Zero Shot Option Invention for Robot Planning Under Uncertainty

Naman Shah
Arizona Sate University
Tempe, AZ, USA
shah.naman@asu.edu

Siddharth Srivastava
Arizona Sate University
Tempe, AZ, USA
siddharts@asu.edu

Abstract:
This paper addresses the problem of inventing and using hierarchical representations for stochastic robot-planning problems. Rather than using hand-coded state or action representations as input, it presents new methods for learning how to create a generalizable high-level action representation for long-horizon, sparse reward robot planning problems in stochastic settings with unknown dynamics. After training, this system yields a robot-class-specific but environment independent planning system that generalizes to different robots, environments, and problem instances. Given new problem instances in unseen stochastic environments, it first creates zero-shot options (without any experience on the new environment) with dense pseudo-rewards and then uses them to solve the input problem in a hierarchical planning and refinement process. Theoretical results identify sufficient conditions for completeness of the presented approach. Extensive empirical analysis shows that even in settings that go beyond these sufficient conditions, this approach convincingly outperforms baselines by $2\times$ in terms of solution time with orders of magnitude improvement in solution quality.

1 Introduction

Recent work on robot planning and learning has led to strong progress on approaches that achieve specific goals in settings with short horizons, dense rewards, and/or deterministic dynamics that are usually encoded in simulators such as MuJoCo. However, this progress has been difficult to translate to pervasive robotics problems that feature long-horizons, sparse rewards, stochastic dynamics, and generalization across different robots, environments, or tasks. In fore-mentioned settings, their performance degrade rapidly. E.g., Fig. 1 shows that recent advances fail to scale for robot motion planning in a large office space with stochastic dynamics as well as fail to learn solution that can be generalized to different problems in the same environment. In fact, their performance drops below that of naïve approaches such as continual re-planning. Such settings constitute a massive departure from the short, dense and/or deterministic settings that has been the focus of much recent work.

This paper addresses the problem of robot planning in the relatively under-studied long horizon, sparse reward and stochastic setting with unknown system dynamics. Solving such problems is challenging: stochasticity implies that deterministic motion planning is not sufficient: the robot can reach unexpected states and thus we need solutions in the form of policies rather than motion plans. Furthermore, the absence of well-defined dynamics models typically requires reinforcement learning (RL) based approaches, but RL algorithms are difficult to scale in long-horizon, sparse-reward settings (as also evidenced in Fig. 1) and generalize to new problems and environments.

We address these technical problems using a novel approach for hierarchical planning and learning that *learns how to invent generalizable abstractions of stochastic robot planning problems* in terms of useful high-level actions represented as options [53] or composable sub-policies. A training phase adapts our general approach to a given robot-class, and yields a robot-class-attuned, environment-independent planning system (Sec. 2). In other words, for a given class of robots with the same number of degrees of freedom, this approach learns how to transform an input continuous problem into a discrete symbolic search problem over the identified options, which is then solved and refined using a hierarchical planning and refinement algorithm (Sec. 3).

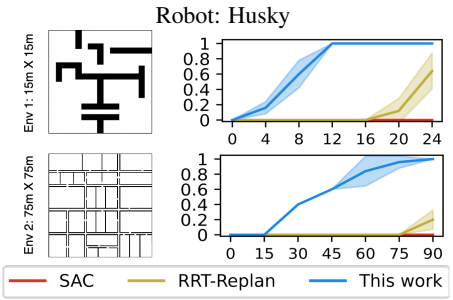


Figure 1: Performance of the two state-of-the-art approaches as a measure of fraction of problems solved (y-axis) in the given time (x-axis). The blue line presents the lower performing variant of the approach developed in this paper. Existing approaches for robot planning show limited scalability in stochastic environments. As the environment size increases, performance falls below that of naïve RRT-Replan.

After training, when given a new robot planning problem in an unseen, stochastic environment, this approach uses the generalized option inventor learned in the first phase for inventing zero-shot options (without any new experience) in the form of desirable pairs of initiation and termination sets, and zero-shot, and dense pseudo-reward that can be used to carry out RL for learning policies for the invented options. Short horizons and dense pseudo-rewards computed for zero-shot options make option-policy learning significantly more sample-efficient than end-to-end policy learning.

Our main contributions are (a) algorithms for learning how to zero-short invent generalizable high-level actions for different robots for stochastic planning problems in test environments not encountered during training; (b) a unified hierarchical learning, planning, and refinement approach for learning how to solve new long-horizon, stochastic problems in sparse reward settings using the learned approach for abstracting them into a space with discrete states and actions; and (c) zero-shot invention of dense pseudo-reward functions for the invented options.

To our knowledge, this paper presents the first approach for *learning how to zero-shot invent* a generalizable hierarchical representation for unseen stochastic robot planning problems. Unlike prior work on the topic, our approach does not require input state or action abstractions, and requires only a kinematic robot specification and a problem generator. This results in robust transferability of learning. If multiple test problems come from the same environment, options can be re-used across instances. Furthermore, this is also the first approach that generalizes the abstraction hierarchy invention process across different robots. Robots used during testing or deployment need not be the same as the robots used while training the option invention pipeline.

