcal{X}_{obs}$.*

A solution to a motion planning problem is a collision-free trajectory $\tau : [0, 1] \to \mathcal{X}$ such that $\tau(0) = x_i$ and $\tau(1) = x_g$. We abuse the notation to define membership in a trajectory as follows: For a configuration $x \in \mathcal{X}$, $x \in \tau$ iff there a exists a $t \in [0, 1]$ such that $\tau(t) = x$. A trajectory is collision free iff $\forall x \in \tau, u(x) = 0$.

Connectivity A pair of low-level configurations $x_i, x_j \in \mathcal{X}$ is said to be connected iff there exists a collision-free motion plan between x_i and x_j. We represent this using a connectivity function $C : \mathcal{X}_{\text{free}} \times \mathcal{X}_{\text{free}} \to \{0, 1\}$: $C(x_i, x_j) = 1$ iff x_i and x_j are connected. Intuitively, the connectivity function C represents Euclidean connectivity in $\mathcal{X}_{\text{free}}$. This is equivalent to path connectivity in configuration space for holonomic robots as each degree of freedom can be controlled independently. However, this may not be the case for non-holonomic robots as some of the motion plans may not be realizable due to their motion constraints.

We use this to define strong connectivity for the set of low-level configurations as follows:

Definition 2. *Let $\bar{\mathcal{X}} \subseteq \mathcal{X}_{free}$ be a set of configurations. $\bar{\mathcal{X}}$ is a strongly connected set iff for every pair of configurations $(x_i, x_j) \in \mathcal{X}_{free} \times \mathcal{X}_{free}$, $C(x_i, x_j) = 1$ and there exists a trajectory τ_{ij} such that $\tau_{ij}(0) = x_i$, $\tau_{ij}(1) = x_j$, and $\forall\, t \in [0, 1], \tau_{ij}(t) \in \bar{\mathcal{X}}$,*

Critical Regions Our approach uses *critical regions (CRs)* to generate abstractions. Intuitively, critical regions are regions in the configuration space that have a high density of valid motion plans passing through them for the given class of motion planning problems. Molina et al. (2020) define critical regions as follows:

Definition 3. *Given a robot R, a configuration space $\mathcal{X}$, and a class of motion planning problems M, the measure of criticality of a Lebesgue-measurable open set $r \subseteq \mathcal{X}$ is defined as $\lim_{s_n \to^+ r} \frac{f(r)}{v(s_n)}$, where $f(r)$ 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 (usually uniform), and $\to^+$ denotes the limit from above along any sequence $\{s_n\}$ of sets containing r ($r \subseteq s_n$, $\forall\, n$).*

Beam Search Lowerre (1976) introduced beam search as an optimization over breadth-first search (BFS) that explores the state space by expanding only a subset of nodes to compute a path from one node to another node in the graph. Beam search prunes the fringe to reduce the size of *OPEN* set. We include pseudocode for the beam search in Appendix C.

B HIERARCHICAL STATE AND ACTION ABSTRACTIONS

We begin describing our formal framework with an example. Fig. 1(b) shows a set of critical regions for a given environment. Ideally, we would like to predict these critical regions and generate a state and action abstraction similar to the one shown in Fig. 1(c). The state abstraction shown in Fig. 1(c), similar to a Voronoi diagram, generates cells around each critical region such that the distance from each point in a cell to its corresponding critical regions is less than that from every other critical

region. We call this structure a *region-based Voronoi diagram (RBVD)*. Each cell in this region-based Voronoi diagram is considered an abstract state and transitions between these Voronoi cells (abstract states) define abstract actions.

