Learning Functionally Decomposed Hierarchies for Continuous Control Tasks with Path Planning

Sammy Christen* **Lukas Jendele*** **Emre Aksan** **Otmar Hilliges**
Department of Computer Science
ETH Zurich, Zurich 8092, Switzerland
sammy.christen@inf.ethz.ch

Abstract

We present HiDe, a novel hierarchical reinforcement learning architecture that successfully solves long horizon control tasks and generalizes to unseen test scenarios. Functional decomposition between planning and low-level control is achieved by explicitly separating the state-action spaces across the hierarchy, which allows the integration of task-relevant knowledge per layer. We propose an RL-based planner to efficiently leverage the information in the planning layer of the hierarchy, while the control layer learns a goal-conditioned control policy. The hierarchy is trained jointly but allows for the composition of different policies such as transferring layers across multiple agents. We experimentally show that our method generalizes across unseen test environments and can scale to tasks well beyond 3x horizon length compared to both learning and non-learning based approaches. We evaluate on complex continuous control tasks with sparse rewards, including navigation and robot manipulation. See videos at https://sites.google.com/view/hide-rl.

1 Introduction

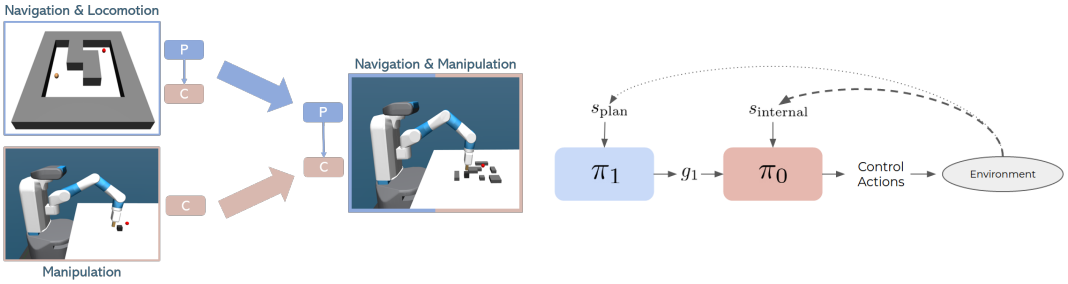
Reinforcement learning (RL) can solve long horizon control tasks with continuous state-action spaces in robotics [1–3], such as locomotion [4], manipulation [5], or human-robot interaction [6]. However, tasks that involve extended planning and sparse rewards still pose many challenges in successfully reasoning over long horizons and in achieving generalization from training to different test environments. Therefore, hierarchical reinforcement learning (HRL) splits the decision making problem into several subtasks at different levels of abstraction [7, 8], often learned separately via curriculum learning [9–12], or end-to-end via off-policy and goal-conditioning [13–15]. However, these methods share the full-state space across layers, even if low-level control states are not strictly required for planning. This limits i) modularity in the sense of transferring higher level policies across different control agents and ii) the ability to generalize to unseen test tasks without retraining.

In this paper, we study how a more explicit hierarchical task decomposition into local control and global planning tasks can alleviate both issues. In particular, we hypothesize that explicit decoupling of the state-action spaces of different layers, whilst providing suitable task-relevant knowledge and efficient means to leverage it, leads to a task decomposition that is beneficial for generalization across agents and environments. Thus, we propose a 2-level hierarchy (see Figure 1b) that is suited for continuous control tasks with a 2D-planning component. Furthermore, we show in our experiments that planning in 3D is also possible. Global environment information is only available to the planning layer, whereas the full internal state of the agent is only accessible by the control layer. To leverage the global information, we propose the integration of an efficient, RL-based planner.

The benefit of this explicit task decomposition is manyfold. First, the individual layers have access only to task-relevant information, enabling layers to focus on their individual tasks [16]. Second, the

*Equal contribution

Presented at Deep RL Workshop, NeurIPS 2020.



(a) Example of agent transfer

(b) Our proposed hierarchical architecture

Figure 1: a) Showing the decompositionality of our approach, the **p**lanning policy of simple agent is combined with more complex **c**ontrol policies. b) Our 2-layer HRL architecture. The planning layer π_1 receives information crucial for planning s_{plan} and provides subgoals g_1 to the lower level. A goal-conditioned control policy π_0 learns to reach the target g_1 given the agent's internal state s_{internal}.

modularity allows for the composition of new agents without retraining. We demonstrate this via transferring the planning layer across different low-level agents ranging from a simple 2DoF ball to a 17DoF humanoid. The approach even allows to generalize across domains, combining layers from navigation and robotic manipulation tasks to solve a compound task (see Figure 1a).

In our framework, which we call HiDe, a goal-conditioned control policy π_0 on the lower-level of the hierarchy interacts with the environment. It has access to the proprioceptive state of an agent and learns to achieve subgoals g_1 that are provided by the planning layer policy π_1. The planning layer has access to task-relevant information, e.g., a top-down view image, and needs to find a subgoal-path towards a goal. We stress that the integration of such additional information into HRL approaches is non-trivial. For example, naively adding an image to HRL methods [13, 14] causes an explosion of the state-space complexity and hence leads to failure as we show in Section 5.1. We propose a specialized, efficient planning layer, based on MVProp [17] with an added learned dynamic agent-centric attention window which transforms the task-relevant prior into a value map. The action of π_1 is the position that maximizes the masked value map and is fed as a subgoal to the control policy π_0. While the policies are functionally decoupled, they are trained jointly, which we show to be beneficial over separately training a control agent and attaching a conventional planner.

We focus on continuous control problems that involve navigation and path planning from top-down view, e.g., an agent navigating a warehouse or a robotic arm pushing a block. However, we show as a proof of concept that HiDe can also work in non-euclidean space and be extended to planning in 3D. In our experiments, we first demonstrate that generalization and scaling remain challenging for state-of-the-art HRL approaches and are outperformed by our method. We also compare against a baseline with a non-learning based planner, where a control policy trained with RL is guided by a conventional RRT planner [18]. We then show that our method can scale beyond 3x longer horizons and generalize to randomly configured layouts. Lastly, we demonstrate transfer across agents and domains. The results indicate that an explicit decomposition of policy layers in combination with task-relevant knowledge and an efficient planner are an effective tool to help generalize to unseen environments and make HRL more practicable. In summary our main contributions include:

- A novel HRL architecture that enforces functional decomposition into global planning and low-level control through a strict separation of the state space per layer, in combination with an RL-based planner on the higher layer of the hierarchy, to solve long horizon control tasks.
- We provide empirical evidence that task-relevant priors are essential components to enable generalization to unseen test environments and to scale to larger environments, which HRL methods struggle with.
- Demonstration of transfer of individual modular layers across different agents and domains.

2 Background

2.1 Goal-Conditioned Reinforcement Learning

We model a Markov Decision Process (MDP) augmented with a set of goals $\mathcal{G}$. We define the MDP as a tuple $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{G}, \mathcal{R}, \gamma, \mathcal{T}, \rho_0, \}$, where $\mathcal{S}$ and $\mathcal{A}$ are set of states and actions, respectively, $\mathcal{R}_t =$

$r(s_t, a_t, g_t)$ a reward function, γ a discount factor $\in [0, 1]$, $\mathcal{T} = p(s_{t+1}|s_t, a_t)$ the transition dynamics of the environment and $\rho_0 = p(s_1)$ the initial state distribution, with $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$. Each episode is initialized with a goal $g \in \mathcal{G}$ and an initial state is sampled from ρ_0. We aim to find a policy $\pi : \mathcal{S} \times \mathcal{G} \to \mathcal{A}$, which maximizes the expected return. We a an actor-critic framework where the goal augmented action-value function is defined as: $Q(s_t, g_t, a_t) = \mathbb{E}_{a_t \sim \pi, s_{t+1} \sim \mathcal{T}} \left[\sum_{i=t}^{T} \gamma^{i-t} \mathcal{R}_t \right]$.