Theoretical results characterize the formal properties of this approach such as sufficient conditions for completeness (Sec. 3.1), and show that this approach ensures downward-refinability of the invented options for a class of robots. Empirical analysis (Sec. 4) shows that even in situations that go beyond our sufficient conditions, this approach provides $\sim 2\times$ improvement over existing approaches in terms of computational efficiency and order-of-magnitude improvements in solution quality.

2 Zero-Shot Option Inventors

Algorithm 1: OptionInventor

Input: robot R, training environments E_{train}, test environment E_{test}
Output: set of option $\mathcal{O}$, cost function C

1 $\Theta \leftarrow$ get_critical_region_predicter(R);
2 **if** Θ *is not trained* **then**
3 train Θ using E_{train}
4 $\Phi \leftarrow$ predict_critical_regions(E_{test},Θ);
5 $\Psi, \mathcal{S}, \mathcal{V} \leftarrow$ construct_RBVD(E_{test},Φ);
6 $\mathcal{O}, C \leftarrow$ construct_options(Ψ,$\mathcal{S}$,$\mathcal{V}$);
7 **foreach** $o \in \mathcal{O}$ **do**
8 $\text{mp}_o \leftarrow$ compute_motion_plan(o);
9 $\mathcal{G}_o \leftarrow$ compute_option_guide(o,mp_o);
10 return $\mathcal{O}, C$

Our overall approach for solving long-horizon, stochastic robot planning problems (Appendix A) is to zero-shot invent a set of options for the given problem (Alg. 1), and then to carry out hierarchical planning using these options (Alg. 2). In this section we outline our approach for automatically identifying options (OptionInventor, Alg. 1) for a given environment.

Given a stochastic motion planning problem, Alg. 1 creates a zero-shot state abstraction (lines 1-5) using the methods presented in Appendix A.1. Once abstract states are constructed, we define abstract actions as options (line 6) with their initiation set in one abstract state and the termination set in a different abstract state (discussed in Sec. 2.1). These options (action abstractions) are independent of problem instances, i.e., they are constructed once per environment and robot and reused for different problems (pairs of initial and goal configurations). However, we still need to learn policies for executing such options. As defined, option termination sets turn out to be insufficient for efficiency: they result in a sparse-reward setting, which makes it difficult to

scale RL algorithms for policy learning. To address this limitation, lines 7-9 also create in zero-shot fashion (without collecting additional experience from the environment), an *option guide*: a dense pseudo-reward function for the invented options (discussed in Sec. 2.2).

2.1 Zero-Shot Option Endpoints

Given a set of zero-shot abstract states $\mathcal{S}$ created using the predicted critical regions for a new environment (Def. 7), a neighborhood function $\mathcal{V}$, and an abstraction function α, we define two types of options: (1) *centroid options* that take the robot from the centroid of one critical region to another, and (2) *interface options* that take the robot across an abstract state, i.e., from the boundary between s_i and s_j to the boundary between s_i and s_k. Both forms of options can be composed to solve long-horizon problems (this process is discussed in the next section).

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

Definition 1. *Let $s_i \in \mathcal{S}$ be an abstract state in the RBVD Ψ 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 configurations: $\{x | x \in s_i \wedge d(x, c_i) < t\}$.*

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

Definition 2. *Let $s_i, s_j \in \mathcal{S}$ be neighboring abstract states such that $\mathcal{V}(s_i, s_j) = 1$ in an RBVD Ψ constructed using the set of critical regions Φ. 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 $\beta_{ij} = c_j$.*

Interface options serve as a dual to centroid options. Rather than defining high-level actions that move from the "center" of one abstract state to the "center" of another, they define high-level actions for going across an abstract state, from one boundary to another. To formally define interface options, we first need to define "interface" regions between a pair of neighboring abstract states:

Definition 3. *Let $s_i, s_j \in \mathcal{S}$ be a pair of neighboring states such that $\mathcal{V}(s_i, s_j) = 1$ and ϕ_i and ϕ_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 4. *Let $s_i, s_j, s_k \in \mathcal{S}$ be abstract states 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 $\beta_{o_{ijk}} = \hat{\phi}_{jk}$.*

We can now utilize these definitions to define, in zero-shot fashion, the complete set of centroid options and interface options for a new environment. Recall that the RBVD Ψ induces a neighborhood function $\mathcal{V} : \mathcal{S} \times \mathcal{S} \rightarrow \{0, 1\}$. The set of centroid options is defined as $\mathcal{O}_c = \{o_{ij} | \forall s_i, s_j \in \mathcal{S}, \mathcal{V}(s_i, s_j) = 1 \wedge \mathcal{I}_{ij} = c_i \wedge \beta_{ij} = c_j\}$, where c_i represents the centroid of the critical region r_i for the abstract state s_i.

Similarly, the set of interface options is defined 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 $\hat{\phi}_{ij}$ represents an interface region for a pair of neighboring abstract states s_i and s_j.

2.2 Zero-Shot Option Guides

Given an option defined using the methods discussed above, we define an option guide as a dense pseudo-reward function. We will use the option guide to improve sample efficiency while learning policy for an option in sparse reward settings.

Intuitively, option guides are defined using conceptual envelopes around *deterministic* motion plans that can be computed relatively easily using existing methods. Formally, we define an *ϵ-clear motion plan* as a motion plan in which every configuration has an ϵ-neighborhood that is collision free. With

a slight abuse of notation we use the abstraction function with a set of low-level configurations rather than a single configuration such that for a set A, $\alpha(A) = \{\alpha(x) | \forall x \in A\}$.