Let ρ be the set of critical regions for the given configuration space $\mathcal{X}$. First we introduce distance metrics d^c and d^r. Here, d^c defines distance between a low-level configuration $x \in \mathcal{X}$ and a critical region $r \in \rho$ such that $d^c(x, r) = \min_{x_i \in r} d(x, x_i)$ and d^r defines the distance between two critical regions r_1, r_2 such that the distance $d^r = \min_{x_i \in r_1, x_j \in r_2} d(x_i, x_j)$ where d is the Euclidean distance. Now, we define region-based Voronoi diagram as follows:

Definition 4. *Let $\rho = \{r_1, ..., r_k\}$ be a set of critical regions for the configuration space $\mathcal{X}$. A region-based Voronoi diagram (RBVD) is a partition $\Psi(\rho, \mathcal{X}) = \{\psi_1, ..., \psi_m\}$ of $\mathcal{X}$ such that for every $\psi_i \in \Psi$ there exists a critical regions r such that forall $x \in \psi_i$ and forall $r_j \neq r$, $d^c(x, r) \leq d^c(x, r_j)$ and each ψ_i is strongly connected.*

State Abstraction We define abstract states as the Voronoi cells of an RBVD. Given an RBVD Ψ, labelling function $\ell : \Psi \rightarrow \mathcal{S}$ maps each cell in the RBVD to a unique abstract state $s \in \mathcal{S}$ where $|\mathcal{S}| = |\Psi|$. We use this to define the state abstraction function α as follows:

Definition 5. *Let R be the robot and $\mathcal{X} = \mathcal{X}_{free} \cup \mathcal{X}_{obs}$ be the configuration space of the robot R with set of critical regions ρ. Let $\Psi(\rho, \mathcal{X}) = \{\psi_1, ..\psi_k\}$ be an RBVD for the robot R, configuration space $\mathcal{X}$, and the set of critical regions ρ and let $\mathcal{S} = \{s_1, .., s_k\}$ be a set of high-level, abstract states. We define abstraction function $\alpha : \mathcal{X}_{free} \rightarrow \mathcal{S}$ such that $\alpha(x) = s$ where $x \in \psi$ and $\ell(\psi) = s$.*

We extend this notation to define membership in abstract states as follows: Given a configuration space $\mathcal{X} = \mathcal{X}_{\text{free}} \cup \mathcal{X}_{\text{obs}}$ and its set of abstract states $\mathcal{S}$ as defined above, a configuration $x \in \mathcal{X}_{\text{free}}$ is said to be a member of an abstract state $s \in \mathcal{S}$ (denoted $x \in s$) iff $\alpha(x) = s$. We also extend the notion of strong connectivity to abstract states as follows: An abstract state $s \in \mathcal{S}$ is strongly connected iff $\ell^{-1}(s)$ is strongly connected. We now define adjacency for Voronoi cells in a region-based Voronoi diagram as follows. Recall that C denotes Euclidean connectivity for configurations.

Definition 6. *Let ψ_i, ψ_j be Voronoi cells of an RBVD Ψ. Voronoi cells ψ_i and ψ_j are adjacent iff there exist configurations x_i, x_j such that $x_i \in \psi_i$, $x_j \in \psi_j$, $C(x_i, x_j) = 1$, and there exists a trajectory τ between x_i and x_j such that $\forall t \in [0, 1], \tau(t) \in \psi_i$ or $\tau(t) \in \psi_j$.*

We extend the above definition to define the neighborhood for an abstract state. Two abstract states $s_i, s_j \in \mathcal{S}$ are neighbors iff $\ell^{-1}(s_i)$ and $\ell^{-1}(s_j)$ are adjacent. We define abstract actions as transitions between abstract states. Let $\mathcal{S}$ be the set of abstract states. We define the set of abstract actions $\mathcal{A}$ using $\mathcal{S}$ such that $\mathcal{A} = \{a_{ij} | \forall\ (s_i, s_j) \in \mathcal{S} \times \mathcal{S}\}$. We now use this formulation of RBVD and state abstraction to prove the soundness of the generated abstractions .