The Q-function (critic) and the policy π (actor) are approximated by using neural networks with parameters θ^Q and θ^π. The objective for θ^Q minimizes the loss:

$$\begin{aligned} L(\theta^Q) &= \mathbb{E}_{\mathcal{M}} \left[\left(Q(s_t, g_t, a_t; \theta^Q) - y_t \right)^2 \right], \text{where} \\ y_t &= r(s_t, g_t, a_t) + \gamma Q(s_{t+1}, g_{t+1}, a_{t+1}; \theta^Q). \end{aligned} \tag{1}$$

The policy parameters θ^π are trained to maximize the Q-value:

$$L(\theta^\pi) = \mathbb{E}_\pi \left[Q(s_t, g_t, a_t; \theta^Q) | s_t, g_t, a_t = \pi(s_t, g_t; \theta^\pi) \right] \tag{2}$$

2.2 Hindsight Techniques

In HAC, Levy et. al [13] apply two hindsight techniques to address the challenges introduced by the non-stationary nature of hierarchical policies and the environments with sparse rewards. In order to train a policy π_i, optimal behavior of the lower-level policy is simulated by *hindsight action transitions*. More specifically, the action a_i of the upper policy is replaced with a state s_{i-1} that is actually achieved by the lower-level policy π_{i-1}. Identically to HER [19], *hindsight goal transitions* replace the subgoal g_{i-1} with an achieved state s_{i-1}, which consequently assigns a reward to the lower-level policy π_{i-1} for achieving the virtual subgoal. Additionally, a third technique called *subgoal testing* is proposed. The incentive of subgoal testing is to help a higher-level policy understand the current capability of a lower-level policy and to learn Q-values for subgoal actions that are out of reach. We find all three techniques effective and apply them to our model during training.

2.3 Value Propagation Networks

Tamar et. al [20] introduce value iteration networks (VIN) for path planning problems. Nardelli et. al [17] propose value propagation networks (MVProp) with better sample efficiency and generalization behavior. MVProp creates reward- and propagation maps covering the environment. A reward map highlights the goal location and a propagation map determines the propagation factor of values through a particular location. The reward map is an image $\bar{r}_{i,j}$ of the same size as the environment image, where $\bar{r}_{i,j} = 0$ if the pixel (i, j) overlaps with the goal position and -1 otherwise. The value map V is calculated by unrolling max-pooling operations in a neighborhood N for k steps as follows:

$$v_{i,j}^{(0)} = \bar{r}_{i,j}, \quad \bar{v}_{i,j}^{(k)} = \max_{(i',j') \in N(i,j)} \left(\bar{r}_{i,j} + p_{i,j}(v_{i',j'}^{(k-1)} - \bar{r}_{i,j}) \right), \quad v_{i,j}^{(k)} = \max \left(v_{i,j}^{(k-1)}, \bar{v}_{i,j}^{(k)} \right) \tag{3}$$

The action (i.e., the target position) is selected to be the pixels (i', j') maximizing the value in a predefined $3x3$ neighborhood $N(i_0, j_0)$ of the agent's current position (i_0, j_0):

$$\pi(s, (i_0, j_0)) = \underset{i',j' \in N(i_0,j_0)}{\arg\max} \; v_{i',j'}^{(k)} \tag{4}$$

Note that the window $N(i_0, j_0)$ is determined by the discrete, pixel-wise actions.

3 Hierarchical Decompositional Reinforcement Learning

We introduce a hierarchical architecture, HiDe, allowing for an explicit functional *decomposition* across layers. Our method achieves temporal abstractions via nested policies. Moreover, our architecture enforces functional decomposition explicitly by reducing the state in each layer to only task-relevant information. The planning layer is responsible for planning a path towards a goal and hence receives global information about the environment. The control layer has access to the agent's internal state and learns a control policy that can achieve subgoals from the planning layer. The layers are jointly-trained by using the hindsight techniques and subgoal testing presented in Section 2.2 to overcome the sparsity of the reward and the non-stationarity caused by off-policy training. Our design significantly improves generalization and makes cross-agent transfer possible (see Section 5).

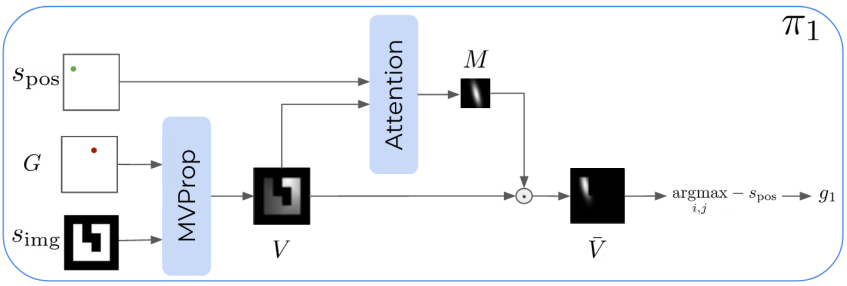


Figure 2: Planning layer $\pi_1(s_{\text{pos}}, s_{\text{img}}, G)$ illustrated for the 2D case. Given an image s_{img} and goal G, the MVProp network computes a value map V. An attention mask M, using the agent's position s_{pos} restricts V to a local subgoal map $\bar{V}$. The policy π_1 selects max value and assigns the control policy π_0 with a sugboal relative to the agent's position. For the 3D case, see Eq. 5-6 and Figure 3c.

3.1 Planning Layer

The planning layer is expected to learn high-level actions over a long horizon, which define a coarse path towards a goal. In related work [13–15], the planning layer learns an *implicit* value function and shares the same architecture as the lower layers. Since the task is learned for a specific environment, generalization is inherently limited. In contrast, we introduce a planning specific layer consisting of several components to learn the map and to find a feasible path to the goal.

Our planning layer for the 2D case is illustrated in Figure 2. We utilize a value propagation network (MVProp) [17] to learn an *explicit* value map which projects the collected rewards onto the environment space. For example, given a discretized 3D model of the environment, a convolutional network determines the per voxel flow probability $p_{i,j,k}$. The probability value of a voxel corresponding to an obstacle should be 0 and that for free passages 1, respectively.

Nardelli et. all [17] only use 2D images and a predefined 3×3 neighborhood of the agent's current position and pass the location of the maximum value in this neighbourhood as goal position to the agent (Equation 4). We extend this to be able to handle both 3D and 2D inputs and augment the MVProp network with an attention model which learns to define the neighborhood dynamically and adaptively. Given the value map V and the agent's current position s_{pos}, we estimate how far the agent can move, modeled by a Gaussian. More specifically, we predict a full covariance matrix Σ with the agent's global position s_{pos} as mean. We later build a mask M of the same size as the environment space s_{img} by using the likelihood function:

$$m_{i,j,k} = \mathcal{N}((i,j,k)|s_{\text{pos}}, \Sigma) \tag{5}$$

Intuitively, the mask defines the density for the agent's success rate. Our planning policy selects an action (i.e., subgoal) that maximizes the masked value map as follows:

$$\begin{aligned} \bar{V} &= M \cdot V \\ \pi_1(s_{\text{pos}}, s_{\text{img}}, G) &= \underset{i,j,k}{\text{argmax}}\ \bar{v}_{i,j,k} - s_{\text{pos}} \end{aligned} \tag{6}$$

where $\bar{v}_{i,j,k}$ corresponds to the value at voxel (i,j,k) on the masked value map $\bar{V}$. Note that the subgoal selected by the planner is relative to the agent's current position s_{pos}, resulting in better generalization as we show in Section 5.1.3. While we present the more general 3D case in Equations 5-6, we reduce the equations by one dimension for the 2D case used in most of our experiments.

