# Learning to Create Abstraction Hierarchies for Motion Planning under Uncertainty

**Naman Shah** , **Siddharth Srivastava**
Arizona State University, Tempe, AZ, USA
{shah.naman, siddharths}@asu.edu,

## Abstract

State and action hierarchies have been found to be invaluable in long-horizon robot motion planning. However, approaches for learning such hierarchies tend to require extensive experience on the target task, target environment and/or deterministic dynamics. This paper considers the problem of learning how to create state and action abstractions for a known robot with stochastic low-level controllers in previously unseen environments. We present a novel and robust approach for learning to create an abstract, searchable state space, high-level options, as well as low-level option policies in this setting. We show that this approach facilitates efficient hierarchical planning in stochastic settings with strong guarantees of composability and completeness for holonomic robots. Extensive empirical analysis with holonomic as well as non-holonomic robots on a total of 60 different combinations unseen environments and tasks shows that the resulting approach is broadly applicable, scales well and enables effective learning and transfer even in tasks with long horizons where baselines are unable to learn.

## 1 Introduction

State and action abstractions have been shown to be vital in solving long-horizon robot planning problems. Approaches for practical robot planning typically require such abstractions to be provided as input. However, designing such abstractions tends to be expensive and time-consuming because (1) abstractions have to be domain-specific, and (2) they need to be designed while considering the range of the robot's feasible movements as well as the potential tasks of interest. These considerations limit the scope and applicability of robot planning in more realistic dynamic settings with changing tasks and requirements. Additionally, it is not clear how to design such abstractions for stochastic robot planning problems given the non-intuitive nature of actions in stochastic settings.

This paper presents a novel approach that learns to create abstract states and actions and then creates them autonomously during motion planning on new tasks and environments. Our approach proceeds by first carrying out robot-specific learning on training environments and tasks. After this phase, which is akin to a robot-specific algorithmic design process, the approach works like any other motion planner except that it dynamically utilizes the results of learning to solve arbitrary stochastic motion planning problems with the same robot. More precisely, given an unseen task in an unseen stochastic environment, it automatically creates state abstractions and action abstractions on-the-fly and uses them to solve the input stochastic motion planning problem. Empirical results show that *even when accounting for the time taken to create abstractions, the resulting approach vastly outperforms prior approaches in terms of solution time as well as solution quality in stochastic environments*.

While prior work has led to immense progress on related problems such as identifying such state or action abstractions for deterministic problems [34, 6, 48], for short-horizon control problems [36, 57, 18], and for learning *environment and task-specific abstractions* [37, 5] (a greater discussion is presented in Sec. 5 and Appendix A), this paper presents the first data-driven approach for learning to create both state abstractions and action abstractions, for robot motion planning in stochastic settings.

Rather than learning an abstraction for a given task or for a given set of high-level actions, the presented approach learns how to create abstractions independent of target tasks and environments. It transfers the learned knowledge by creating on-the-fly abstractions for unseen tasks in unseen environments. In this way, it effectively incorporates sub-symbolic learning with symbolic planning and reasoning to learn from experience while providing strong efficiency, robustness, and generalizability.

We show that the abstract actions (expressed as options [55]) generated through this approach have three desirable characteristics: (1) they are composable, i.e., they enable high-level search algorithm for computing abstract plans, (2) they are downward refinable for holonomic robots, and (3) they can be zero-shot transferred to different tasks in the same environment without any additional learning. This constitutes the first known approach for using learning to automatically transform an input stochastic robot planning problem into a high-level search problem with auto-generated abstract states and options.

We also present a formal analysis presenting the sufficient conditions under which we can assert desirable properties

of our approach such as completeness. However, empirical results indicate that these conditions may be more conservative than necessary in practice. We extensively evaluate our approach with three different robots in a total of 7 different environments and 60 different tasks in *stochastic* settings. Our empirical evaluation confirms that the learned options are effective when combined with a high-level planning algorithm, and they can be zero-shot transferred to new tasks without any additional training. It also shows that the learned abstractions scale up robot planning in stochastic settings to much larger problems compared to known approaches.

The rest of the paper is organized as follows: Sec. 2 provides background on stochastic motion planning; Sec. 3 describes our approach; Sec. 4 presents an extensive empirical evaluation of our approach; Sec. 5 briefly discusses closely related methods (Appendix A presents detailed discussion).

## 2 Background

Let $\mathcal{X} \subseteq \mathbb{R}^d = \mathcal{X}_{\text{free}} \cup \mathcal{X}_{\text{obs}}$ be the configuration space of a robot $R$ and let $O$ be a set of obstacles in a given environment. Given a collision function $f : \mathcal{X} \to \{0, 1\}$, $\mathcal{X}_{\text{free}}$ represents the set of configurations that are not in collision with any obstacle $o \in O$ such that $f(x) = 0$ and let $\mathcal{X}_{\text{obs}} = \mathcal{X} \setminus \mathcal{X}_{\text{free}}$. Let $x_i \in \mathcal{X}_{\text{free}}$ be the initial configuration of the robot and $x_g \in \mathcal{X}_{\text{free}}$ be the goal configuration of the robot. The motion planning problem can be defined as:

**Definition 1.** *A **motion planning problem** $\mathcal{M}$ is defined as a 4-tuple $\langle \mathcal{X}, f, x_i, x_g \rangle$, where $\mathcal{X}$ is the configuration space, $f$ is the collision function, $x_i$ and $x_g$ are initial and goal configurations.*

A solution to a motion planning problem is a motion plan $\tau$. A motion plan is a sequence of configurations $\langle x_0, \ldots, x_n \rangle$ such that $x_0 = x_i$, $x_n = x_g$, and $\forall x \in \tau, f(x) = 0$. Robots use controllers that accept sequenced configurations from the motion plan and generate controls that take the robot from one configuration to the next configuration. In practice, environment dynamics can be noisy, which introduces stochasticity in the problem. We define the stochastic motion planning (SMP) problem as a variant of stochastic shortest path problems (SSPs) [8]. We define a stochastic motion planning problem as $P = \langle \mathcal{X}, \mathcal{U}, T, x_0, x_g \rangle$ where $\mathcal{X} \subseteq \mathbb{R}^d$ is a $d$-dimensional configuration space. $\mathcal{U} \subseteq \mathbb{R}^d$ is the uncountably infinite set of stochastic control actions defined in terms of the intended change in each degree of freedom of the robot. Each $u \in \mathcal{U}$ follows a stochastic transition function $T_u : x \mapsto \mu(x + u)$ where $\mu(x + u)$ is a probability measure parameterized using the intended target $x + u$ of the control action. $x_0$ is the initial configuration and $x_g$ is the goal configuration. A solution to a stochastic motion planning problem is a partial policy $\pi : \mathcal{X} \to \mathcal{U}$ that maps each reachable configuration in the configuration space (when starting with $x_0$ and following $\pi$) to a control action from the set of controls (actions) $\mathcal{U}$.

## 3 Our Approach

The central idea of this paper is to learn to automatically create abstractions for unseen tasks and environments and efficiently use them to compute policies for long-horizon stochastic motion planning problems under sparse reward. We propose a novel hierarchical approach -- **s**tochastic **h**ierarchical **a**bstraction-guided **r**obot **p**lanner (SHARP) -- for stochastic motion planning (Sec. 3.3). SHARP first learns to create state abstractions (Sec. 3.1) and uses it to automatically identify state abstractions in an unseen environment. SHARP uses these identified state abstractions to identify task-independent action abstractions in this environment (Sec. 3.2).

### 3.1 Learning to Create State Abstractions

We build upon prior work to learn how to create state abstractions as follows. We use critical regions [40] to learn state abstractions for a given configuration space. Intuitively, critical regions generalize the concept of hubs as well as bottlenecks in the configuration space: regions that are helpful in solving multiple tasks (e.g., the center of the room from which all locations are easily accessible) as well as those that are difficult to sample under a uniform sampling density (e.g., bottlenecks such as doors and hallways). Molina et al. [40] formally define them as follows:

**Definition 2.** *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 $r$ ($r \subseteq s_n, \forall n$).*

Our approach uses a deep neural network for learning to predict critical regions. Once this DNN is trained, our approach uses it to predict critical regions in unseen environments. We use the same framework proposed by Shah and Srivastava [48] for learning this critical region predictor. This critical region predictor can be then zero-shot transferred for different environments as well as robots with the same kinematic characteristics. Appendix B discusses learning this predictor in detail.