Theorem B.1. *Let $\mathcal{X} = \mathcal{X}_{free} \cup \mathcal{X}_{obs}$ be a configuration space and ρ be a set of critical regions for $\mathcal{X}$. Let Ψ be an RBVD for the critical regions ρ and the configuration space $\mathcal{X}$ and let $\mathcal{S}$ be the set of abstract states corresponding to Ψ with a mapping function ℓ. Let x_0 and x_g be the initial and goal configurations of a holonomic robot R. If every state $s \in \mathcal{S}$ is strongly connected and there exists a sequence of abstract states $P = \langle s_{\psi_0}, ..., s_{\psi_g} \rangle$ such that $x_0 \in s_{\psi_0}$, $x_g \in s_{\psi_g}$, and all consecutive states $s_{\psi_i}, s_{\psi_{i+1}} \in P$ are neighbors, then there exists a motion plan for R that reaches x_g from x_0 with a trajectory τ such that $\tau(0) = x_0$, $\tau(1) = x_g$, and $\forall x_i \in \tau, x_i \in s_{\psi_k}$ such that $s_{\psi_k} \in P$.*

Theorem B.1 proves that the computed abstractions would be sound as well as satisfy downward refinement property for holonomic robots. The proof does not hold for non-holonomic robots as the low-level trajectories may not be realizable given their motion constraints. However, the algorithm developed below is probabilistically complete for all robots and performed well for non-holonomic robots in our empirical evaluation.

C VANILLA BEAM SEARCH

Algorithm 2: Beam Search

Input: Graph $G =< V, E >$, initial state $s_0 \in V$, goal state $s_g \in V$, beam width w, heuristic h
Output: A set of paths from s_0 to s_g

```
fringe = PriorityQueue()
fringe.add(0,(s_0,None,[]))
while path not found and fringe is not empty do
    working_fringe ← select top w nodes from the fringe
    empty_fringe(fringe)
    while working_fringe is not empty do
        current, path, visited ← working_fring.pop()
        if current = s_g then
            return path
        else
            path.add(current)
            visited.add(current)
            foreach node ∈ current.successors do
                if node ∉ visited then
                    p ← g(current) + h(s')
                    fringe.add(p, (current, path, visited))
return false
```

D MULTI-SOURCE BI-DIRECTIONAL BEAM SEARCH

Algorithm 3: Multi-source Bi-directional beam search

Input: Set of states $\mathcal{S}$, set of actions $\mathcal{A}$, initial state s_0, goal state s_g, distance function h', beam width w, number of high-level plans N
Output: A set of N paths from s_0 to s_g

```
fringe = PriorityQueue()
fringe.add(s_0), fringe.add(s_g)
S̄ ← sample_states(S)
solutions = Set()
foreach s ∈ S̄ do
    fringe.add(0,(s, None, Set()))
while N > 0 and fringe is not empty do
    working_fringe ← select top w nodes from the fringe
    empty_fringe(fringe)
    while working_fringe is not empty do
        current, path, visited ← working_fringe.pop()
        if current = s_g then
            add path to solutions
            N ← N − 1
        else
            path.add(current)
            visited.add(current)
            foreach node ∈ current.successors do
                if node ∉ visited then
                    p ← h'(current, node) + min{h'(node, s_0), h'(node, s_g)}
                    fringe.add(p, (current, path, visited))
return solutions
```