The benefits of having an attention model are twofold. First, the planning layer considers the agent dynamics in assigning subgoals which may lead to fine- or coarse-grained subgoals depending on the underlying agent's performance. Second, the Gaussian window allows us to define a dynamic set of actions for the planning policy π_1, which is essential to find a path of subgoals on the map. While the action space includes all pixels of the value map V, it is limited to the subset of only reachable pixels by the Gaussian mask M. We find that this leads to better obstacle avoidance behaviour such as the corners and walls shown in Figure 9 in the Appendix.

Since our planning layer operates in a discrete action space, the resolution of the projected image defines the minimum amount of displacement for the agent, affecting maneuverability. This could be tackled by using a soft-argmax [21] to select the subgoal pixel, allowing to choose real-valued

actions and providing invariance to image resolution. In our experiments we see no difference in terms of the final performance. However, since the former setting allows for the use of DQN [22] instead of DDPG [1], we prefer the discrete action space for simplicity and faster convergence. Both the MVProp (Equation 3) and Gaussian likelihood (Equation 5) operations are differentiable. Hence, MVProp and the attention model parameters are trained by minimizing the standard mean squared Bellman error objective as defined in Equation 1.

3.2 Control Layer

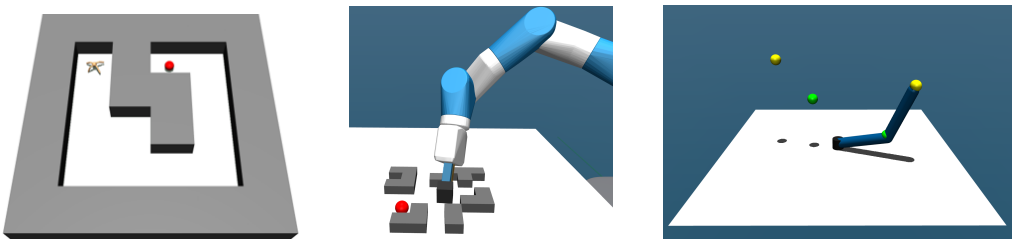
The control layer learns a goal-conditioned control policy. Unlike the planning layer, it has access to the agent's full internal state s_{internal}, including joint positions and velocities. In the control tasks we consider, the agent has to learn a policy to reach a certain goal position, e.g., reach a target position in a navigation domain. We use the hindsight techniques (cf. Section 2.2) so that the control policy receives rewards even in failure cases. All policies in our hierarchy are trained jointly. We use DDPG [1] (Equations 1-2) to train the control layer and DQN [22] for the planning layer.

4 Related Work

Hierarchical Reinforcement Learning Learning hierarchical policies has seen lasting interest [7, 23–26, 16], but many approaches are limited to discrete domains. Sasha et. al [12] introduce FeUdal Networks (FUN), inspired by [16]. In FUN, a hierarchic decomposition is achieved via a learned state representation in latent space, but only works with discrete actions. More recently, off-policy methods that work for goal-conditioned continuous control tasks have been introduced [13–15, 27]. Nachum et. al [14, 15] present HIRO and HIRO-LR, an off-policy HRL method with two levels of hierarchy. The non-stationary signal of the upper policy is mitigated via off-policy corrections. In HIRO-LR, the method is extended by learning a representation of the state and subgoal space space from environment images. In contrast to our approach, both methods use a dense reward function. Levy et. al [13] introduce Hierarchical Actor-Critic (HAC) that can jointly learn multiple policies in parallel via different hindsight techniques from sparse rewards. HAC, HIRO and HIRO-LR consist of a set of nested policies where the goal of a policy is provided by the top layer. In contrast to our method, the same state space is used in all layers, which prohibits transfer of layers across agents. We introduce a modular design to decouple the functionality of individual layers. This allows us to define different state, action and goal spaces for each layer. Our method is closest to HIRO-LR, which also has access to a top-down view image. Although the learned space representation of HIRO-LR can generalize to a mirrored environment, the policies need to be retrained for each task. Contrarily, HiDe generalizes without retraining through the explicit use of the environment image for planning.

Planning in Reinforcement Learning In model-based RL, much attention has been given to learning of a dynamics model of the environment and subsequent planning [28–30]. Eysenbach et. al [31] propose a planning method that performs a graph search over the replay buffer. However, they require to spawn the agent at different locations in the environment and let it learn a distance function in order to build the search graph. Unlike model-based RL, we do not learn state transitions explicitly. Instead, we learn a spatial value map from collected rewards.

Recently, differentiable planning modules that are trained via model-free RL have been proposed [17, 20, 32, 33]. Tamar et. al [20] establish a connection between CNNs and Value Iteration [34]. They propose *Value Iteration Networks* (VIN), where model-free RL policies are additionally conditioned on a fully differrentiable planning module. MVProp [17] extends this by making it more parameter-efficient and generalizable. Our planning layer is based on MVProp. However, we do not rely on a fixed neighborhood mask for action selection. Instead we learn an attention mask which is used to generate intermediate goals for the low-level policy. We also extend our planner to 3D, whereas MVProp is only shown in 2D. Gupta et. al [35] learn a map of indoor spaces and do planning using a multi-scale VIN. However, the robot operates only on a discrete set of macro actions. Nasiriany et. al [36] use a goal-conditioned policy for learning a TDM-based planner on latent representations. Srinivas et. al [33] propose Universal Planning Networks (UPN), which also learn how to plan an optimal action trajectory via a latent space representation. Müller et. al. [37] separate planning from low-level control to achieve generalization by using a supervised planner and a PID-controller. In contrast to our approach, the latter methods either rely on expert demonstrations or need to be retrained in order to achieve transfer to harder tasks.



(a) Complex Ant Navigation (b) Robotic Manipulation (c) 3D Reacher Task

Figure 3: a) The complex navigation training environment from Section 5.1.2. The red sphere indicates the goal. b) The robot manipulation task from Section 5.2. c) The 3D reacher task presented in Section 5.3. The green and yellow spheres indicate the goals for the robot's elbow and tip.

5 Experiments

We evaluate our method on a series of continuous control tasks[2]. Experiment and implementation details are provided in the Appendix. First, we compare to various baseline methods in navigation tasks (see Figure 4) in Section 5.1.1 and provide an ablation study for our design choices in Section 5.1.3. In Section 5.1.2, we show that HiDe can scale beyond 2-3x larger environments (see Figure 3a). Section 5.2 demonstrates that our approach indeed leads to functional decomposition by transferring layers across agents and domains (see Figure 3b) . Finally, we show in Section 5.3 that HiDe can be extended to planning in 3D and use non-image based priors, such as a joint map (see Figure 3c).

5.1 Maze Navigation

We introduce the following task configurations:
Maze Training The training environment, where the task is to reach a goal from a fixed start position.
Maze Backward The training environment with swapped start and goal positions.
Maze Flipped The mirrored training environment.
Maze Random A set of 500 randomly generated mazes with random start and goal positions.

We always train in the Maze Training environment. The reward signal during training is constantly -1, unless the agent reaches the given goal (except for HIRO/HIRO-LR which use an L2-shaped reward). We test the agents on the above task configurations. We intend to answer the following questions:

1. Can our method generalize to unseen test cases and environment layouts?
2. Can we scale to larger environments with more complex layouts (see Figure 3a)?

We compare our method to state-of-the-art HRL approaches including HIRO [15], HIRO-LR [15], HAC [13], and a more conventional navigation baseline dubbed RRT+LL. HIRO-LR is the closest related work, since it also receives a top-down view image of the environment and is a fully learned hierarchy. Our preliminary experiments have shown that HAC and HIRO cannot solve the task when provided with an environment image (see Table 5 in the Appendix), likely due to the increase of the state space by factor of 14. We therefore only show results of HAC and HIRO where they are able to solve the training task, i.e., without accessing an image. To compare against a baseline with complete separation, we introduce RRT+LL. We train a goal-conditioned control policy with RL in an empty environment and attach an RRT planner [18], which finds a path from top-down views via tree-search and does not require training. See Table 3 in the Appendix for an overview of all the baselines.