Once critical regions are predicted for the given environment and the robot, they are used to construct a region-based Voronoi diagram (RBVD) [48]. RBVDs partition the configuration space into multiple Voronoi cells. Shah and Srivastava [48] formally define an RBVD as follows:

**Definition 3.** *Given a configuration space $\mathcal{X}$, let $d^c$ define the minimum distance between a configuration $x \in \mathcal{X}$ and a region $\phi \subseteq \mathcal{X}$. Given a set of regions $\Phi$ and a robot $R$, a **region-based Voronoi diagram** $\Psi$ is a partitioning of $\mathcal{X}$ such that for every Voronoi cell $\psi_i \in \Psi$ there exists a region $\phi_i \in \Phi$ such that forall $x \in \psi_i$ and forall $\phi_j \neq \phi_i$, $d^c(x, \phi_i) < d^c(x, \phi_j)$ and each $\psi_i$ is connected.*

In this framework, abstract states are defined using a bijective function $\ell : \Psi \to \mathcal{S}$ that maps each Voronoi cell $\psi \in \Psi$ to an abstract state $s \in \mathcal{S}$. The RBVD $\Psi$ induces an abstraction function $\alpha : \mathcal{X} \to \mathcal{S}$ such that $\alpha(x) = s$ if and only if there exists a Voronoi cell $\psi$ such that $x \in \psi$ and $\ell(\psi) = s$. A configuration $x \in \mathcal{X}$ is said to be in the *high-level abstract state* $s \in \mathcal{S}$ (denoted by $x \in s$) if $\alpha(x) = s$. We also define a neighborhood function $\mathcal{V} : \mathcal{S} \times \mathcal{S} \to \{0, 1\}$ such that for

a pair of states $s_1, s_2 \in \mathcal{S}$, $\mathcal{V}(s_1, s_2) = 1$ iff $s_1$ and $s_2$ are neighbors.

## 3.2 Learning to Create Action Abstractions

Given a stochastic motion planning problem, we first create state abstractions using the learned environment-independent abstraction framework presented above in Sec. 3.1. Once abstract states are constructed, we learn abstract actions as options [55] that go from one abstract state to another. These action abstractions are task-independent, i.e., they are constructed once per environment and robot and reused for different tasks (pairs of initial and goal configurations). One critical observation we make is that learning policies for these options can be challenging in sparse-reward settings. Therefore, we introduce a concept of *option guides* for constructing a dense pseudo-reward function.

We define an option $o$ as a 4-tuple $\langle \mathcal{I}_o, \beta_o, \mathcal{G}_o, \pi_o \rangle$ where $\mathcal{I}_o \subseteq \mathcal{X}$ is an initiation set, $\beta_o \subseteq \mathcal{X}$ is a termination set, $\mathcal{G}_o$ is an *option guide*, and $\pi_o$ is a policy. For learning useful options, we first compute option endpoints $\langle \mathcal{I}_o, \beta_o \rangle$ (Sec. 3.2.1) and then use them to generate option guides (Sec. 3.2.2).

### 3.2.1 Identifying Option Endpoints

Given a set of critical regions $\Phi$ and a corresponding RBVD $\Psi$ that induces a set of abstract states $\mathcal{S}$, a neighborhood function $\mathcal{V}$, and an abstraction function $\alpha$, we define two types of task-independent options: (1) Options between centroids of two critical regions -- *centroid options* and (2) options between interfaces of two pairs of abstract states – *interface options*.

First, we define centroid options. Intuitively, these options define abstract actions that transition between a pair of critical regions. Formally, we defined them using centroid regions as follows:

**Definition 4.** *Let $s_i \in \mathcal{S}$ be an abstract state in the RBVD $\Psi$ with the corresponding critical region $\phi_i \in \Phi$. Let $d$ be the Euclidean distance measure and let $t$ define a threshold distance. Let $c_i$ be the centroid of the critical region $r_i$. A* ***centroid region*** *of the critical region $r_i$ with the centroid $c_i$ is defined as a set of configuration points $\{x | x \in s_i \wedge d(x, c_i) < t\}$.*

We use this definition to define the endpoints for the centroid options as follows:

**Definition 5.** *Let $s_i, s_j \in \mathcal{S}$ be a pair of neighboring abstract states such that $\mathcal{V}(s_i, s_j) = 1$ in an RBVD $\Psi$ constructed using the set of critical regions $\Phi$. Let $\phi_i, \phi_j \in \Phi$ be the critical regions for the abstract states $s_i$ and $s_j$ and let $c_i$ and $c_j$ be their centroids regions. The* ***endpoints for a centroid option*** *are defined as a pair $\langle \mathcal{I}_{ij}, \beta_{ij} \rangle$ such that $\mathcal{I}_{ij} = c_i$ and $\beta_{ij} = c_j$.*

Now, we define interface options. Intuitively, these options define high-level temporally abstracted actions between intersections of pairs of abstract states and take the robot across the high-level states. To construct interface options, we first need to identify interface regions between a pair of neighboring abstract states. We define interface region as follows:

**Definition 6.** *Let $s_i, s_j \in \mathcal{S}$ be a pair of neighboring states such that $\mathcal{V}(s_i, s_j) = 1$ and $\phi_i$ and $\phi_j$ be their corresponding critical regions. Let $d^c(x, \phi)$ define the minimum Euclidean distance between configuration $x \in \mathcal{X}$ and some point in a region $\phi \subset \mathcal{X}$. Let $p$ be a configuration such that $d^c(p, \phi_i) = d^c(p, \phi_j)$ that is, $p$ is on the border of the Voronoi cells that define $s_i$ and $s_j$. Given the Euclidean distance measure $d$ and a threshold distance $t$, an* ***interface region*** *for a pair of neighboring states $(s_i, s_j)$ is defined as a set $\{x | (x \in s_i \vee x \in s_j) \wedge d(x, p) < t\}$.*

We use this definition of interface regions to define endpoints for the interface options as follows:

**Definition 7.** *Let $s_i, s_j, s_k \in \mathcal{S}_\Psi$ be abstract states in the RBVD $\Psi$ such that $\mathcal{V}(s_i, s_j) = 1$ and $\mathcal{V}(s_j, s_k) = 1$. Let $\hat{\phi}_{ij}$ and $\hat{\phi}_{jk}$ be the interface regions for pairs of high-level states $(s_i, s_j)$ and $(s_j, s_k)$. The* ***endpoints for an interface option*** *are defined as a pair $\langle \mathcal{I}_{o_{ijk}}, \beta_{o_{ijk}} \rangle$ such that $\mathcal{I}_{o_{ijk}} = \hat{\phi}_{ij}$ and $\beta_{o_{ijk}} = \hat{\phi}_{jk}$.*

Now given the learned state abstraction in the form of identified critical regions $\Phi$ and the RBVD $\Psi$, our approach synthesizes endpoints for a set of centroid or interface options as follows: Recall that the RBVD $\Psi$ induces a neighborhood function $\mathcal{V} : \mathcal{S} \times \mathcal{S} \rightarrow \{0, 1\}$. Our approach synthesizes a set of endpoints for centroid options $\mathcal{O}_c = \{o_{ij} | \forall s_i, s_j \in \mathcal{S}, \mathcal{V}(s_i, s_j) = 1 \wedge \mathcal{I}_{ij} = c_i \wedge \beta_{ij} = c_j\}$ or the set of interface options as $\mathcal{O}_i = \{o_{ijk} | \forall s_i, s_j, s_k \in \mathcal{S}, \mathcal{V}(s_i, s_j) = 1 \wedge \mathcal{V}(s_j, s_k) = 1 \wedge \mathcal{I}_{ij} = \hat{\phi}_{ij} \wedge \beta_{ij} = \hat{\phi}_{jk}\}$ where $c_i$ represents the centroid of the critical region $r_i$ for the abstract state $s_i$ and $\hat{\phi}_{ij}$ represents an interface region for a pair of neighboring abstract states $s_i$ and $s_j$.

We now discuss option guides and our approach for constructing them for a set of option endpoints.

### 3.2.2 Constructing Option Guides

We use option guides for synthesizing a dense pseudo-reward function for improving sample efficiency while learning policy for an option in sparse reward settings. We define an option guide as an *$\epsilon$-clear motion plan*. A motion plan is an $\epsilon$-clear motion plan if, for every configuration in the motion plan, there exists a collision-free $\epsilon$-neighborhood.