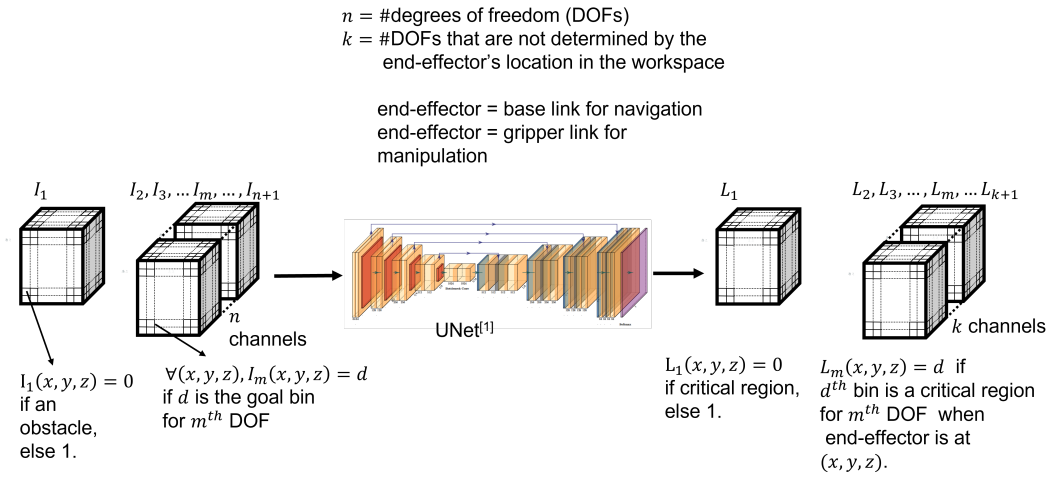


Figure 7: Overall architecture of our network. Shapes for input and label tensors are determined by the geometry of the robot and the number of degrees of freedom of the robot. The base UNet[1] architecture is updated using these calculated input and output shape.

E Learning to Predict Critical Regions

We now present our approach for learning a model Φ that predicts critical regions for a given environment. We first explain how our approach is able to generate a robot-specific architecture of the network and then describe the training process.

Deriving Robot Specific Network Architectures

We use the standard fully convolutional UNet architecture (Ronneberger et al., 2015) as our base network. Fig. 7 shows the overall architecture of our network alongside its input and label format. We use the robot's geometry and its number of DOFs to derive a robot specific architecture as follows:

Let n be the number of DOFs of the robot and let k be the number of DOFs that are not determined by the location of the robot's end-effector in the workspace. For manipulation problems, we consider gripper of the robot as the its end-effector and for navigational problems, we consider the robot's base link as its end effector. First, we use these parameters to update the base UNet architecture. We use the last layer of the base architecture to predict critical regions for the end-effector's location and include k additional convolutional layers to predict critical regions for each k DOFs that are not determined by the location of the robot's end effector.

The input to the network is a tensor of dimension four. The size of the first three dimensions of the input tensor depends on the number of bins used to discretize the environment (which can be arbitrary). The number of channels in the input tensor is determined using the parameter n. If the robot has n DOFs, then the input tensor would have a total of $n + 1$ channels. The first channel in the input represents the occupancy matrix of the environment. It is generated by performing a raster scan of the environment. The rest of the n channels represent goal values for each DOF of the robot -- one for each DOF of the robot.

Similarly, each label is a tensor of dimension four. The size of the first three dimensions is similar to the input tensor. The number of channels in the label tensor is also computed using the robot's geometry. For a robot with k DOFs that are not determined by the location of robot's end effector in the workspace, the label tensor would have a total of $k + 1$ channels. The first channel represents critical regions for the end-effector's location in the workspace and the rest of the k channels repre-

[1]O. Ronneberger, P.Fischer, and T. Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In Proc. MICCAI, 2015

sent critical regions for the k DOFs that are not determined by this location -- one channel for each of the k DOFs of the robot.

E.g, consider a 5-DOF hinged robot. The robot's 5 DOFs are $(x, y, z, \theta, \omega)$ where x, y, and z represent the location of the robot's base link in the workspace, θ represents the rotation of the base link, and ω represents the hinged angle. So for this robot, n would equal to 5 and k would equal to 2 as only the base rotation θ and the hinged angle ω are not determined by the location of the robot's end-effector (base link in this case) in the workspace. So according to the previous discussion, the network would contain $k = 2$ additional layers to predict critical regions for θ and ω. The input tensor would have a total of $n + 1 = 6$ channels and the label tensor would have a total of $k + 1 = 3$ channels.