5.1.1 Simple Maze Navigation

Table 1 (left) summarizes the results for the simple maze tasks. All HRL models successfully learned the training task (see Figure 4). The models' generalization abilities are evaluated in the unseen Maze Backward and Maze Flipped tasks. While HIRO, HIRO-LR and HAC manage to solve the training environment with success rates between 91% and 82%, they suffer from overfitting to the training task, indicated by the 0% success rates in the unseen test scenarios. HIRO-LR, which uses the top-down view implicitly to learn a goal space representation,

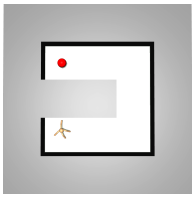


Figure 4: Simple Maze Training.

[2]Videos available at: `https://sites.google.com/view/hide-rl`

Table 1: Success rates of achieving a goal in the maze navigation environments.

	Simple Maze			Complex Maze			
Method	Training	Backward	Flipped	Training	Random	Backward	Flipped
HAC	82 ± 16	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
HIRO	91 ± 2	0 ± 0	0 ± 0	68 ± 8	36 ± 5	0 ± 0	0 ± 0
HIRO-LR	83 ± 8	0 ± 0	0 ± 0	20 ± 21	15 ± 7	0 ± 0	0 ± 0
RRT+LL	25 ± 13	22 ± 12	29 ± 10	13 ± 9	48 ± 3	7 ± 4	5 ± 5
HiDe	$\mathbf{94 \pm 2}$	$\mathbf{85 \pm 9}$	$\mathbf{93 \pm 2}$	$\mathbf{87 \pm 2}$	$\mathbf{85 \pm 3}$	$\mathbf{79 \pm 8}$	$\mathbf{79 \pm 12}$

also fails. For the navigation baseline RRT+LL, the success rate stays between 22% and 29% for all tasks. Although the planner can generalize to different tasks, it cannot learn to cooperate with the low-level control as in our hierarchy. Contrarily, our method is able to achieve 93% and 85% success rates in the generalization tasks *without* retraining. We argue that this is mainly due to the strict separation of concerns, which allows the integration of task-relevant priors, in combination with HiDe's efficient planner.

5.1.2 Complex Maze Navigation

In this experiment, we evaluate how well the methods scale to larger environments with longer horizons. Thus, we train an ant agent in a more complex environment layout (cf. Figure 3a), i.e., we increase the size of the environment by roughly 50% and add more obstacles, thereby also increasing the distance to the final reward. The results are reported in Table 1 (right). HAC fails to learn the training task, while HIRO and HIRO-LR reach success rates of 68% and 20%, respectively. Hence, there is a significant performance drop for both methods if the state-space increases. RRT+LL only reaches success rates between 5% and 13%, except for the Maze Random task. The higher success rates in Maze Random compared to the other test cases can be attributed to the randomization of both the environment layout as well as the start and goal position, which can result in short trajectories without obstacles. Contrary to the baselines, our model's performance decreases only slightly in the training task compared to the simple maze in Section 5.1.1 and also generalizes to all of the unseen test cases. The decrease in performance is due to the increased difficulty of the task. In terms of convergence, we find that in the simple maze case HIRO is competitive with HiDe, due to its reduced state space (no image), but convergence slows with an increase in maze complexity. Contrary, HiDe shows similar convergence behavior in both experiments (cf. Figure 7 in the Appendix). To push the limits of our method, we gradually increase the environment size and observe that only at a 300% increase, the performance drops to around 50% (see Figure 8 in the Appendix). These results indicate that task-relevant information and efficient methods to leverage it are essential components to scale to larger environments. Most failure cases for HiDe arise if the agent gets stuck at a wall and falls over.

5.1.3 Ablation Study

To support the claim that our architectural design choices support the generalization and scaling capabilities, we analyze empirical results of different variants of our method. To show the benefits of relative positions, we compare HiDe against a variant with absolute positions, dubbed HiDe-A. Unlike the case of relative positions, the policy needs to learn all values within the range of the environment dimensions in this setting. Second, we run an ablation study for HiDe with a fixed window size, i.e., we train and evaluate an ant agent on window sizes 3×3, 5×5, and 9×9. Lastly, we compare to a variant where HiDe's decoupled state-space structure is used for training, but the RL-based planner is replaced with an RRT planner. As indicated in Table 2, HiDe-A is competitive in the training task, but fails to match the generalization performance of HiDe. showing that relative positions are crucial for generalization. The learned attention window (HiDe) achieves better or comparable performance than the fixed window variants. Moreover, it eliminates the need for tuning the window size per agent and environment. HiDe-RRT performs significantly worse in all tasks, showing that our learned planner outperforms training with a conventional planner.

Table 2: Ablation study in the simple maze navigation environments from Section 5.1.1.

Methods	Training	Backward	Flipped
HiDe-A	88 ± 2	17 ± 15	36 ± 16
HiDe-3x3	46 ± 32	2 ± 3	31 ± 28
HiDe-5x5	92 ± 4	41 ± 35	82 ± 18
HiDe-9x9	93 ± 4	16 ± 27	79 ± 7
HiDe-RRT	77 ± 9	53 ± 13	72 ± 6
HiDe	$\mathbf{94 \pm 2}$	$\mathbf{85 \pm 9}$	$\mathbf{93 \pm 2}$

5.2 Transfer of Policies

We argue that a key to transferability and generalization behavior in hierarchical RL lies in enforcing a separation of concerns across different layers. To examine whether the overall task is truly split into separate sub-tasks, we perform a set of experiments to demonstrate transfer behavior.

Agent Transfer For this experiment, we train different control agents with HiDe. We then transfer the planning layer of one agent to another agent, e.g., we replace the planning layer of a complex ant agent by the planning layer trained on a simple 2 DoF ball agent. We observe that transfer is possible and only leads to a marginal decrease in performance (cf. Table 10 in the Appendix). Most failure cases arise at corners, where the ball's planner tries to use a path close to the walls. Contrarily, the ant's planner is more conservative as subgoals close to the wall may lead to overturning. We hereby show that transfer of layers between agents is possible and therefore find our hypothesis to be valid. To further demonstrate our method's transfer capabilities, we train a humanoid agent (17 DoF) in an empty environment. We then use the planning layer from a ball agent and connect it as is with the control layer of the trained humanoid (see Figure 5)[3].

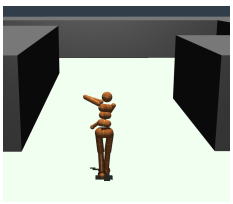


Figure 5: Humanoid agent transfer.

Domain Transfer In this experiment, we demonstrate the capability of HiDe to transfer the planning layer from a simple ball agent, trained on a pure navigation task, to a robot manipulation agent (see Figure 3b). To this end, we train a ball agent with HiDe. Moreover, we train a control policy for a robot manipulation task in the OpenAI Gym "Push" environment [38], which learns to move a cube to a relative position goal. Note that the manipulation task does not encounter any obstacles during training. To attain the compound agent, we attach the planning layer of the ball agent to the manipulation policy (cf. Figure 1a). The planning layer has access to the environment layout and the cube's position, which is a common assumption in robot manipulation tasks. For testing, we generate 500 random environment layouts. As in the navigation experiments in Section 5.1.1, state-of-the-art methods are able to solve these tasks when trained on a single, simple environment layout. However, they do not generalize to other layouts without retraining. In contrast, our evaluation of the compound HiDe agent on unseen testing layouts shows a success rate of 49% (cf. Table 11 in the Appendix). Thus, our modular approach can achieve domain transfer and generalize to different environments.