**Definition 8.** *Let $\mathcal{X}$ be the C-space of the robot $R$. Given an option $o_i$ with endpoints $\langle \mathcal{I}_i, \beta_i \rangle$, let $c_{\mathcal{I}_i}$ and $c_{\beta_i}$ define centroids for the $\mathcal{I}_i$ and $\beta_i$ respectively. Given a threshold distance $t$, an arbitrary neighborhood radius $\epsilon$, and the Euclidean distance measure $d$, an* ***option guide*** *$\mathcal{G}_i = \langle p_1, \ldots, p_n \rangle$ for an option $o_i$ is a $\epsilon$-clear motion plan such that $p_1 = c_{\mathcal{I}_i}$, $p_n = c_{\beta_i}$, for each pair of consecutive points $p_i, p_j \in \mathcal{G}_i$, $d(p_i, p_j) < t$, and for every point $p_i \in \mathcal{G}_i$, $p_i \in \alpha(\mathcal{I}_i)$ or $p_i \in \alpha(\beta_i)$.*

Here, we abuse the notation and use the abstraction function with a set of low-level configurations rather than a single configuration such that for a set $A$, $\alpha(A) = \{\alpha(x) | \forall x \in A\}$. In practice, we find that any sampling-based motion planner with $\epsilon$-inflated obstacles can be used to construct option guides for identified option endpoints. We use HARP [48] and $\epsilon = 0$ for our experiments (Sec. 4).

Our approach uses this option guide to automatically formalize a dense pseudo-reward function for learning option policy. This pseudo-reward function provides the robot with

**Algorithm 1:** Stochastic Hierarchical Abstraction-guided Robot Planner (SHARP)

**Input:** Training environments $E_{\text{train}}$, test environment $e_{\text{test}}$, initial and goal configurations $x_i$ and $x_g$
**Output:** A policy $\Pi$ composed of options

1 $\Theta \leftarrow$ get_critical_regions_predicter();
2 **if** $\Theta$ *is not trained* **then**
3 train $\Theta$ using $E_{\text{train}}$
4 **if** *abstraction is **not** constructed* **then**
5 $\Phi \leftarrow$ predict_critical_regions($e_{\text{test}}$,$\Theta$);
6 $\Psi, \mathcal{S}, \mathcal{V} \leftarrow$ construct_RBVD($e_{\text{test}}$,$\Phi$);
7 $\mathcal{O}, C \leftarrow$ synthesize_options($\Phi$,$\Psi$,$\mathcal{S}$,$\mathcal{V}$);
8 $s_i, s_g \leftarrow$ get_abstract_states($x_i$,$x_g$);
9 **while** *not refined* **do**
10 $p \leftarrow$ high_level_plan($s_i$,$s_g$,$\mathcal{O}$,$C$);
11 **if** $p = \emptyset$ **then**
12 break;
13 $\Pi$ = empty_list;
14 $\pi_0 \leftarrow$ ll_policy($x_i$,$\mathcal{I}_{o_1}$);
15 $\Pi$.add($\pi_0$);
16 **foreach** $o \in p$ **do**
17 **if** $\pi_o$ *does **not** exist* **then**
18 $\mathcal{G}_o \leftarrow$ option_guide($\mathcal{I}_o$,$\beta_o$);
19 **if** $\mathcal{G}_o = \emptyset$ **then**
20 flag $o$ infeasible;
21 break;
22 $\pi_o \leftarrow$ ll_policy($\mathcal{I}_o$,$\beta_o$,$\mathcal{G}_o$);
23 adjust the option cost $C_o$;
24 $\Pi$.add($\pi_o$);
25 refined $\leftarrow$ True;
26 **if** *refined* **then**
27 $\pi_{n+1} \leftarrow$ ll_policy($\beta_{o_n}$,$x_g$);
28 $\Pi$.add($\pi_{n+1}$);
29 return $\Pi$;
30 **else**
31 return failure;

a large positive reward when it reaches the termination set of the goal, a penalty for drifting to a different abstract state, and a smoothened reward for making progress on the option guide. Appendix C explains this in detail. The next section uses these concepts to present our approach for computing a policy for a stochastic motion planning problem.

### 3.3 The SHARP Algorithm for Motion Planning Under Uncertainty

SHARP (Alg. 1) uses learned abstractions for motion planning under uncertainty. Given an SMP problem $P = \langle \mathcal{X}, \mathcal{U}, x_i, x_g \rangle$, it starts with a simulator and an occupancy matrix of the environment and computes a partial policy $\Pi : \mathcal{X} \rightarrow \mathcal{U}$ that maps each reachable state to a control action.

Alg. 1 start with using the learned critical region predictor and identifies a set of critical regions $\Phi$ in the given test environment $e_{\text{test}}$ (line 5). It then constructs an RBVD (Def. 3) $\Psi$ using the predicted critical regions $\Phi$ (line 6). This RBVD $\Psi$ induces a set of abstract states $\mathcal{S}$, an abstraction function $\alpha$, and a neighborhood function $\mathcal{V}$. Alg. 1 uses these critical regions $\Phi$ and RBVD $\Psi$ to synthesize a set of endpoints for centroid or interface options.

Alg. 1 uses these options as high-level actions with A$^*$ search for computing high-level plans. It considers the initiation and termination sets of these options as their preconditions and effects. Alg. 1 uses an incremental plan generator that takes the set of options along with the abstract initial and goal states as input and generates a high-level plan using A$^*$ search (line 10). It uses the Euclidean distance between the termination set of the option and the goal configuration as the heuristic and the Euclidean distance between the initiation and termination sets as an initial approximation to the cost of the option.

Given a plan in the form of a sequence of options, Alg. 1 starts refining these options by computing policies for every option in the sequence. First, it generates an additional option $o_0$ such that $\mathcal{I}_{o_0} = x_i$ and $\beta_{o_0} = \mathcal{I}_{o_1}$ and learns its policy (line 14). Alg. 1 then starts computing policies for each option in the high-level plan. If a policy exists for the option from the previous invocation of the algorithm, then it uses the same policy. Otherwise, it starts computing a policy for it. However, before computing a policy using reinforcement learning, Alg. 1 computes an option guide (Def: 8) as described in Sec. 3.2.2 (line 18). If it fails to compute a valid option guide for an option then we mark the option as infeasible and compute a new high-level plan from the initial abstract state (line 19). Once an option guide is computed for an option, Alg. 1 uses an off-the-shelf low-level policy learner to learn a policy for it (line 22). After computing (or reusing) policies for all the options in the plan, it again generates an option $o_{n+1}$ such that $\mathcal{I}_{o_{n+1}} = \beta_{o_n}$ and $\beta_{o_{n+1}} = x_g$ and learns its policy (line 27).

Alg. 1 only synthesizes options once per each environment and robot. To efficiently transfer the learned option policies, our approach needs to update the option costs that A$^*$ search uses to compute the sequence of options. We update this cost (line 20) by collecting rollouts of the learned policy and using the average number of steps from the initiation set to the termination set as an approximation of the cost of the option. We now prove that Alg. 1 is probabilistically complete.

Finally, Alg. 1 computes a *composed policy* by composing policies for every option in this high-level plan. A **composed policy** $\Pi$ for a high-level plan is a finite state automaton with one controller state for each option in the plan. As mentioned earlier, Alg. 1 computes $\pi_0$ and $\pi_{n+1}$ as special cases (lines 12 and 22). For a controller state $q_i$, $\Pi(x) = \pi_i(x)$ where $\pi_i$ represents the policy for option $o_i \in \mathcal{O}$. The controller makes a transition $q_i \rightarrow q_{i+1}$ when the robot reaches a configuration $x \in \mathcal{I}_{o_{i+1}}$.