Let o_i be an option with endpoints $\langle \mathcal{I}_i, \beta_i \rangle$, and centroids $c_{\mathcal{I}_i}$ and c_{β_i} for $\mathcal{I}_i$ and β_i respectively. Given a threshold distance t, an arbitrary neighborhood radius ϵ, and the Euclidean distance measure d, we can define an ϵ-clear motion plan $\mathcal{G}_i$ for the option o_i as follows: $\mathcal{G}_i = \langle p_1, \ldots, p_n \rangle$ 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 $p_i \in \alpha(\beta_i)$. In practice, we found that any sampling-based motion planner with ϵ-inflated obstacles can be used to construct such motion plans.

We define the option guide for o_i as a dense pseudo-reward defined using $\mathcal{G}_i$ as follows. Intuitively, the option guide is a dense pseudo-reward function that provides the robot with 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. Formally, this is defined as follows:

Definition 5. *Let o_i be an option with endpoints $\langle \mathcal{I}_i, \beta_i \rangle$ and let $\mathcal{G}_i = \langle p_1, \ldots, p_m \rangle$ be an ϵ-clear motion plan for a given ϵ. Given a configuration $x \in \mathcal{X}$, let $n(x) = p_i$ define the closest point on $\mathcal{G}_i$. The **option guide** $R_i(x)$ is defined as:*

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

3 Hierarchical Stochastic Motion Planning Using Zero-Shot Options

Algorithm 2: Stochastic Hierarchical Abstraction-guided Robot Planner (SHARP)

Input: Training environments E_{train}, test environment E_{test}, initial and goal configurations x_i and x_g
Output: A policy Π composed of options

1 **if** *abstraction is **not** constructed* **then**
2 $\quad \mathcal{O}, C \leftarrow$ OptionInventor(R,E_{train},E_{test});
3 $s_i, s_g \leftarrow$ get_abstract_states(x_i,x_g);
4 **while** *not refined* **do**
5 $\quad p \leftarrow$ get_new_high_level_plan(s_i,s_g,$\mathcal{O}$,C);
6 $\quad$ **if** $p = \emptyset$ **then**
7 $\quad\quad$ break;
8 $\quad \Pi$ = empty_list;
9 $\quad \pi_0 \leftarrow$ lear_ll_policy(x_i,$\mathcal{I}_{o_1}$);
10 $\quad \Pi$.add(π_0);
11 $\quad$ **foreach** $o \in p$ **do**
12 $\quad\quad$ **if** π_o *does **not** exist* **then**
13 $\quad\quad\quad$ **if** $\mathcal{G}_o = \emptyset$ **then**
14 $\quad\quad\quad\quad$ flag o infeasible;
15 $\quad\quad\quad\quad$ break;
16 $\quad\quad\quad \pi_o \leftarrow$ learn_ll_policy($\mathcal{I}_o$,β_o,$\mathcal{G}_o$);
17 $\quad\quad\quad$ adjust the option cost C_o;
18 $\quad\quad \Pi$.add(π_o);
19 $\quad$ refined $\leftarrow$ True;
20 **if** *refined* **then**
21 $\quad \pi_{n+1} \leftarrow$ learn_ll_policy(β_{o_n},x_g);
22 $\quad \Pi$.add(π_{n+1});
23 $\quad$ **return** Π;
24 **return** failure;

The SHARP algorithm (Alg. 2) presents our overall approach for using the zero-shot options defined above for hierarchical motion planning under uncertainty. It takes as input an SMP problem $P = \langle \mathcal{X}, \mathcal{U}, x_i, x_g \rangle$, a simulator, and an occupancy matrix of the environment, and produces a partial policy $\Pi : \mathcal{X} \rightarrow \mathcal{U}$ that maps each reachable robot configuration to a control action. The algorithm starts by invoking the OptionInventor in line 2 to construct zero-shot state and action abstractions (in the form of options) if they have not been constructed for the given robot R and the environment E_{test} pair (Sec. 2).

Lines 4-19 use these options as high-level actions for computing high-level plans. Line 5 uses an incremental plan generator that takes the set of invented options along with the abstract initial and goal states as input and generates a high-level plan using A* search. This module considers the initiation and termination sets of the invented options as preconditions and effects. 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.

Once a plan in the form of a sequence of options is obtained in line 5, SHARP starts refining each option in the plan by computing option policies. However, before computing the policy for the first option in the plan, 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 9). If a policy exists for the option from the previous invocation of the algorithm, then our approach uses the same policy. Before

computing a policy for an option, Alg. 2 checks for its option guide. If an option guide does not exist, the option is marked as infeasible and a new high-level plan is computed from the initial abstract state (line 14). Once an option guide is computed for an option, line 16 uses an off-the-shelf low-level policy learner to compute a policy for it. After computing (or reusing) policies for all the options in the plan, line 21 generates an additional 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.

Finally, we compute a *composed policy* by composing policies for every option in this high-level plan (lines 18 and 22). A **composed policy** Π for a high-level plan is an FSA with one controller state for each option in the plan. For a controller state q_i, $\Pi(x) = \pi_i(x)$ where π_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 $x \in \mathcal{I}_{o_{i+1}}$.