5.3 Representation of Priors and 3D-Planning

While the majority of our experiments use 2D-images for planning, we show via a proof-of-concept that our method i) can be extended to planning in 3D ii) works with non-top-down view sources of information. To this end, we add a 3DoF robotic arm that has to reach goals in 3D configuration space (see Figure 3c). Instead of a top-down view, we project the robot's joint angles onto a 3D-map which is used as input to our planning layer. We train the 3D variant of our planning layer (see Section 3.1) and 3D CNNs to compute the value map. Our method can successfully solve the task[3]. If the planning space would exceed 3 dimensions, a mapping from higher dimensional representation space to a 2D or 3D latent space could be a potential solution. We leave this for future work.

6 Conclusion

In this paper, we introduce a novel HRL architecture that can solve long-horizon, continuous control tasks with sparse rewards that require planning. The architecture, which is trained end-to-end, consists of a RL-based planning layer which learns an explicit value map and is connected with a low-level control layer. Our method is able to generalize to previously unseen settings and environments. Furthermore, we show that transfer of planners between different agents can be achieved, enabling us to move a planner trained with a simplistic agent to a more complex agent, such as a humanoid or a robot manipulator. The key insight lies in a strict separation of concerns with task-relevant priors that allow for efficient planning and in consequence leads to better generalization. Interesting directions for future work include extensions to higher dimensional planning tasks and multi-agent scenarios.

[3] Videos available at `https://sites.google.com/view/hide-rl`

Acknowledgements

The authors would like to thank Stefan Stevšić and Christoph Gebhardt for their valuable comments and discussions throughout this project, and Alexis E. Block for the help with the preparation of the contributed talk. The project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme grant agreement No 717054.

Broader Impact

Our method HiDe has potential applications in the field of continuous control in robotics, specifically for navigation and manipulation robots. Example applications include a robot navigating a warehouse or pick and place tasks for a manipulator. So far RL methods have struggled to demonstrate good generalization across training and test environments, leaving much work to be done before these are applicable in the real-world. We see our work as one important building block towards this goal and provide one example of how such generalization can be achieved.

While our work is mostly academic in nature and practical applications are still far-off, we see potential benefits in advancements in automation of tedious manual tasks, for example, in healthcare, factories and the supply chain. While automation may render some occupations redundant, it can also create new jobs and hence opportunities to alter the type of work we do. Moreover, robots may fill in or assist jobs in areas with labor shortages, such as in health and elderly care.

One potential risk in deploying RL robots is the lack of interpretability and explainability of actions, which is an active area of research in and of itself. Enforcing a strict separation of subtasks in a hierarchical system such as HiDe may be helpful in understanding the behavior of robots trained with RL. More specifically, as a human, it may not be necessary to understand the inner workings of a robot (the control), but is rather important to understand decisions on a more abstract level (planning). This may also have implications for future systems that let people interactively shape and train complex robotic systems.

References

[1] 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. *arXiv preprint arXiv:1509.02971*, 2015.

[2] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. abs/1504.00702, 2015.

[3] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[4] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, Jan 2019. ISSN 2470-9476.

[5] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[6] S. Christen, S. Stevšić, and O. Hilliges. Guided deep reinforcement learning of control policies for dexterous human-robot interaction. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2161–2167, 2019.

[7] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, August 1999. ISSN 0004-3702. doi: 10.1016/S0004-3702(99)00052-1.

[8] David Andre and Stuart J Russell. State abstraction for programmable reinforcement learning agents. In *AAAI/IAAI*, pages 119–125, 2002.

[9] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. abs/1710.09767, 2017.

[10] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. abs/1704.03012, 2017.

[11] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. abs/1609.05140, 2016.

[12] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. abs/1703.01161, 2017.

[13] Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning Multi-Level Hierarchies with Hindsight. In *International Conference on Learning Representations*, 2019.

[14] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient Hierarchical Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pages 3303–3313, 2018.

[15] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Near-Optimal Representation Learning for Hierarchical Reinforcement Learning. In *International Conference on Learning Representations*, 2019.

[16] Peter Dayan and Geoffrey Hinton. Feudal reinforcement learning. *Advances in Neural Information Processing Systems*, 5, 09 2000. doi: 10.1002/0471214426.pas0303.

[17] Nantas Nardelli, Gabriel Synnaeve, Zeming Lin, Pushmeet Kohli, Philip H. S. Torr, and Nicolas Usunier. Value Propagation Networks. In *International Conference on Learning Representations*, 2019.

[18] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[19] 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 *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[20] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value Iteration Networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162, 2016.

[21] Olivier Chapelle and Mingrui Wu. Gradient descent optimization of smoothed information retrieval metrics. *Information retrieval*, 13(3):216–235, 2010.

[22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. abs/1312.5602, 2013.

[23] Jürgen Schmidhuber. Learning to generate subgoals for action sequences. *IJCNN-91-Seattle International Joint Conference on Neural Networks*, ii:453 vol.2–, 1991.

[24] Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. cs.LG/9905014, 1999.

[25] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10*, NIPS '97, pages 1043–1049, Cambridge, MA, USA, 1998. MIT Press. ISBN 0-262-10076-2.

[26] Amy McGovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 361–368, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1.

[27] Dhruva Tirumala, Hyeonwoo Noh, Alexandre Galashov, Leonard Hasenclever, Arun Ahuja, Greg Wayne, Razvan Pascanu, Yee Whye Teh, and Nicolas Heess. Exploiting hierarchy for learning and transfer in kl-regularized RL. *CoRR*, abs/1903.07438, 2019.

[28] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *ML*, 1990.

[29] Richard S. Sutton, Csaba Szepesvári, Alborz Geramifard, and Michael Bowling. Dyna-style planning with linear function approximation and prioritized sweeping. abs/1206.3285, 2012.

[30] Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. abs/1907.02057, 2019.

[31] Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. Search on the Replay Buffer: Bridging Planning and Reinforcement Learning. *arXiv preprint arXiv:1906.05253*, 2019.

[32] Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. abs/1707.03497, 2017.

[33] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks. abs/1804.00645, 2018.

[34] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2nd edition, 2000. ISBN 1886529094.

[35] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2616–2625, 2017.

[36] Soroush Nasiriany, Vitchyr Pong, Steven Lin, and Sergey Levine. Planning with Goal-Conditioned Policies. In *Advances in Neural Information Processing Systems*, pages 14814–14825, 2019.

[37] Matthias Müller, Alexey Dosovitskiy, Bernard Ghanem, and Vladlen Koltun. Driving policy transfer via modularity and abstraction. In *CoRL*, 2018.

[38] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[39] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[40] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking Deep Reinforcement Learning for Continuous Control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[41] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. DeepMind Control Suite. Technical report, January 2018.

[42] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. 2017.

[43] Atsushi Sakai, Daniel Ingram, Joseph Dinius, Karan Chawla, Antonin Raffin, and Alexis Paques. Pythonrobotics: a python code collection of robotics algorithms. *CoRR*, abs/1808.10703, 2018.

[44] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

A Environment Details

We build on the Mujoco [39] environments used in [14]. The rewards in all experiments are sparse, i.e., 0 for reaching the goal and -1 otherwise. We consider the goal reached if $|s - g|_{\max} < 1$. All environments use $dt = 0.02$. Each episode in the simple maze navigation experiment (Section 5.1.1) is terminated after 500 steps and after 800 steps for the complex maze experiment (Section 5.1.2). In the robot manipulation experiment (Section 5.2), we terminate after 800 steps and in the reacher experiments (Section 5.3) after 200 steps.