### 3.4 Theoretical Results

We now present theoretical properties of Alg. 1. Let $B_\delta(x)$ for $\delta > 0$ define the $\delta$-neighborhood of $x \in \mathcal{X}$ under the Euclidean metric. Recall that each controller implicitly defines as a transition function with a probability distribution $\mu(x+u)$ for the control action $u$ (see Sec. 2). A *$\delta$-compliant controller*

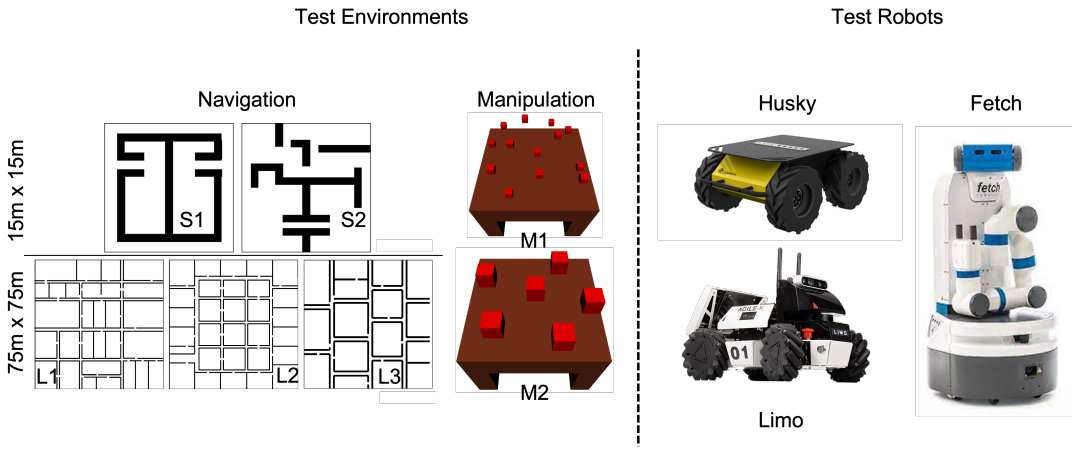



Figure 1: Test environments and robots used to evaluate our approach.

is defined as for which the set of support of one whose set of support for $\mu(x+u)$ is $B_\delta(x+u)$. Our formal guarantees do not require knowledge of $\mu$ other than an upper bound on the support radius. Here, we refer to $\delta$ as the *support radius* for the given controller.

**Theorem 3.1.** *For a given stochastic motion planning problem $P = \langle \mathcal{X}, \mathcal{U}, x_1, x_n \rangle$, let $\Phi$ be the set of identified critical regions and $\Psi$ be the RBVD that induces the set of abstract state $\mathcal{S}$ and a neighborhood function $\mathcal{V}$. If there exists a sequence of distinct abstract states $\langle s_1, \ldots, s_n \rangle$ such that $\mathcal{V}(s_i, s_{i+1}) = 1$ then there exists a composed policy $\Pi$ such that the resulting configuration after the termination of every option in $\Pi$ would be the goal configuration $x_n$.*

*Proof.* (Sketch) The proof directly derives from the definition of the endpoints for the centroid and interface options. Given a sequence of adjacent abstract states $\langle s_1, \ldots, s_n \rangle$, Def. 5 and 7 ensures a sequence of options $\langle o_1, \ldots, o_n \rangle$ such that $\beta_i = \mathcal{I}_{i+1}$. This implies that an option can be executed once the previous option is terminated. Given this sequence of options $\langle o_1, \ldots, o_n \rangle$, according to the definition of the composed policy, there exists a composed policy $\Pi$ such that for every pair of options $o_i, o_j \in \Pi, \mathcal{I}_{o_j} = \beta_{o_i}$. Thus, we can say that if every option in $\Pi$ terminates, then the resulting configuration would be the goal configuration. □

**Theorem 3.2.** *Given a stochastic motion planning problem $P = \langle \mathcal{X}, \mathcal{U}, x_i, x_g \rangle$ for a holonomic robot $R$ using a controller with a support radius $\delta_c < \delta$, a motion planner that can compute $\delta$-clear motion plans, and an optimal low-level policy learner, if there exists a $\delta$-clear motion plan for the robot $R$ from $x_1$ to $x_n$ that forms a sequence of distinct abstract states, then Alg. 1 will find a proper policy for the given stochastic motion planning problem.*

*Proof.* (Sketch) Let $T = \langle x_i, \ldots, x_g \rangle$ be the $\delta$-clear motion plan from the initial configuration $x_i$ to goal configuration $x_g$. This $\delta$-clear motion plan forms a non-repeating sequence of abstract states. Let $p = \langle s_1, \ldots, s_n \rangle$ be this sequence of distinct abstract states. Given that Alg. 1 explores all possible sequences of high-level states between a given pair of initial and goal abstract states (line 10), we can say that eventually, it would find this sequence of abstract states $p$ and the corresponding sequence of options for it. We can also deduce that for every pair of consequent abstract states $s_j, s_{j+1} \in p$, there exists a pair of consequent configurations $x_j, x_{j+1} \in T$ such that $x_j \in s_j$ and $x_{j+1} \in s_{j+1}$ and $\mathcal{V}(s_j, s_{j+1}) = 1$ and that there exists $\delta$-clear motion plan between abstract states $s_j$ and $s_{j+1}$. Now, lemmas D.1 and D.2 (provided in Appendix D) show that given a motion planner that computes $\delta$-clear motion plan and an optimal low-level policy learner, Alg. 1 would be able to learn options with proper policies for every pair of neighboring states. This implies that our approach would be able to learn options for each pair of consequent states in $p$. Lastly, Theorem 3.1 proves that if there exists a sequence of distinct abstract states then there exists a composed policy of learned options that when executed successfully in $x_i$ terminates in $x_g$ i.e., a solution for the given problem. □

These results provide the foundations for analyzing such approaches and show a completeness result for the presented approach. However, our approach generalizes beyond the sufficient (and not necessary) conditions used in the theorems above. In fact our empirical evaluation (Sec. 4) is conducted on non-holonomic robots that violate the premises of these results. Furthermore, we use default controllers with unknown support radii.

## 4 Empirical Results

We present the salient aspects of our implementation, setup, and observations here; additional results, code, and data are available with supplementary material.

**Implementation details** We implemented our approach using PyBullet and PyTorch [43] and conducted an extensive empirical evaluation. PyBullet does not explicitly model stochasticity in the movement of the robot. Therefore, we use random perturbations in intended targets of control actions to introduce stochasticity in the environment while training and using the default controllers to evaluate the learned policies.

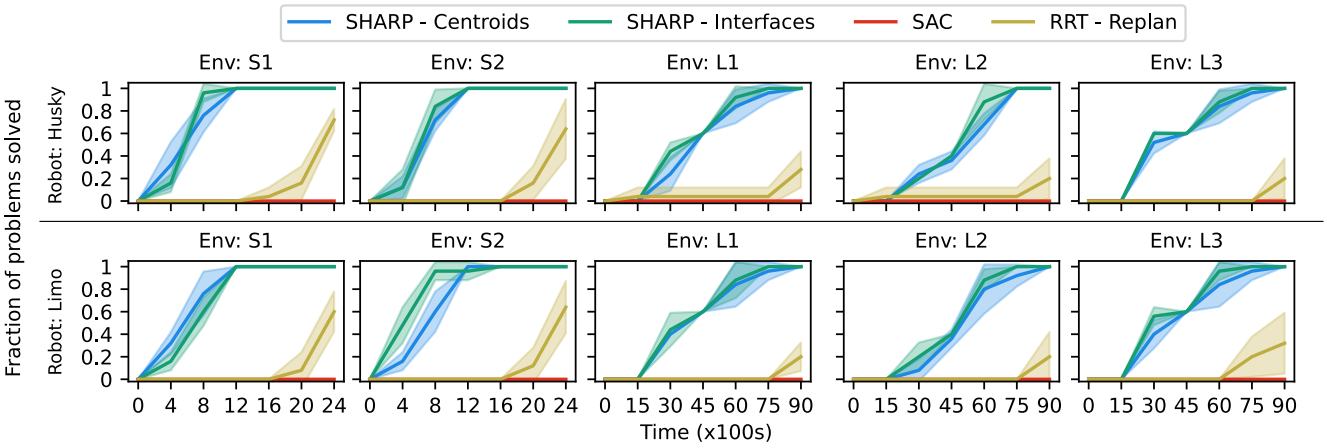



Figure 2: (Higher values are better) Times taken by our approach (SHARP) and baselines to compute path plans in the test environment. The x-axis shows the time and the y-axis shows the fraction of the test tasks solved in the given time. The numbers are averaged over 5 independent trails.

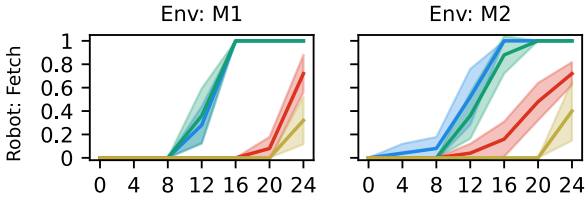



Figure 3: (Contd. from Fig. 2 with same setup) Results for manipulation tasks with the Fetch robot.

We used $2-$layered neural networks with 256 neurons in each layer for representing local policies for the learned options. Inputs to these networks were the current configuration of the robot and a vector to the nearest point on the option guide for the current option. We used +1000 as a pseudo reward for reaching the termination set of each option and -100 as a penalty for drifting to a different abstract state. We use SAC [17] as a low-level policy learner.