In order to aid transferability, SHARP only synthesizes options once per each environment and robot. It efficiently transfers the learned option policies by updating the option costs (C) using the average number of steps from initiation sets to the termination sets of the options in multiple rollouts of the learned option policies (line 17).

3.1 Theoretical Results

We now present theoretical properties of Alg. 2. Let $B_\delta(x)$ for $\delta > 0$ define the δ-neighborhood of $x \in \mathcal{X}$ under the Euclidean metric. Recall that each controller implicitly defines a transition function with a probability distribution $\mu(x+u)$ for the control action u (see Sec. A). A δ-*compliant controller* 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 μ other than an upper bound on the support radius. Here, we refer to δ as the *support radius* for the given controller.

The following theoretical results charaterize formal properties of the presented approach. We present the results below; proofs are included in Appendix D.

Thm. 3.1 shows that the construction process of the options ensures that the zero-shot options are indeed composable and can be used for high-level deterministic planning.

Theorem 3.1. *For a given stochastic motion planning problem $P = \langle \mathcal{X}, \mathcal{U}, x_1, x_n \rangle$, let Φ be the set of identified critical regions and Ψ 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 Π such that the resulting configuration after the termination of every option in Π would be the goal configuration x_n.*

Thm. 3.2 asserts that when used with an optimal low-level policy learner, SHARP is probabilistically complete for holonomic robots.

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 δ-clear motion plans, and an optimal low-level policy learner, if there exists a δ-clear motion plan for the robot R from x_1 to x_n that forms a sequence of distinct abstract states, then Alg. 2 will find a proper policy for the given stochastic motion planning 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 in the supplementary material.

Our evaluation is organized to address the following key questions: (1) Does the presented approach of zero-shot option invention followed by hierarchical planning and refinement improve performance in terms of computational efficiency and solution quality?; and (2) Can zero-shot options be transferred to new problems in the same environment?

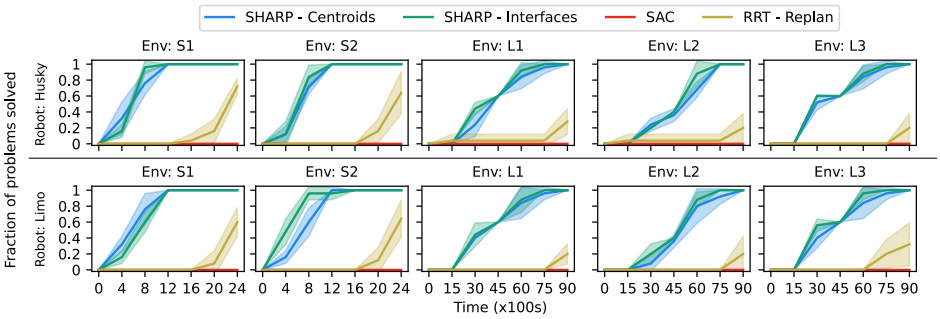


Figure 3: (Higher values are better) Times taken (averaged over 5 trials) by our approach (SHARP) and baselines to compute solutions in the test environments. X-axis shows the time and y-axis shows the fraction of the problems solved in the given time.

Results across an extensive evaluation suite indicate that the presented approach creates and uses zero-shot options effectively. In larger environments (L1-L3), ours is the only approach that shows significant learning, and it achieves a significantly higher solution quality than all baselines. We now present our evaluation framework and results in detail.

Evaluation framework and metrics We organized the overall evaluation of the presented approach as follows. Given a previously unseen environment E_{test} and a problem instance $\langle x_i, x_g \rangle$, SHARP (Alg. 2) zero-shot invents options for E_{test} and uses them to compute a policy for the test problem instance. The total solution time recorded for SHARP includes the time taken to run OptionInventor (which includes predicting critical regions, creating state abstractions, inventing option signatures, and computing option guides), and to execute hierarchical planning and refinement process listed under the SHARP algorithm (Alg. 2).

We evaluated the *computational efficiency* of all considered approaches in terms of the number of problems solved in a given amount of time. For learning-based approaches, a problem is considered to be solved in these experiments when the current learned policy yields an average reward of +500 over 10 rollouts. For RRT-replan, a problem is considered to be solved when the robot reaches the 0.2m neighborhood of the goal configuration. All approaches were assigned a uniform timeout per problem of 2400 or 9000 seconds.

In addition, we use two metrics to evaluate solution quality since it is often easy to compute meaningless policies in a short time frame: The *average solution cost* is defined as the average number of steps taken while executing a computed solution; *solution reliability* is defined as the likelihood of solving the given problem by executing the computed solution. Both metrics are computed over 20 independent trials of the computed solution on the input problem instance.

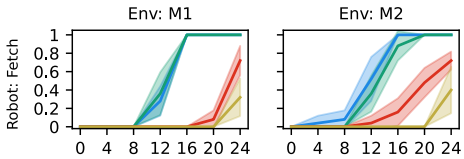


Figure 2: (Contd. from Fig. 3 with same setup) Results for manipulation problems with the Fetch.

Figs. 3, 2, and 4 summarize the results of our evaluation in terms of these metrics across a wide range of robots, environments and test problems. We discuss the details of this evaluation including notes on our implementation, environment and baseline selection, and our main observations below.