TRAINING THE NETWORK

The layer predicting critical regions for the end-effector's locations in the workspace, (L_l), uses the sigmoid activation as the task is similar to element-wise classification. The layers predicting critical regions for the rest of the degrees of freedom that are not determined by the end-effector's location, (L_i) use the softmax activation as the task corresponds to multi-class classification. The loss function is defined as follows:

$$\mathcal{L} = \mathcal{L}_{L_1} + \sum_{i=0}^{k} \mathcal{L}_{L_i}$$

where, $\mathcal{L}_{L_1}$ is weighted log loss and $\mathcal{L}_{L_i}$ is softmax cross entropy loss for i^{th} degree of freedom not determined by end-effecot's location in the workspace. We use *ADAM Optimizer* (Kingma & Ba, 2014) with learning rate 10^{-4}. We implement the UNet architecture shown in the appendix using Tensorflow Abadi et al. (2016). We train the network for $50,000$ epochs.

GENERATING TRAINING DATA

We use 20 training environments to generate training data for navigational problems (robots R, Car, and H) and 6 training environments for manipulation problems (Fetch robot). To generate training data for each training environment E, we randomly sample a set of 100 configurations $\mathcal{G}$, which would serve as goal states for the motion planning problems. For each goal configuration $g_i \in \mathcal{G}$, we sample a set of 50 initial states $\mathcal{I}$. We use an off-the-shelf motion planner to compute motion plans for these inital and goal states and combined solutions to generate critical regions for each goal state. We use these motion plans to compute critical regions for the given pair of environment E and the goal configuration g using the Def. 3. We use OpeanRAVE robot simulator (Diankov, 2010) and OMPL's implementation of BiRRT (Kuffner & LaValle, 2000) to generate the training data.

To generate the input vector, we discretize the environment into n_d bins. This also implies that the degrees of freedom of the robot that are determined by the robot's end-effecotr's location in the workspace are also discretized into n_d bins. We discretize the rest of the degrees of freedom that are not determined by the robot's end-effecotr's location into p bins to generate input and label tensors according to the discussion in Sec. E. We augment the computed tensors by rotating them by $90°$, $180°$, and $270°$ to obtain more training samples. We also omit an additional dimension from the tensors for navigational problems as we fix the robot's z-axis for these problems. The table below shows the training details for each robot.

Robot	n_d	p	Input Shape	Label Shape	#Training Environments	# Samples
R, Car	224	4	(224,224,4)	(224,224,2)	20	8000
H	224	5	(224,224,3)	(224,224,21)	20	8000
Fetch	64	10	(64,64,64,11)	(64,64,64,9)	6	720