**Test environments and robots** We evaluated our approach using a total of 7 test environments (Fig: 1) (not included in training the critical region predictor) using 3 different non-holonomic robots (Fig: 1) in a total of 60 navigation and manipulation tasks. Dimensions of the first two environments (S1, S2) were $15m \times 15m$. The rest of the environments (L1, L2, L3) were of the size $75m \times 75m$. For each environment, we generated 5 test tasks by randomly sampling different initial and goal configuration pairs. We used the following robots: the ClearPath Husky robot, the AgileX Limo robot, and the Fetch manipulator robot. The Husky is a 4-wheeled differential drive robot that can move in one direction and rotate in place; the Limo is also a 4-wheeled omnidirectional robot with an Ackermann dynamics; the Fetch is an 8-DOF manipulator robot.

**Selection of baselines** We considered and evaluated several learning and planning approaches [35, 17, 32, 38, 37, 5] as potential baselines for this work. Of these, only RRT-Replan [35] and SAC [17] solved any tasks within the timeouts (see "Evaluation framework" below). Therefore, we compared our approach against SAC and RRT-replan. SAC is an off-policy deep reinforcement learning approach and learns a single policy for the overall stochastic motion planning problem. We used the same network architecture as ours for SAC's neural policy. We used a terminal reward of $+1000$ and a step reward of $-1$ to train the SAC agent. RRT-replan is a version of the popular RRT algorithm [35] that recomputes a plan from the robot's current configuration if the robot fails to successfully reach the goal after executing the initial plan. All approaches considered use the same input robots, simulators, and low-level controllers as our approach.

**Evaluation framework and metrics** We evaluated the efficiency and quality of our approach using the following metrics: the fraction of test tasks solved in a given amount of time (Fig. 2 and 3); the average number of steps taken while executing the learned solutions (Fig. 4); and the success rate of computed solutions (Fig. 4). For our approach, we include the time taken to predict abstractions, build abstract actions (option end points), compute high-level plans, and learn the necessary option policies. We used pre-determined timeouts and average rewards to terminate the training/planning process for our approach and the baselines. For all the approaches, we used a timeout of 2400 seconds for small navigation environments (S1 and S2) as well as manipulation environments (M1 and M2) and 9000 seconds for large navigation environments (L1-L3). For learning-based approaches (ours and SAC), we evaluated the model for 20 episodes after every $10k$ learning steps and stopped the training if it achieved an average evaluation reward of 500 or reached 150k training steps. RRT-Replan continued its plan-execute loop until the robot reached the goal configuration or the timeout was reached.

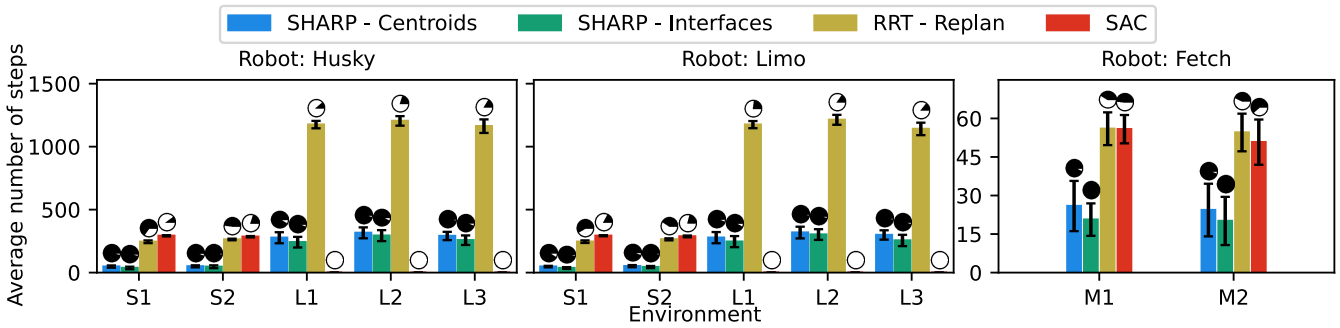



Figure 4: (Lower values or darker circles are better) Average number of steps taken in the **successful** execution of the learned policies and success rates for our approach and the baselines. These numbers do not include instances when the execution of the policy did not end at the goal configuration. The pie chart over each bar represents the success rate (shaded black area) while executing the learned policy.

| | S1 | S2 | L1 | L2 | L3 | M1 | M2 |
|---|---|---|---|---|---|---|---|
| Interface Options | 43% | 33% | 37% | 33% | 42% | 50% | 75% |
| Centroid Options | 50% | 50% | 39% | 36% | 50% | 65% | 75% |

Figure 5: Percentage of options that our approach reused from the task they were computed to every subsequent task they were needed across 5 test tasks in each environment.

### 4.1 Analysis of Results

We thoroughly evaluated our approach for answering two critical questions: (1) Does this approach learn useful high-level planning representations? (2) Can these learned options be zero-shot transferred to new tasks in the same environment? Our experiments show that the presented approach effectively generates hierarchies and significantly outperforms all baselines across all environments. In larger environments (L1-L3), the presented approach is the only approach that is able to show significant learning, and it achieves a significantly higher success rate than all baselines.

**Learning to predict abstractions** One of the key contributions of this paper is to be able to learn to predict abstractions from experience in training environment and zero-short transfer them in the form of a predictor to identify critical regions in unseen environments. Appendix E shows the predicted critical regions, 2D projections of the RBVDs, and synthesized option endpoints for our test environments. These automatically identified abstractions show that our approach is able to zero-short transfer the learned critical region predictor to new unseen test environments and identify useful abstract states and option endpoints without any additional training.

**Improved learning and planning performance** We evaluate the efficiency of our approach using the total time taken to produce a policy starting from the common input given to all baselines. In our case, this includes the time taken to create the abstract states and actions as well as to compute the solutions. Figs. 2 and 3 show the fraction of tasks solved in a given amount of time. Each subsequent task uses learned high-level actions (policies and options) from the previous tasks when available. Results show that our approach was able to learn policies significantly faster than the baselines. In most cases, our approach was able to compute solutions in less than half of the time taken by the baselines. These results illustrate the impact of learning to create and utilize abstractions: even when the time for predicting critical regions, building abstractions, computing high-level plans, and learning low-level policies is included, SHARP significantly outperforms the baselines.

**Improved solution quality** We evaluate the quality of computed solutions using the average number of steps taken while executing the policies, as well as per-task success rates of the computed solution policies (Fig 4). These results are averaged over 20 independent executions of each learned policy. These results show that SHARP learns policies that require significantly fewer steps during execution compared to baselines. Furthermore, SHARP's solutions have a success rate of more than 90% across all tasks. On the other hand, the best performing baseline, RRT-replan has a success rate of only $\sim 50\%$ in the smaller navigation (S1, S2) and manipulation (M1, M2) tasks and a success rate of less than one-third in the larger navigation (L1-L3) tasks. SAC only solved 20% of the robot navigation tasks and 60% of the manipulation tasks.

**Zero-shot action transfer across tasks** Lastly, we also conducted a thorough analysis of options being used across all tasks in each environment. Fig. 5 shows the fraction of options reused from the task they were computed to all the subsequent tasks they were needed by our approach. These reuse rates combined with the success rates show that our approach is able to successfully zero-shot transfer learned options across new tasks.

## 5 Related Work

In this section, we briefly discuss some related approaches. Here we focus on broad classifications of these approaches and include a detailed discussion in Appendix A.

To the best of our knowledge, this is the first data-driven approach that learns to identify a discrete set of abstract states and plannable task-independent abstract actions automatically for unseen environments with stochastic dynamics. In this paper, we exclusively focus on solving long-horizon motion planning problems under uncertainty.

Several other approaches have been developed to solve

similar class of problems. These approaches can be broadly classified as follows. Approaches for stochastic motion planning [11, 33, 59, 22, 7, 20] require an analytical dynamics model of the robot. In practice, such analytical models may not be available. Another approach is to learn task-specific subgoals in the given test environment [32, 4, 41, 42, 10]. In contrast to our approach, which learns to predict state and action abstractions without any experience collected from the test environment, these approaches require interactions with the environments in order to learn useful subgoals. Finally, approaches for learning task-specific options [53, 52, 9, 13, 5, 6] learn options for specific tasks while interacting in the target environment. In contrast, our approach conducts transferrable, task and environment-independent learning, permits theoretical analysis, and exhibits strong transfer and generalizability across tasks and environments.