A.1 Agents

Our ant agent is equivalent to the one in [13]. In other words, the ant from Rllab [40] with gear power of 16 instead of 150 and 10 frame skip instead of 5. Our ball agent is the PointMass agent from DM Control Suite [41]. We changed the joints so that the ball rolls instead of sliding. Furthermore, we resize the motor gear and the ball itself to match the maze size. For the manipulation robot, we slightly adapt the "Push" task from OpenAI gym [38]. The original environment uses an inverse kinematic controller to steer the robot, whereas joint positions are enforced and realistic physics are ignored. This can cause unwanted behavior, such as penetration through objects. Hence, we change the control inputs to motor torques for the joints. For the robot reacher task, we also adapt the version of the "Reacher" task from OpenAI [38]. Instead of being provided with euclidean position goals, the agents needs to reach goals in the robot's joint angle configuration space.

A.2 Environments

A.2.1 Navigation Mazes

All navigation mazes are modelled by immovable blocks of size $4 \times 4 \times 4$. [14] uses blocks of $8 \times 8 \times 8$. The environment shapes are clearly depicted in Figure 6. For the randomly generated maze, we sample each block with probability being empty $p = 0.7$. The start and goal positions are also sampled randomly at uniform with a minimum of 5 blocks distance apart. Mazes where start and goal positions are adjacent or where the goal is not reachable are discarded. For evaluation, we generated 500 of such environments and reused them (one per episode) for all experiments. We will provide the random environments along with the code.

A.2.2 Manipulation Environments

The manipulation environments differ from the navigation mazes in scale. Each wall is of size $0.05 \times 0.05 \times 0.03$. We use a layout of 9×9 blocks. The object position is the position used for the planning layer. When the object escapes the top-down view range, the episodes are terminated. The random layouts were generated using the same methodology as for the navigation mazes.

A.2.3 Robot Reacher Environments

The robot reacher environment is an adapted version of the OpenAI gym [38] implementation. We add an additional degree of freedom to extend it to 3D space. The joint angles are projected onto a map of size $12 \times 12 \times 12$, where the axes correspond to the joint angles of the robot links. The goals are randomly sampled points in the robot's configuration space.

B Implementation Details

Our PyTorch [42] implementation will be available on the project website[4].

B.1 Baseline Experiments

For HIRO, HIRO-LR and HAC we used the authors' original implementations[5][6]. To improve the performance of HAC, we modified their Hindsight Experience Replay [19] implementation so that

[4] HiDe: `https://sites.google.com/view/hide-rl`

[5] HIRO: `https://github.com/tensorflow/models/tree/master/research/efficient-hrl`

[6] HAC: `https://github.com/andrew-j-levy/Hierarchical-Actor-Critc-HAC-`

they use the FUTURE strategy. More importantly, we also added target networks to both the actor and critic to improve the performance. For HIRO, we ran the hiro_xy variant, which uses only position coordinates for subgoals instead of all joint positions to have a fair comparison with our method. For HIRO-LR, we provide top-down view images of size 5x5x3 as in the original implementation. We train both HIRO and HIRO-LR for 10 million steps as reported in [15]. For RRT+LL, we adapted the RRT python implementation[7] from [43] to our problem. We trained a goal-conditioned low-level control policy in an empty environment with DDPG. During testing, we provide the low-level control policies with subgoals from the RRT planner. We used OpenAI's baselines [44] for the DDPG+HER implementation. When pretraining for domain transfer, we made the achieved goals relative before feeding them into the network. For a better overview of the features the algorithms use, see Table 3.

Features	HiDe	RRT+LL	HIRO-LR	HIRO	HAC	DDPG+HER
Images	✓	✓	✓	x	x	x
Random start pos	x	x	x	x	✓	✓
Random end pos	x	✓	✓	✓	✓	✓
Agent position	✓	✓	x	✓	✓	✓
Shaped reward	x	x	✓	✓	x	x
Agent transfer	✓	✓	x	x	x	x

Table 3: Overview of related work and our method with their respective features. Features marked with a tick are included in the algorithm whereas features marked with a cross are not. See glossary below for a detailed description of the features.

Glossary:
Images: If the state space has access to images.
Random start pos: If the starting position is randomized during training.
Random end pos: If the goal position is randomized during training.
Agent position: If the state space has access to the agent's position.
Shaped reward: If the algorithm learns using a shaped reward.
Agent transfer: Whether transfer of layers between agents is possible without retraining.

B.2 Ablation Baselines

We compare HiDe against several different versions to justify our design choices. In HiDe-A(bsolute), we use absolute positions for the goal and the agent's position. In HiDe-3x3, HiDe-5x5, HiDe-9x9, we compare our learned attention window against fixed window sizes for selecting subgoals. In HiDe-RRT, we use RRT planning [18] for the top-layer, while the lower layer is trained as in HiDe. This is to show that our learned planner improves the overall performance.

B.3 Evaluation Details

For evaluation in the maze navigation experiment of Section 5.1, we trained 5 seeds each for 2.5M steps for the simple maze navigation and 10M steps for the complex maze navigation "Training" environments. We performed continuous evaluation (every 100 episodes for 100 episodes). After training, we selected the best checkpoint based on the continuous evaluation of each seed. Then, we tested the learned policies for 500 episodes and reported the average success rate. Although the agent and goal positions are fixed, the initial joint positions and velocities are sampled from uniform distribution as is standard in OpenAI Gym environments [38]. Therefore, the tables in the results (cf. Table 1) contain means and standard deviations across 5 seeds.

B.4 Network Structure

B.4.1 Planning Layer

Input images for the planning layer were binarized in the following way: each pixel corresponds to one block (0 if it was a wall or 1 if it was a corridor). In our planning layer, we process the input

[7] RRT: `https://github.com/AtsushiSakai/PythonRobotics`

image via two convolutional layers with $3 \times 3 \times 3$ for the 3D case and 3×3 kernels for the 2D case. Both layers have only 1 input and output channel and are padded so that the output size is the same as the input size. We propagate the value through the value map as in [17] $K = 35$ times using a $3 \times 3 \times 3$ max pooling layer (3×3 for 2D). Finally, the value map and position image is processed by 3 convolutions with 32 output channels and $3 \times 3 \times 3$ filter window (3×3 in 2D) interleaved by $2 \times 2 \times 2$ max pool (2×2 for 2D) with ReLU activation functions and zero padding. The final result is flattened and processed by two fully connected layers with 64 neurons, each producing outputs: $\sigma_1, \sigma_2, \sigma_3$ and the respective pairwise correlation coefficients ρ. We use $\mathrm{softplus}$ activation functions for the σ values and $\tanh$ activation functions for the correlation coefficients. The final covariance matrix Σ is given by

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \rho_{1,2}\,\sigma_1\sigma_2 & \rho_{1,3}\,\sigma_1\sigma_3 \\ \rho_{1,2}\,\sigma_1\sigma_2 & \sigma_2^2 & \rho_{2,3}\,\sigma_2\sigma_3 \\ \rho_{1,3}\,\sigma_1\sigma_3 & \rho_{2,3}\,\sigma_2\sigma_3 & \sigma_3^2 \end{pmatrix}$$

so that the matrix is always symmetric and positive definite. For numerical reasons, we multiply by the binarized kernel mask instead of the actual Gaussian densities. We set the values greater than the mean to 1 and the others to zeros.

B.4.2 Control Layer

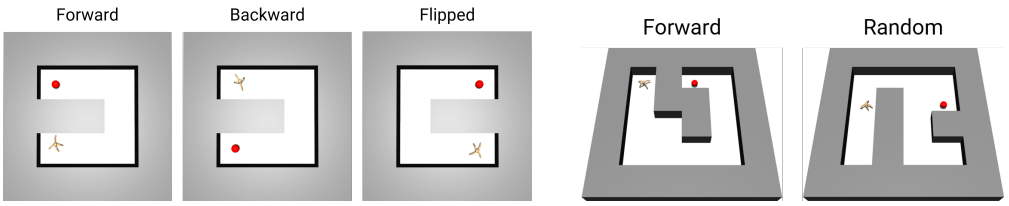
We use the same network architecture for the lower layer as proposed by [13], i.e. we use 3 times a fully connected layer with ReLU activation function. The control layer is activated with $\tanh$, which is then scaled to the action range.