Our implementation We implemented two variants of our approach: SHARP-centroids and SHARP-interfaces, which invent and use centroid options and interface options, respectively. Both implementations use PyBullet and PyTorch [42]. PyBullet does not feature stochasticity robot movements. We introduced stochasticity by adding random perturbations (unknown to Alg. 2) in control targets of actions during training and execution. We used default robot controllers to evaluate the learned policies. We used HARP [47] with $\epsilon = 0$ for computing zero-shot option guides.

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 [16] as a low-level policy learner in lines 9, 16, and 21 of Alg. 2.

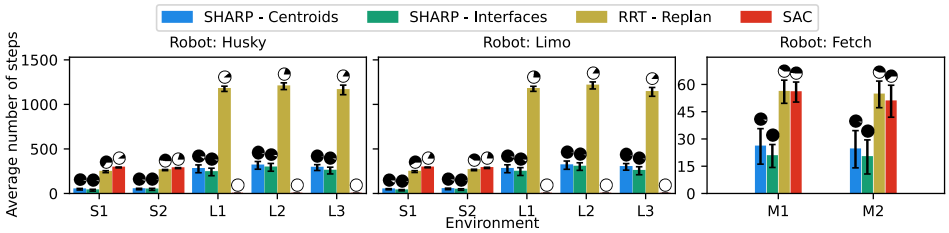


Figure 4: (Lower values or darker circles are better) Average number of steps taken in the **successful** execution of the learned policies and success rates for our approach and the baselines. The pie chart over each bar represents the success rate (shaded black area) while executing the learned policy.

Test environments and robots We evaluated our approach across 7 test environments (Fig. 8) (not included in training the critical region predictor), 3 non-holonomic robots (Fig. 8) and a total of 60 navigation and manipulation problems. Dimensions of the environments are as follows: S1, S2: $15m \times 15m$; L1, L2, L3: $75m \times 75m$. Problem specific timeouts were set at 2400s for small environments and manipulation problems and 9000s for larger environments. For each environment, we generated 5 problem instances by randomly sampling different initial and goal configuration pairs.

We used the following robots: the ClearPath Husky (3-DOF), the AgileX Limo (3-DOF), and the Fetch manipulator robot (7-DOF). Details of these robots are presented in Appendix F.

Baseline selection We considered and evaluated several learning and planning approaches [34, 16, 31, 37, 36, 5] as potential baselines for this work. Of these, only RRT-Replan [34] and SAC [16] solved any problem instances within the timeouts discussed above. Therefore, we compared our approach against SAC and RRT-Replan. SAC is an off-policy deep reinforcement learning approach that 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 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 used the same input robot models, simulators, and low-level controllers as our approach.

4.1 Analysis of Results

Computational Efficiency Figs. 3 and 2 show the fraction of problem instances solved in a given amount of time by both variants of SHARP and the baselines. In our case, this includes the time taken to create the abstract states and actions as well as to compute the solutions. Each subsequent problem uses learned high-level actions (policies and options) from the previous problem instances when available. Results show SHARP shows significantly greater scalability and computational efficiency. In most cases, baselines take $2\times$ the time taken by SHARP to compute a solution. These differences increase for larger environments, where baselines were able to solve less than 50% of the environments that SHARP solved within the same timeouts.

These results illustrate the impact of learning to zero-shot invent and utilize options: 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. Manipulation environments show a relatively smaller difference between performance of all the approaches owing to smaller horizons. This reinforces the key contribution of our approach of creating problems with smaller horizons using options in order to solve problems with significantly large horizons.

Solution quality Fig. 4 shows solution cost and solution reliability (as defined above) for solutions computed by all considered approaches. These results show that SHARP's planning over zero-shot options results in lower cost solutions: they require significantly fewer steps during execution compared to baselines, with the differences frequently spanning *orders of magnitude*. We acknowledge that RRT-Replan is not an optimal planning approach. However, the solution quality also represents the amount of time RRT-Replan had to re-compute and re-execute the solution.

Computing policies that account for stochasticity makes SHARP's solution reliability uniformly above 90%, nearly $3\times$ that of RRT-Replan (the best performing baseline) on the larger test environments. RRT-Replan's solutions had an execution success rate of $\sim 50\%$ in the smaller navigation (S1, S2) and

manipulation (M1, M2) environments, and a success rate of less than 33% in the larger environments (L1-L3). SAC's solution reliability was lower, indicating limited scalability of end-to-end learning in long-horizon problems.

Zero-shot option invention and reuse Appendix G shows the predicted critical regions, 2D projections of the RBVDs, and synthesized option endpoints for our test environments. These results show that our approach is able to zero-short invent options for new, unseen test environments. When new problem instances come from a common environment, our approach is able to transfer these zero-shot options and their policies to new problem instances (Appendix E). Centroid options showed greater reuse rates on average across all environments (52%) than interface options (45%).

Generalization across robots Our approach generalizes the option inventor to different robots. Thus, it supports different robots for training the option inventors and deployment or evaluation. We test this generalization by using a simple non-holonomic robot with 3-DOF and deterministic dynamics to generate the training data in 20 training environments of the size $5m \times 5m$ for training the option inventor for navigation environments. These learned inventors were evaluated using two different non-holonomic robots in much larger environments.