Finally, there has been a lot of progress on short-horizon ($\sim 5$ seconds) dense-reward problems where the robot receives frequent feedback for its actions from the environment. These approaches include conventional control approaches as well as image-based DRL approaches for model predictive control [60, 36, 14, 15, 19, 57, 34, 12, 2, 18]. While this paper's focus is on long-horizon sparse-reward motion planning problems, these approaches can be used as low-level policy learners in our approach (Alg. 1, line 22).

## 6 Conclusions, Limitations, and Future Work

This paper presents the first approach that uses a data-driven process to learn to create state and action abstractions for unseen environments and tasks. We provide theoretical results as well as a thorough empirical evaluation for the presented methods. These results show that the presented approach effectively learns to create abstractions that provide strong performance and quality advantages on a broad set of tasks that are currently beyond the scope of known methods.

One of the main limitations of the presented work is that like much of contemporary AI research, it requires a simulator of the environment and it has only been tested in simulated settings. However, advances from independent and active research threads on sim2real approaches would benefit these approaches and allow the learned policies to be executed in the real world. Another limitation of the presented approach is that it does not currently address the problem of learning abstractions for task and motion planning problems [49, 16], which can be a promising direction for future work.

## Acknowledgements

We thank Kiran Prasad for his help in implementing a primitive version of the presented approach. The presented work is supported by NSF under grant IIS 1942856.

## References

[1] Ron Alterovitz, Thierry Siméon, and Ken Goldberg. The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty. In *Proc. R:SS*, 2007.

[2] Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter. Differentiable MPC for end-to-end planning and control. In *Proc. NeurIPS*, volume 31, 2018.

[3] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Proc. NeurIPS*, 2017.

[4] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proc. AAAI*, 2017.

[5] Akhil Bagaria and George Konidaris. Option discovery using deep skill chaining. In *Proc. ICLR*, 2020.

[6] Akhil Bagaria, Jason K Senthil, and George Konidaris. Skill discovery for exploration and planning using deep skill graphs. In *Proc. ICML*, 2021.

[7] Jur van den Berg, Sachin Patil, and Ron Alterovitz. Motion planning under uncertainty using differential dynamic programming in belief space. In *Robotics Research*, pages 473–490. Springer, 2017.

[8] Dimitri P Bertsekas and John N Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991.

[9] Emma Brunskill and Lihong Li. Pac-inspired option discovery in lifelong reinforcement learning. In *Proc. ICML*, 2014.

[10] Konrad Czechowski, Tomasz Odrzygóźdź, Marek Zbysiński, Michał Zawalski, Krzysztof Olejnik, Yuhuai Wu, Łukasz Kuciński, and Piotr Miłoś. Subgoal search for complex reasoning tasks. In *Proc. NeurIPS*, 2021.

[11] Yanzhu Du, David Hsu, Hanna Kurniawati, Wee Sun, Lee Sylvie, CW Ong, and Shao Wei Png. A POMDP approach to robot motion planning under uncertainty. In *Proc. ICAPS, Workshop on Solving Real-World POMDP Problems*. Citeseer, 2010.

[12] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

[13] Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In *Proc. NeurIPS*, 2019.

[14] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *Proc. ICRA*, 2016.

[15] Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. Improving PILCO with bayesian neural network dynamics models. In *Data-efficient machine learning workshop, ICML*, 2016.

[16] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.

[17] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proc. ICML*, 2018.

[18] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *Proc. ICML*, 2019.

[19] Mikael Henaff, William F Whitney, and Yann LeCun. Model-based planning with discrete and continuous actions. *arXiv preprint arXiv:1705.07177*, 2017.

[20] Michael Hibbard, Abraham P Vinod, Jesse Quattrociocchi, and Ufuk Topcu. Safely: safe stochastic motion planning under constrained sensing via duality. *arXiv preprint arXiv:2203.02816*, 2022.

[21] Robert C Holte, Maria B Perez, Robert M Zimmer, and Alan J MacDonald. Hierarchical A*: Searching abstraction hierarchies efficiently. In *Proc. AAAI*, pages 530–535, 1996.

[22] Vu Anh Huynh, Sertac Karaman, and Emilio Frazzoli. An incremental sampling-based algorithm for stochastic optimal control. *The International Journal of Robotics Research*, 35(4):305–333, 2016.

[23] Yuu Jinnai, David Abel, David Hershkowitz, Michael Littman, and George Konidaris. Finding options that minimize planning time. In *Proc. ICML*, 2019.

[24] Tom Jurgenson and Aviv Tamar. Harnessing reinforcement learning for neural motion planning. In *Proc. RSS*, 2019.

[25] Tom Jurgenson, Or Avner, Edward Groshev, and Aviv Tamar. Sub-goal trees a framework for goal-based reinforcement learning. In *Proc. ICML*, pages 5020–5030. PMLR, 2020.

[26] 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.

[27] Junsu Kim, Younggyo Seo, and Jinwoo Shin. Landmark-guided subgoal generation in hierarchical reinforcement learning. In *Proc. NeruIPS*, 2021.

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

[29] Harsha Kokel, Arjun Manoharan, Sriraam Natarajan, Balaraman Ravindran, and Prasad Tadepalli. RePReL: Integrating relational planning and reinforcement learning for effective abstraction. In *Proc. ICAPS*, 2021.

[30] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018.

[31] James J Kuffner and Steven M LaValle. RRT-Connect: An efficient approach to single-query path planning. In *Proc. ICRA*, 2000.

[32] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Proc. NeurIPS*, 2016.

[33] Hanna Kurniawati, Tirthankar Bandyopadhyay, and Nicholas M Patrikalakis. Global motion planning under uncertain motion, sensing, and environment map. *Autonomous Robots*, 33(3):255–272, 2012.

[34] Thanard Kurutach, Aviv Tamar, Ge Yang, Stuart J Russell, and Pieter Abbeel. Learning plannable representations with causal InfoGANs. In *Proc. NeruIPS*, 2018.

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

[36] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[37] Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. In *Proc. ICLR*, 2019.

[38] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *Proc. ICLR*, 2016.

[39] Daoming Lyu, Fangkai Yang, Bo Liu, and Steven Gustafson. SDRL: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *Proc. AAAI*, 2019.

[40] Daniel Molina, Kislay Kumar, and Siddharth Srivastava. Learn and link: learning critical regions for efficient planning. In *Proc. ICRA*, 2020.

[41] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Proc. NeurIPS*, 2018.

[42] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Near-optimal representation learning for hierarchical reinforcement learning. In *Proc. ICLR*, 2019.

[43] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Proc. NeurIPS*. 2019.

[44] Sujoy Paul, Jeroen Vanbaar, and Amit Roy-Chowdhury. Learning from trajectories via subgoal discovery. In *Proc. NeurIPS*, 2019.

[45] Mihail Pivtoraiko, Ross A Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

[46] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: convolutional networks for biomedical image segmentation. In *Proc. MICCAI*, 2015.

[47] Dhruv Mauria Saxena, Tushar Kusnur, and Maxim Likhachev. AMRA*: Anytime multi-resolution multi-heuristic a. In *Proc. ICRA*. IEEE, 2022.

[48] Naman Shah and Siddharth Srivastava. Using deep learning to bootstrap abstractions for hierarchical robot planning. In *Proc. AAMAS*, 2022.

[49] 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.

[50] David Silver and Kamil Ciosek. Compositional planning using optimal option models. In *Proc. ICML*, 2012.

[51] Tom Silver, Rohan Chitnis, Joshua Tenenbaum, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning symbolic operators for task and motion planning. In *Proc. IROS*, 2021.

[52] Özgür Şimşek, Alicia P Wolfe, and Andrew G Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proc. ICML*, 2005.

[53] Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *International Symposium on abstraction, reformulation, and approximation*, pages 212–223. Springer, 2002.

[54] Wen Sun, Jur van den Berg, and Ron Alterovitz. Stochastic extended LQR for optimization-based motion planning under uncertainty. *IEEE Transactions on Automation Science and Engineering*, 13(2):437–447, 2016.

[55] Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[56] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Proc. NeurIPS*, volume 29, 2016.

[57] Aviv Tamar, Garrett Thomas, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. Learning from the hindsight plan—episodic MPC improvement. In *Proc. ICRA*, 2017.

[58] Jur Van Den Berg, Sachin Patil, and Ron Alterovitz. Motion planning under uncertainty using iterative local optimization in belief space. *The International Journal of Robotics Research*, 31(11):1263–1278, 2012.

[59] Michael P Vitus, Wei Zhang, and Claire J Tomlin. A hierarchical method for stochastic motion planning in uncertain environments. In *Proc. IROS*. IEEE, 2012.

[60] Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Proc. NeurIPS*, 2015.