F RESULTS

F.1 3-DOF RECTANGULAR AND CAR ROBOTS

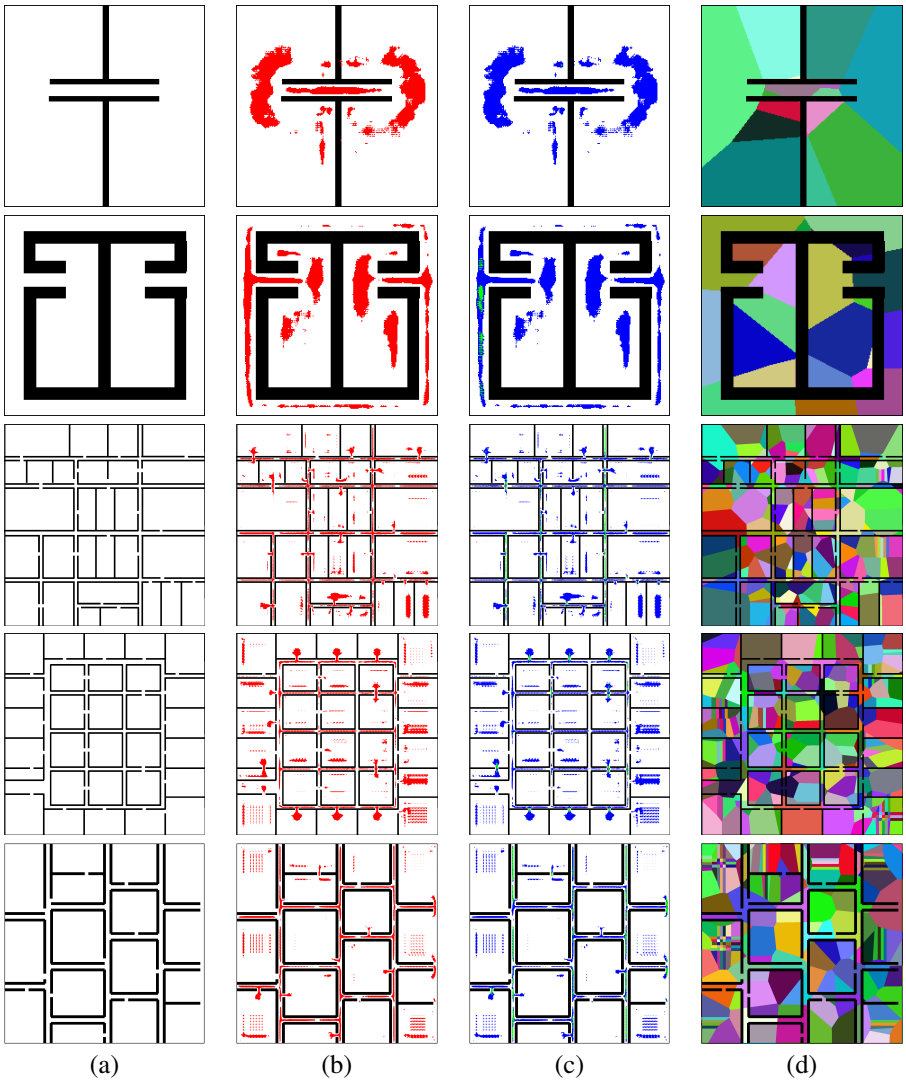


Figure 8: Predicted critical regions and computed state abstractions for 3-DOF rectangular (R) and Car robots. (a) Input to the environment. (b) Critical regions for the location of the robot's base link in the workspace. (c) Critical regions for the rotation of the robot's base link of the the robot. Blue regions are locations where the network predicted the robot to be horizontal and green regions are the regions where the network predicted the robot to be vertical. (d) 2D projections of state abstraction generated by our approach. State abstractions are strictly for visualization as our approach does not require to generate them.