B.4.3 Training Parameters

- Discount $\gamma = 0.98$ for all agents.
- Adam optimizer. Learning rate 0.001 for all actors and critics.
- Soft updates using moving average; $\tau = 0.05$ for all controllers.
- Replay buffer size was designed to store 500 episodes, similarly as in [13]
- We performed 40 updates after each epoch on each layer, after the replay buffer contained at least 256 transitions.
- Batch size 1024.
- No gradient clipping
- Rewards 0 and -1 without any normalization.
- Observations also were not normalized.
- 2 HER transitions per transition using the FUTURE strategy [19].
- Exploration noise: 0.05 and 0.1 for the planning and control layer respectively.

B.5 Computational Infrastructure

All HiDe, HAC and HIRO experiments were trained on 1 GPU (GTX 1080). OpenAI DDPG+HER baselines were trained on 19 CPUs using the baseline repository [44].



(a) Simple Maze Navigation (b) Complex Maze Navigation

Figure 6: Maze environments for the tasks reported in a) Section 5.1.1 and 5.1.3, b) Section 5.1.2. The red sphere indicates the goal, the pink cubes represent the subgoals. Agents are trained in only *Training* and tested in *Backward*, *Flipped*, and *Random* environments.

C Additional Results

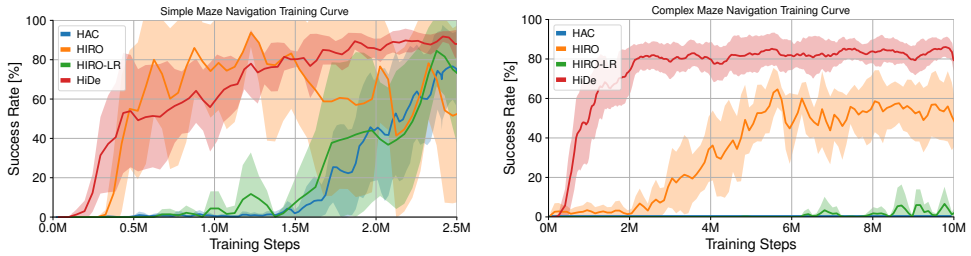
C.1 Maze Navigation

Table 4: Success rates of the individual seeds for achieving a goal in the maze navigation tasks.

	Simple Maze			Complex Maze			
Method	Training	Backward	Flipped	Training	Backward	Flipped	Random
HAC 1	96.4	00.0	00.0	00.0	00.0	00.0	00.0
HAC 2	82.0	00.0	00.0	00.0	00.0	00.0	00.0
HAC 3	85.6	00.4	00.0	00.0	00.0	00.0	00.0
HAC 4	92.8	00.0	00.0	00.0	00.0	00.0	00.0
HAC 5	55.6	00.0	00.0	00.0	00.0	00.0	00.0
HIRO 1	89.0	00.0	00.0	68.0	00.0	00.0	44.6
HIRO 2	89.2	00.0	00.0	64.0	00.0	00.0	36.2
HIRO 3	94.0	00.0	00.0	80.0	00.0	00.0	33.6
HIRO 4	91.3	00.0	00.0	61.0	00.0	00.0	32.8
HIRO 5	90.8	00.0	00.0	81.0	00.0	00.0	34.4
RRT+LL 1	13.8	24.0	19.6	4.8	5.6	1.2	43.6
RRT+LL 2	40.2	6.2	45.2	6.8	4.6	13.6	50.6
RRT+LL 3	20.2	23.8	29.0	17.0	8.4	0.4	47.0
RRT+LL 4	37.2	18.6	30.0	25.6	2.2	3.4	47.2
RRT+LL 5	15.0	38.2	23.4	9.4	13.2	5.8	50.6
HIRO-LR 1	84.2	00.0	00.0	00.0	00.0	00.0	14.4
HIRO-LR 2	80	00.0	00.0	00.0	00.0	00.0	10.4
HIRO-LR 3	79.8	00.0	00.0	48.2	00.0	00.0	16.2
HIRO-LR 4	76	00.0	00.0	20.4	00.0	00.0	26.6
HIRO-LR 5	96.8	00.0	00.0	33.8	00.0	00.0	9.6
HiDe 1	93.4	90.2	94.0	86.4	82.6	87.4	88.6
HiDe 2	90.8	68.2	91.8	83.4	66.0	66.0	81.6
HiDe 3	94.8	91.4	96.2	87.6	84.6	91.0	85.4
HiDe 4	94.0	85.2	92.6	87.2	76.6	77.6	83.2
HiDe 5	96.2	87.8	92.2	89.0	87.4	77.4	87.6

Table 5: Results of the simple maze navigation for HAC and HIRO when provided with an image. Both methods fail to solve the task, as the state space complexity is too high.

	Training	Backward	Flipped
HAC with Image	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
HIRO with Image	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0



(a) (b)

Figure 7: Learning curves with success rates for the training tasks averaged over 5 seeds for a) the simple maze experiment from Section 5.1.1, b) the complex maze experiment from Section 5.1.2. HiDe matches convergence properties of HIRO in the simple maze (left), albeit having a much larger state space in the planning layer. In the more complex maze (right), HiDe shows similar convergence, while the convergence of HIRO slows.

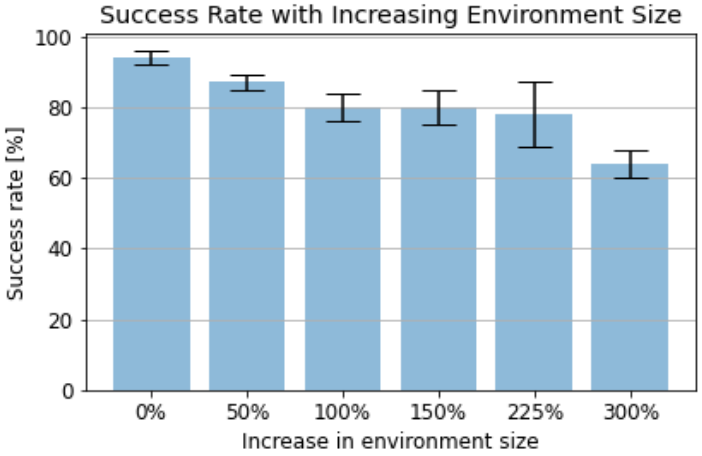


Figure 8: Success rates for the training task when gradually increasing the environment size and number of obstacles. At a 300% increase, HiDe's performance drops to 64%. See Table 6 for the more detailed results.

Table 6: Results for individual seeds of the HiDe experiments with gradually increasing number of obstacles and environment size.