5 Related Work

We discuss the relationship of our work with the most closely related approaches here and present a broader discussion in Appendix B. To our knowledge, this is the first approach for zero-shot option invention and hierarchical planning and refinement for stochastic robot planning problems that does not require hand-coded abstractions or options as input. In addition, it can be applied to problems and environments not seen during training.

Approaches for stochastic motion planning [11, 32, 57, 21, 7, 19] utilize analytical dynamical models of the robot while this paper addresses problems where such models may not be available. Another direction of research aims to learn task-specific subgoals in the given test environment [31, 4, 40, 41, 10]. These approaches utilize interactions with the test environments to learn useful subgoals which can then be utilized for learning options and other forms of high-level actions. A related direction of research focuses on learning task-specific options while interacting in the target environment [51, 50, 9, 13, 5, 6]. In contrast, this paper focuses on zero-shot options that are created without interacting with the test environments or tasks to improve efficiency and scalability.

Finally, there has been a lot of progress on short-horizon (~ 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 DRL approaches for visual model predictive control (MPC) [58, 35, 14, 15, 18, 55, 33, 12, 2, 17]. While this paper's focus is on long-horizon sparse-reward planning problems with unknown stochastic dynamics, (visual) MPC techniques can be used for learning low-level policies in conjunction with our approach (Alg. 2, line 22).

6 Conclusion

This paper presents the first approach that uses a data-driven process to learn to create state and action abstractions for unseen environments and problem instances. 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 problems that are currently beyond the scope of known methods.

Acknowledgements

We thank Kiran Prasad for helping to implement the initial version of the approach. The work is funded by NSF under the 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] 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.

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

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

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

[24] 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.

[25] 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.

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

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

[28] 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.

[29] 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.

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

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

[32] 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.

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

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

[35] 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.

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

[37] 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.

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

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

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

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

[42] 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.

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

[44] 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.

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

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

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

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

[49] 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.

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

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

[52] 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.

[53] 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.

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

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

[56] 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.

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

[58] 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.

[59] 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 Background

Let $\mathcal{X} \subseteq \mathbb{R}^d = \mathcal{X}_{\text{free}} \cup \mathcal{X}_{\text{obs}}$ be the configuration space (C-space) of a robot R and let O be a set of obstacles in a given environment. Given a collision function $f : \mathcal{X} \rightarrow \{0, 1\}$, $\mathcal{X}_{\text{free}}$ 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 6. *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 C-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 τ. 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 stochastic motion planning (SMP) problems in a manner similar to stochastic shortest path problems (SSPs) [8]. Formally, a *stochastic motion planning problem* P is defined 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 $\pi : \mathcal{X} \rightarrow \mathcal{U}$ that maps each reachable configuration in the configuration space (when starting with x_0 and following π) to a control action from the set of controls (actions) $\mathcal{U}$.

A.1 State Abstractions

In this work we use the notion of state abstractions as a finite partitioning of a continuous state space. Formally, a state abstraction from a concrete state space $\mathcal{X}$ to a finite, discrete state space S is a function that maps each $x \in \mathcal{X}$ to an element of S. State abstractions created by domain experts have been used extensively to speed up planning and learning. Recent work also presents approaches for learning state abstractions (e.g., [29, 47]). In this work we utilize the critical-region based state abstraction technique developed by Shah and Srivastava [47] since it allows zero-shot state abstractions for new problems. We present here a brief summary of this approach to highlight the properties and input requirements that are utilized in the current paper.

In this approach, critical regions of a configuration space characterize regions that tend to have a high solution density and thus are essential for solving problem instances using sampling based motion planners. This concept generalizes and unifies notions of hubs (e.g., the center of a room from which multiple locations are accessible) and bottlenecks (e.g., a doorway that forces the robot to follow a narrow path). Given the kinematic model of a robot, it is possible to train a deep neural network to predict critical regions for new environments. A critical region prediction can, in turn be used to define an abstraction of the configuration space:

Definition 7. *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 critical regions Φ and a robot R, a **region-based Voronoi diagram (RBVD)** Ψ 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 ψ_i is connected.*

In this framework, abstract states are defined using a bijective function $\ell : \Psi \rightarrow \mathcal{S}$ that maps each Voronoi cell $\psi \in \Psi$ to an abstract state $s \in \mathcal{S}$. The RBVD Ψ induces an abstraction function $\alpha : \mathcal{X} \rightarrow \mathcal{S}$ such that $\alpha(x) = s$ iff there exists a cell ψ 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 $\alpha(x) = s$. A neighborhood function $\mathcal{V} : \mathcal{S} \times \mathcal{S} \rightarrow \{0, 1\}$ 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. We say a pair of abstract states s_1 and s_2 are neighbors iff there exists a pair of configuration $x_1 \in s_1$ and $x_2 \in s_2$ and there exists a motion plan between x_1 and x_2.