F.2 4-DOF HINGED ROBOT

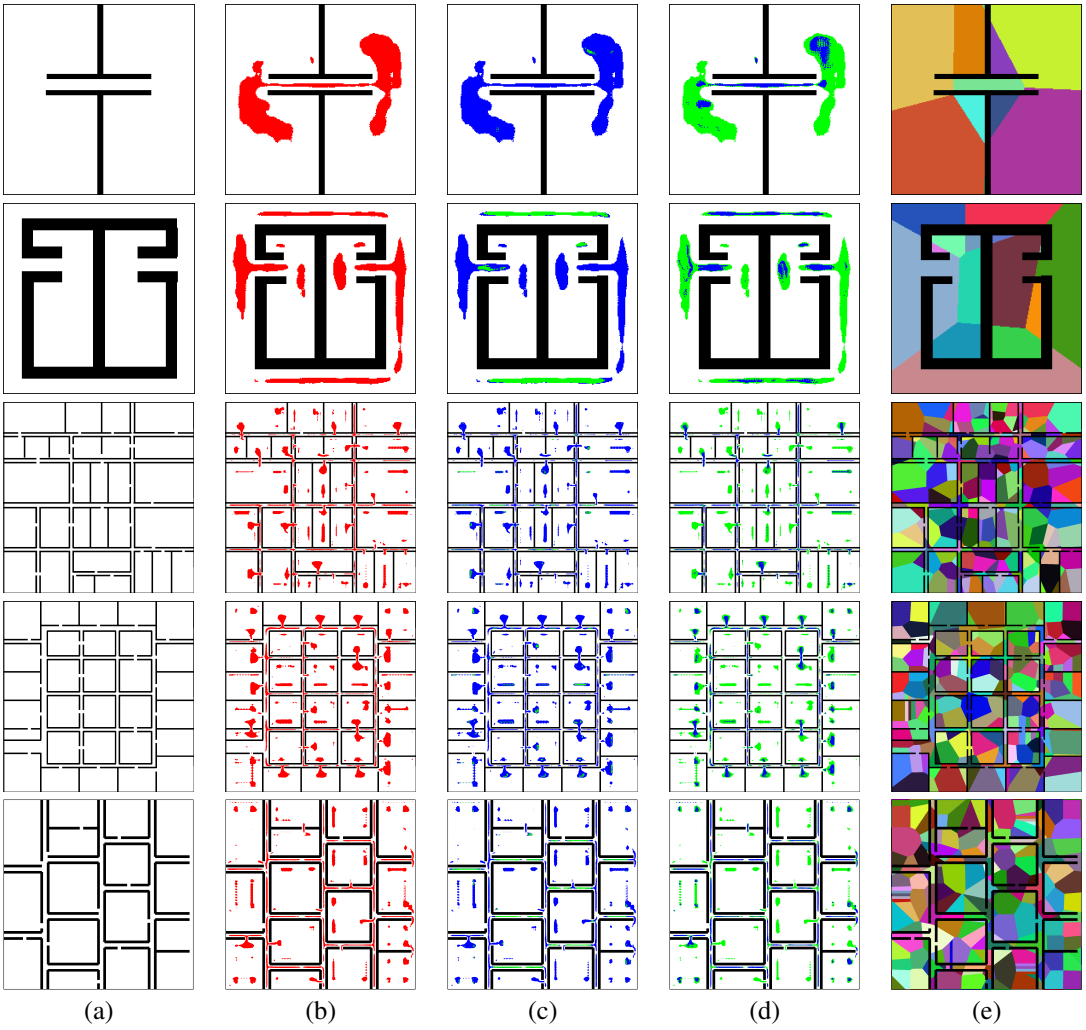


Figure 9: Predicted critical regions and generated state abstractions for a 4-DOF hinged robot. (a) Input to the environment. (b) Critical regions for the location of the robot's base link in the workspace. (c) Critical regions for the orientation of the the robot's base link. Blue regions are locations where the network predicted the robot to be horizontal and green regions are the regions where the network predicted the robot to be vertical. (d) Critical regions for the hinge joint. Blue regions show that the network predicted it to be closer to $180°$ and green regions show that the network predicted the hinge angle close to $90°$ or $270°$. (e) 2D projections of state abstraction generated by our approach. State abstractions are strictly for visualization as our approach does not require to generate this explicitly.

F.3 8-DOF FETCH ROBOT

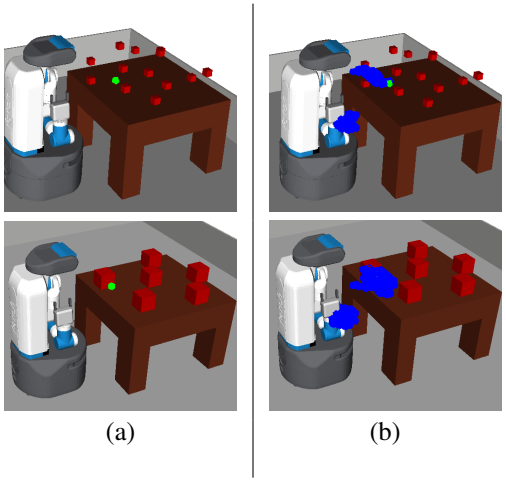


Figure 10: Predicted critical regions for an 8-DOF Fetch robot. The green region in (a) shows the goal location for the end effector. (b) shows the critical regions generated by the learned model. Although the network predicts critical regions for all the joints, only critical regions or end-effector's location in the workspace are shown.