Maze Training	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Averaged
HiDe env + 100%	84.0	83.4	82.5	75.8	75.8	80.3 ± 4
HiDe env + 150%	74.2	85.4	81.4	75.6	84.8	80.3 ± 5
HiDe env + 225%	78.0	69.6	70.8	82.2	91.4	78.4 ± 9
HiDe env + 300%	62.8	58.8	69.8	66.4	64.0	64.4 ± 4

C.2 Ablation Studies

Experiment	Training	Backward	Flipped
HiDe-A 1	88.2	5.4	49.4
HiDe-A 2	89.2	28.4	53.2
HiDe-A 3	84.8	35.8	29.6
HiDe-A 4	91.2	13.6	32.2
HiDe-A 5	88.2	00.0	14.4
HiDe-RRT 1	82.0	35.2	78.8
HiDe-RRT 2	80.0	72.4	75.0
HiDe-RRT 3	73.6	53.6	69.4
HiDe-RRT 4	63.4	50.6	63.4
HiDe-RRT 5	85.6	54.0	74.2
HiDe 3x3 1	28.4	0.0	4.4
HiDe 3x3 2	67.6	0.6	44.0
HiDe 3x3 3	59.6	0.4	36.4
HiDe 3x3 4	0.0	0.0	0.0
HiDe 3x3 5	76.8	6.6	67.8
HiDe 5x5 1	91.0	82.8	97.0
HiDe 5x5 2	94.6	10.6	89.6
HiDe 5x5 3	88.0	72.4	87.2
HiDe 5x5 4	91.2	14.4	83.6
HiDe 5x5 5	97.6	83.6	50.6
HiDe 9x9 1	90.6	9.4	81.2
HiDe 9x9 2	92.4	64.4	79.4
HiDe 9x9 3	89.8	1.2	85.6
HiDe 9x9 4	99.0	4	83.0
HiDe 9x9 5	94.6	3	68.2

Table 7: Individual seed results of the ablation study for the simple maze navigation task.

	Ant 1	Ant 2	Ant 3	Ant 4	Ant 5	Averaged
Training	0.0	78.8	0.0	0.0	0.0	16 ± 35
Random	67.2	88.8	17.6	0.0	0.0	35 ± 41
Backward	0.0	62.2	0.0	0.0	0.0	12 ± 28
Flipped	0.0	79.8	0.0	0.0	0.0	16 ± 36

Table 8: Results for the complex maze navigation task with HiDe-Absolute.

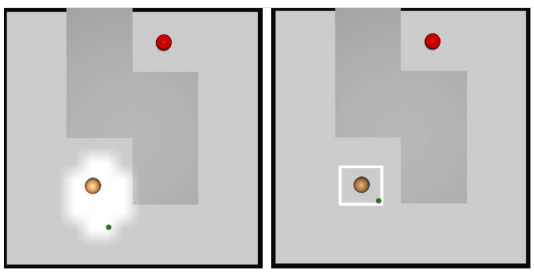


Figure 9: A visual comparison of *(left)* our dynamic attention window with a *(right)* fixed neighborhood. The green dot corresponds to the selected subgoal in this case. Notice how our window is shaped so that it avoids the wall and induces a further subgoal.

C.3 Agent Transfer

	A→B 1	A→B 2	A→B 3	A→B 4	A→B 5	Averaged
Training	99.8	100	100	100	100	100 ± 0
Random	98.4	98.6	98.0	98.6	98.8	98 ± 0
Backward	100	100	99.8	100	100	100 ± 0
Flipped	99.6	100	99.6	100	100	100 ± 0

Table 9: Results of HiDe for ant to ball transfer for individual seeds.

	B→A 1	B→A 2	B→A 3	B→A 4	B→A 5	Averaged
Training	73.4	73.4	71.2	68.2	71.6	72 ± 2
Random	86.2	84.4	83.8	82.6	78.0	83 ± 3
Backward	56.8	48.4	66.4	57.8	58.4	58 ± 6
Flipped	74.4	77.4	80.0	61.6	72.8	73 ± 7

Table 10: Results of HiDe for ball to ant transfer for individual seeds.

C.4 Robotic Arm Manipulation

	Arm 1	Arm 2	Arm 3	Arm 4	Arm 5	Averaged
Random	50	48	47	48	51	49 ± 1

Table 11: Results of the different seeds for the domain transfer experiments.

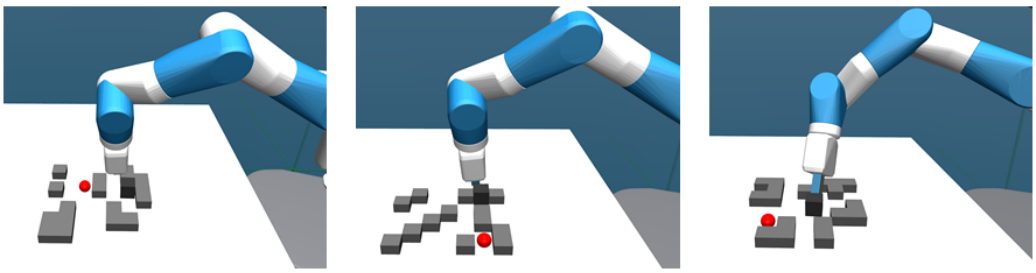


Figure 10: Example of three randomly configured test environments we use to demonstrate the domain transfer of the planning layer from a locomotion domain to a manipulation robot.

D Algorithm

Algorithm 1 Hierarchical Decompositional Reinforcement Learning (HiDe)

Input:

- Agent position s_{pos}, goal position G, and projection from environment coordinates to image coordinates and its inverse $Proj$,$Proj^{-1}$.

Parameters:

1. maximum subgoal horizon $H = 40$, subgoal testing frequency $\lambda = 0.3$

Output:

- $k = 2$ trained actor and critic functions $\pi_0, ..., \pi_{k-1}, Q_0, ..., Q_{k-1}$

for M episodes **do** ▷ Train for M episodes
  $s \leftarrow S_{init}$, $g \leftarrow G_{k-1}$ ▷ Get initial state and task goal
  $train\_top\_level(s, g)$ ▷ Begin training
  Update all actor and critic networks
end for

function $\pi_1(s :: state, g :: goal)$
  $v_{map} \leftarrow MVProp(I, g_1)$ ▷ Run MVProp on top-down view image and goal position
  $\sigma, \rho \leftarrow CNN(v_{map}, Proj(s_{pos}))$ ▷ Predict mask parameters
  $v \leftarrow v_{map} \bigodot \mathcal{N}(\cdot|s_{pos}, \Sigma)$ ▷ Mask the value map
  return $a_1 \leftarrow Proj^{-1}(\arg\max v) - s_{pos}$ ▷ Output relative subgoal corresponding to the max value pixel
end function

function $\pi_0(s :: joints\_state, g :: relative\_subgoal)$
  return $a_0 \leftarrow MLP(s, g)$ ▷ Output actions for actuators
end function

function TRAIN_LEVEL($i :: level, s :: state, g :: goal$)
  $s_i \leftarrow s$, $g_i \leftarrow g$ ▷ Set current state and goal for level i
  for H attempts or until g_n, $i \leq n < k$ achieved **do**
    $a_i \leftarrow \pi_i(s_i, g_i)$ + $noise$ (if not subgoal testing) ▷ Sample (noisy) action from policy
    if $i > 0$ **then**
      Determine whether to test subgoal a_i
      $s_i^{'} \leftarrow train\_level(i-1, s_i, a_i)$ ▷ Train level $i-1$ using subgoal a_i
    else
      Execute primitive action a_0 and observe next state $s_0^{'}$
    end if
    ▷ Create replay transitions
    if $i > 0$ and a_i not reached **then**
      if a_i was subgoal tested **then** ▷ Penalize subgoal a_i
        $Replay\_Buffer_i \leftarrow [s = s_i, a = a_i, r = Penalty, s^{'} = s_i^{'}, g = g_i, \gamma = 0]$
      end if
      $a_i \leftarrow s_i^{'}$ ▷ Replace original action with action executed in hindsight
    end if
    ▷ Evaluate executed action on current goal and hindsight goals
    $Replay\_Buffer_i \leftarrow [s = s_i, a = a_i, r \in \{-1, 0\}, s^{'} = s_i^{'}, g = g_i, \gamma \in \{\gamma, 0\}]$
    $HER\_Storage_i \leftarrow [s = s_i, a = a_i, r = TBD, s^{'} = s_i^{'}, g = TBD, \gamma = TBD]$
    $s_i \leftarrow s_i^{'}$
  end for
  $Replay\_Buffer_i \leftarrow$ Perform HER using $HER\_Storage_i$ transitions
  return $s_i^{'}$ ▷ Output current state
end function