B 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 [25, 34, 30, 44, 46] have been developed for motion planning in deterministic environments. Kavraki et al. [25], LaValle [34], Kuffner and LaValle [30] 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. [20], Pivtoraiko et al. [44], Saxena et al. [46] 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, 32, 57, 21, 7, 19] 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) [25] where the weights capture the probability of the robot making the corresponding transition. Huynh et al. [21] extend SMR for computing stochastic policies through value iteration over motion trees constructed using RRT [34]. Sun et al. [52] 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. [54] 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, 52, 54] require discretized actions. Du et al. [11], Van Den Berg et al. [56] formulate a stochastic motion planning problem as a POMDP to capture uncertainty in robot sensing and movements. Multiple approaches [23, 13, 24] 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. [31], Bacon et al. [4], Nachum et al. [40, 41], 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. [43] combine imitation learning with reinforcement learning for identifying subgoals from expert trajectories and bootstrap reinforcement learning. Levy et al. [36] 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. [26] 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 [51, 50, 9, 33, 13, 5, 6] perform hierarchical learning by identifying task-specific options through experience collected in the test environment and then use these options [53] along with low-level primitive actions. Stolle and Precup [51], Şimşek et al. [50] 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 [53]. 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 [58, 35, 14, 15, 18, 55, 12, 2, 17] 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 (~ 5 seconds).

Planning with options Approaches for combining symbolic planning with reinforcement learning [48, 59, 22, 38, 28, 29, 49] 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.

C Training The Critical Region Predictor

Alg. 2 first needs to identify critical regions [39] 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. 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_{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 [47]. 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 [47], however, we used the critical regions predictor developed by them for a rectangular holonomic robot.

Shah and Srivastava [47] use 20 training environments (E_{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 [47] 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 [45] 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 [27] with a learning rate of 10^{-4} and train the network for $50,000$ epochs.

D Theoretical Results

Lemma D.1. *Let $\mathcal{X}$ be the configuration space of the robot R and let Φ and Ψ 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 β_{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. 2 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. 2. □

Proposition D.1. *Let R be a holonomic robot using a δ_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 β_o in the form of a δ-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. 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 δ-neighborhood represented with $B_\delta(p_i)$. Now, given that the robot uses a δ_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 δ-clear $\mathcal{G}_i$. Given that the robot is using δ_c-complient controller with the support radius $\delta_c < \delta$, the robot would always end up in B_{δ_c} neighborhood of a point in the option

guide which a subset of B_δ 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 δ_c-complient controller. If there exists a pair of option endpoints $\mathcal{I}_i$ and β_i with an option guide $\mathcal{G}_i$ in the form of a δ-clear motion plan between $\mathcal{I}_i$ and β_i such that $\delta_c < \delta$, and if the low-level policy learner is optimal, then Alg. 2 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 π_i for an option with endpoints $\mathcal{I}_i, \beta_i$ and a holonomic robot R using δ_c-complient controller if there exists δ-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. 2.2) that provides a smooth reward function that guides the robot to the termination set of the robot. Given that π_i is an optimal policy (proposition D.1) and the low-level policy learner is optimal, it should compute π_i. □

Theorem D.1. *For a given stochastic motion planning problem $P = \langle \mathcal{X}, \mathcal{U}, x_1, x_n \rangle$, let Φ be the set of identified critical regions and Ψ 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 Π such that the resulting configuration after the termination of every option in Π 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. 2 and 4 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 Π such that for every pair of consecutive options $o_i, o_j \in \Pi$, $\mathcal{I}_{o_j} = \beta_{o_i}$. Thus, we can say that if every option in Π terminates, then the resulting configuration would be the goal configuration. □

Theorem D.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 δ-clear motion plans, and an optimal low-level policy learner, if there exists a δ-clear motion plan for the robot R from x_1 to x_n that forms a sequence of distinct abstract states, then Alg. 2 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 δ-clear motion plan from the initial configuration x_i to goal configuration x_g. This δ-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. 2 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 a δ-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 a δ-clear motion plan and an optimal low-level policy learner, Alg. 2 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 D.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. □

E Option Reuse Rates

F Evaluation Robots

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.

	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.