[61] Fangkai Yang, Daoming Lyu, Bo Liu, and Steven Gustafson. PEORL: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making. In *Proc. IJCAI*, 2018.

# A Extended Related Work

To the best of our knowledge, this is the first approach that uses a data-driven method for synthesizing transferable and composable options and leverages these options with a hierarchical algorithm to compute solutions for stochastic path planning problems. It builds upon the concepts of abstraction, stochastic motion planning, option discovery, and hierarchical reinforcement learning and combines reinforcement learning with planning. Here, we discuss related work in these areas.

Motion planning is a well-researched area. Numerous approaches [26, 35, 31, 45, 47] have been developed for motion planning in deterministic environments. Kavraki et al. [26], LaValle [35], Kuffner and LaValle [31] develop sampling-based techniques that randomly sample configurations in the environment and connect them for computing a motion plan from the initial and goal configurations. Holte et al. [21], Pivtoraiko et al. [45], Saxena et al. [47] discretize the configuration space and use search techniques such as A$^*$ search to compute motion plans in the discrete space.

**Stochastic motion planning** Multiple approaches [11, 33, 59, 22, 7, 20] have been developed for performing motion planning with stochastic dynamics. Alterovitz et al. [1] build a weighted graph called stochastic motion roadmap (SMR) inspired by the probabilistic roadmaps (PRM) [26] where the weights capture the probability of the robot making the corresponding transition. Huynh et al. [22] extend SMR for computing stochastic policies through value iteration over motion trees constructed using RRT [35]. Sun et al. [54] use linear quadratic regulator -- a linear controller that does not explicitly avoid collisions -- along with value iteration to compute a trajectory that maximizes the expected reward. However, these approaches require an analytical model of the transition probability of the robot's dynamics. Tamar et al. [56] develop a fully differentiable neural module that approximates value iteration (VI) and can be used for computing solutions for stochastic path planning problems. However, these approaches [1, 54, 56] require discretized actions. Du et al. [11], Van Den Berg et al. [58] formulate a stochastic motion planning problem as a POMDP to capture uncertainty in robot sensing and movements. Multiple approaches [24, 13, 25] design end-to-end reinforcement learning approaches for solving stochastic motion planning problems. These approaches only learn policies to solve one path-planning problem at a time and do not transfer knowledge across multiple problems. In contrast, our approach does not require discrete actions and it learns options that are transferrable to different problems.

**Subgoal discovery** Several approaches have considered the problem of learning task-specific subgoals. Kulkarni et al. [32], Bacon et al. [4], Nachum et al. [41, 42], Czechowski et al. [10] use intrinsic reward functions to learn a two-level hierarchical policy. The high-level policy predicts a subgoal that the low-level goal-conditioned policy should achieve. The high-level and low-level policies are then trained simultaneously using simulations in the environment. Paul et al. [44] combine imitation learning with reinforcement learning for identifying subgoals from expert trajectories and bootstrap reinforcement learning. Levy et al. [37] learn a multi-level policy where each level learns subgoals for the policy at the lower level using Hindsight Experience Replay (HER) [3] for control problems rather than long-horizon motion planning problems in deterministic settings. Kim et al. [27] randomly sample subgoals in the environment and use a path planning algorithm to select the closest subgoal and learn a policy that achieves this subgoal.

**Option discovery** Numerous approaches [53, 52, 9, 34, 13, 5, 6] perform hierarchical learning by identifying task-specific options through experience collected in the test environment and then use these options [55] along with low-level primitive actions. Stolle and Precup [53], Şimşek et al. [52] lay the foundation for discovering options in discrete settings. They collect trajectories in the environment and use them to identify high-frequency states in the environment. These states are used as termination sets of the options and initiation sets are derived by selecting states that lead to these high-frequency states. Once options are identified, they use Q-learning to learn policies for these options independently to formulate Semi-MDPs [55]. Bagaria and Konidaris [5] learn options in a reverse fashion. They compute trajectories in the environment that reaches the goal state. In these trajectories, they use the last $K$ points to define an option. These points are used to define the initiation set of the option and the goal state is used as a termination set. They continue to partition the rest of the collected trajectories similarly and generate a fixed number of options.

Several approaches [60, 36, 14, 15, 19, 57, 12, 2, 18] have explored vision-based model predictive control for robot planning problems. These approaches learn latent representations of the kinematic and dynamics model of the robot and use them to perform model-based control for the given robot control problem. These approaches focus on stochastic optimal control problems. In contrast, our approach focuses on relatively long-horizon robot planning problems and can be used with arbitrary controllers for short-horizon control ($\sim 5$ seconds).

**Planning with options** Approaches for combining symbolic planning with reinforcement learning [50, 61, 23, 39, 29, 30, 51] use pre-defined abstract models to combine symbolic planning with reinforcement learning. In contrast, our approach learns such options (including initiation and termination sets) as well as their policies and uses them to compute solutions for stochastic path planning problems with continuous state and action spaces.

# B Training The Critical Region Predictor

Alg. 1 first needs to identify critical regions (Def. 2) to synthesize options in the given environment. Recall that critical regions are regions in the environment that have a high density of solutions for the given class of problems but are hard to sample under uniform distribution (Def. 2). We train a deep neural network that learns to identify critical regions in a given environment using an occupancy matrix of the environment. Given a set of training environments $E_{\text{train}}$, training data for such a network can be generated by solving multiple randomly sampled motion planning problems.

These critical region predictors are environment independent, and they are also generalizable across robots to a large extent. Furthermore, the approach presented here directly used

the open-source critical regions predictors made available by Shah and Srivastava [48]. These predictors are environment independent and need to be trained only once per the kinematic characteristics of a robot. E.g., the non-holonomic robots used to evaluate our approach (details in Sec. 4) are different from those used by Shah and Srivastava [48], however, we used the critical regions predictor developed by them for a rectangular holonomic robot.

Shah and Srivastava [48] use 20 training environments ($E_{\text{train}}$) to generate the training data. For each training environment $e_{\text{train}} \in E_{\text{train}}$, they randomly sample 100 goal configurations. Shah and Srivastava [48] randomly sample 50 initial configurations for each goal configuration and compute motion plans for them using an off-the-shelf motion planner and a kinematic model of the robot. They use UNet [46] with *Tanh* activation function for intermediate layers and *Sigmoid* activation for the last layer. They use the weighted logarithmic loss as the loss function. Lastly, they use ADAM optimizer [28] with a learning rate of $10^{-4}$ and train the network for $50,000$ epochs.

## C Dense Pesudo-Reward Function

The option guide is used to define a pseudo reward function $R$ for the option. It is a dense reward function that provides a reward to the robot according to the distance of the robot from the nearest point in the option guide and the distance from the last point of the option guide. For an option $o_i$, we define the dense pseudo reward function $R_i : \mathcal{X} \to \mathbb{R}$ as follows:

**Definition 9.** *Let $o_i$ be an option with endpoints $\langle \mathcal{I}_i, \beta_i \rangle$ and let $\mathcal{G}_i = [p_1, \ldots, p_n]$ be the option guide. Given a configuration $x \in \mathcal{X}$, let $n(x) = p_i$ define the closest point on the option guide. Let $d$ be the Euclidean distance measure. The* ***pseudo reward function*** *$R_i(x)$ is defined as:*

$$R_i(x) = \begin{cases} r_t & \text{if } x \in \beta_i \\ r_p & \text{if } x \in \mathcal{S}/\{\alpha(\mathcal{I}_i), \alpha(\beta_i)\} \\ -(d(x, n(x)) & \\ \quad +d(n(x), p_n)) & \text{otherwise} \end{cases}$$

Here $r_t$ is a large positive reward and $r_p$ is a large negative reward that can be tuned as hyperparameters. Intuitively, this automatically generates a dense pseudo-reward function based on the information available from the environment. Thus rather than providing only a sparse reward function (e.g. "$+1$ when the robot reaches the termination set of the option"), Alg. 1 automatically constructs a pseudo reward function that captures this condition (first case), a penalty for straying away from the source and target abstract states (second case), and a reward for covering more of the option guide (third case).