G Automatically Identified Critical Regions and RBVDs

G.1 Environments S1-S4 $15m \times 15m$

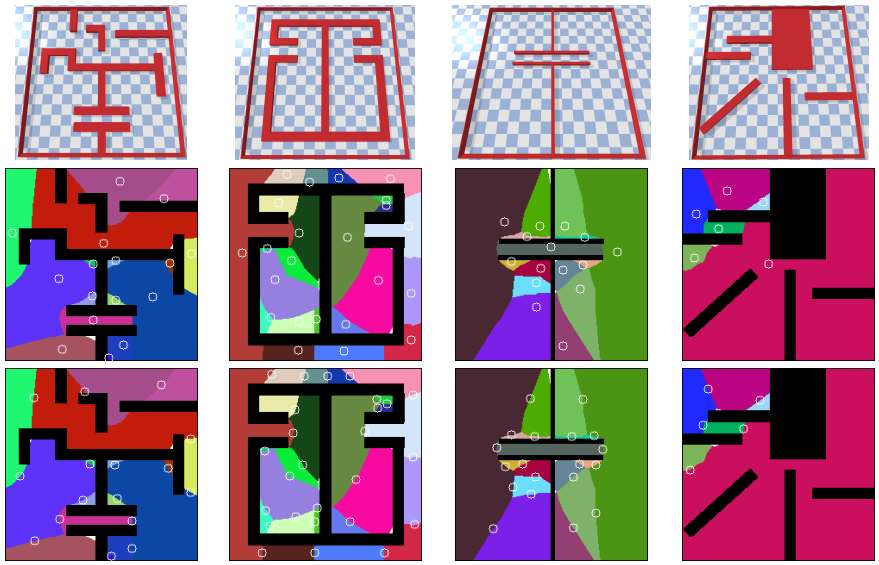
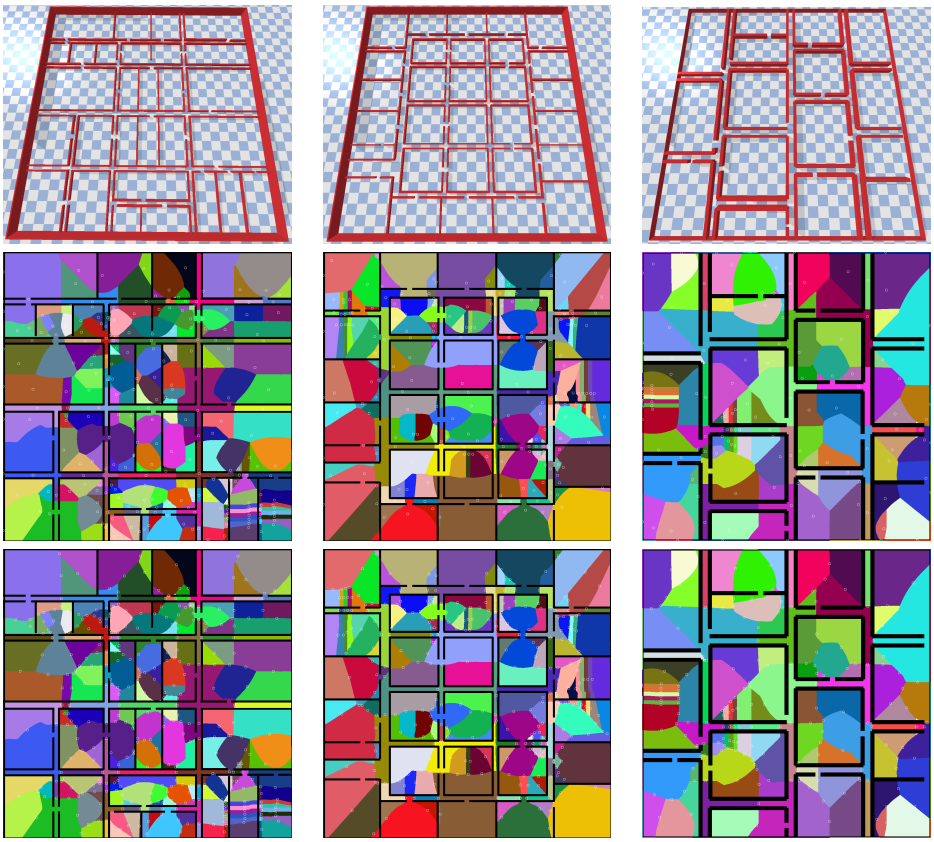


Figure 6: Test environments of the size $15m \times 15m$ with the identified abstract states. These images show 2D projections of high-dimensional region-based Voronoi diagrams. Each colored partition represents an abstract state. Top: The white circles represent centroids of the predicted critical regions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

H Evaluation Configuration and Hyper Parameters

All experiments were conducted on an Ubuntu 18.04 system with 3.6 GHz 24 thread AMD Threadripper PRO 5965WX with 64 GB memory. All algorithms were implemented using Python. We did not use GPU to train our DRL approaches given RRT-Replan can not use GPUs.

The hyperparameters other than the 2400 and 9000 seconds of timeout for each approach are as follows.

SAC Parameters:

Parameter	Value
network architecture	[256,256]
action noise	0.1
learning_starts	1000 steps
enf_coef	auto
optimize_memory	True
learning rate	0.003
use_sde	True
batch size	256
buffer size	1000000

RRT Parameters

Parameter	Value
step size	0.05 / 0.5
goal tolerance	0.2

I Test Environments and Robots

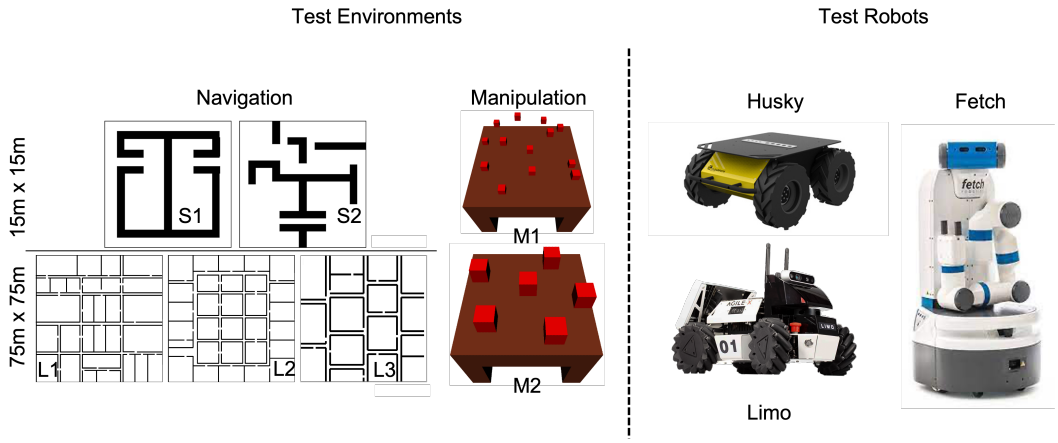


Figure 8: Our test environments and robots.