## D Theoretical Results

**Lemma D.1.** *Let $\mathcal{X}$ be the configuration space of the robot $R$ and let $\Phi$ and $\Psi$ be the set of critical regions and RBVD respectively inducing the set of abstract states $\mathcal{S}$ and the neighborhood function $\mathcal{V}$. If there exists a pair of neighboring abstract states $s_i, s_j \in \mathcal{S}$ such that $\mathcal{V}(s_i, s_j) = 1$ then there would exist a pair of option endpoints $\mathcal{I}_{ij}$ and $\beta_{ij}$ such that $\mathcal{I}_{ij} \subset s_i$ and $\beta_{ij} \subset s_j$.*

*Proof.* (Sketch) The proof is straightforward and directly follows from the Alg. 1 itself. Our approach for create options considers all pairs of neighboring abstract states and creates options that transition between them. For more details, refer to Sec. 3.2. □

**Proposition D.1.** *Let $R$ be a holonomic robot using a $\delta_c$-complient controller. For an option $o$ with a pair of endpoints $\langle \mathcal{I}_o, \beta_o \rangle$, if there exists an option guide between $\mathcal{I}_o$ and $\beta_o$ in the form of a $\delta$-clear motion plan such that $\delta_c < \delta$ then there exists a proper partial policy for the option $o$.*

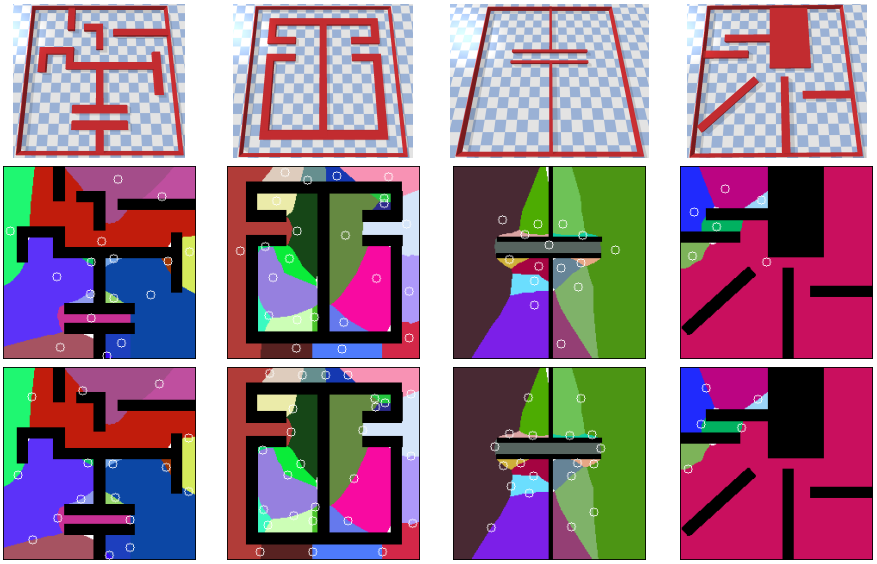
*Proof.* (Sketch) Let $\mathcal{G}_o = \langle p_1, \ldots, p_n \rangle$ be an option guide for the option $o$ as defined in Sec. 3.2.2. Here, each $p_i \in \mathcal{G}_o$ refers to a collision-free configuration $x_i \in \mathcal{X}_{\text{free}}$ that has a collision-free $\delta$-neighborhood represented with $B_\delta(p_i)$. Now, given that the robot uses a $\delta_c$-complient controller such that $\delta_c < \delta$, an optimal partial proper policy can be defined using a function that gives the next closest point on the option guide moving towards the termination set of the option $o$. Let $N_o : x \mapsto p_i$ such that $\forall j > i,\ d(p_j, x) > d(p_i, x)$ and $d(p_i, \beta_o) < d(x, \beta_o)$. An optimal policy can be such that $\pi_o(x) = N_o(x)$ given a $\delta$-clear $\mathcal{G}_i$. Given that the robot is using $\delta_c$-complient controller with the support radius $\delta_c < \delta$, the robot would always end up in $B_{\delta_c}$ neighborhood of a point in the option guide which a subset of $B_\delta$ collision-free neighborhood. This ensures existence of a proper policy for the option $o$. □

**Lemma D.2.** *Let $R$ be a holonomic robot using a $\delta_c$-complient controller. If there exists a pair of option endpoints $\mathcal{I}_i$ and $\beta_i$ with an option guide $\mathcal{G}_i$ in the form of a $\delta$-clear motion plan between $\mathcal{I}_i$ and $\beta_i$ such that $\delta_c < \delta$, and if the low-level policy learner is optimal, then Alg. 1 will learn an option $o_i = \langle \mathcal{I}_i, \beta_i, \mathcal{G}_i, \pi_i \rangle$.*

*Proof.* (Sketch) The proof is straightforward. Proposition D.1 proves existence of a proper policy $\pi_i$ for an option with endpoints $\mathcal{I}_i, \beta_i$ and a holonomic robot $R$ using $\delta_c$-complient controller if there exists $\delta$-clear option guide $\mathcal{G}_i$ such that $\delta_c < \delta$. The rest of the proof relies on the optimality of the low-level learning. The option guide $\mathcal{G}_i$ also induces a dense pseudo-reward function $R_i$ (Sec. 3.2.2 and Appendix. C) that provides a smooth reward function that guides the robot to the termination set of the robot. Given that $\pi_i$ is an optimal policy (proposition D.1) and the low-level policy learner is optimal, it should compute $\pi_i$. □

# E Automatically Identified Critical Regions and RBVDs

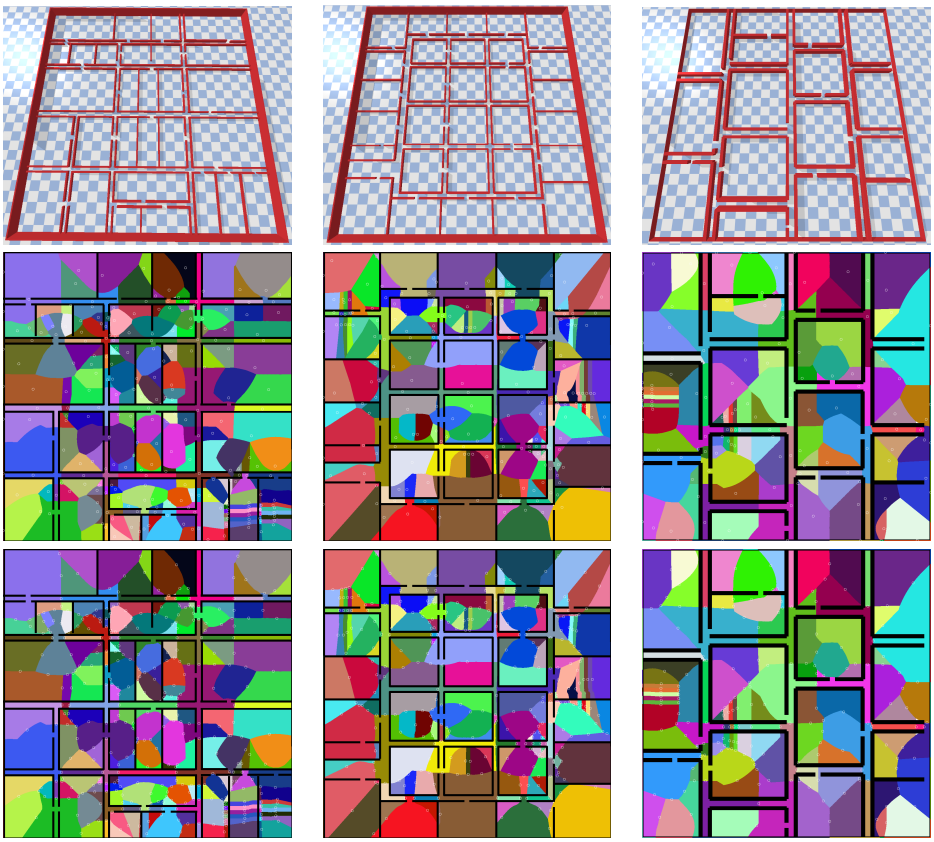
## E.1 Environments S1-S4 $15m \times 15m$



Env S1 Env S2 Env S3 Env S4

Figure 6: Test environments of the size $15m \times 15m$ with the identified abstract states. These images show 2D projections of high-dimensional region-based Voronoi diagrams. Each colored partition represents an abstract state. Top: The white circles represent centroids of the predicted critical regions used to synthesize centroid options. Bottom: The white circles represent the interface regions for each pair of abstract states used to synthesize interface options.ere

**Environments L1-L3** $75m \times 75m$



Env L1 Env L2 Env L3

Figure 7: Test environments of the size $75m \times 75m$ with the identified abstract states. These images show 2D projections of high-dimensional region-based Voronoi diagrams. Each colored partition represents an abstract state. Top: The white circles represent centroids of the predicted critical regions used to synthesize centroid options. Bottom: The white circles represent the interface regions for each pair of abstract states used to synthesize interface options.