Learning Functionally Decomposed Hierarchies for Continuous Control Tasks with Path Planning

Sammy Christen[1], Lukas Jendele[1], Emre Aksan[1] and Otmar Hilliges[1]

***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 modular transfer of policy layers across hierarchies of different agents. We experimentally show that our method generalizes across unseen test environments and can scale to 3x horizon length compared to both learning and non-learning based methods. We evaluate on complex continuous control tasks with sparse rewards, including navigation and robot manipulation.**

I. Introduction

Reinforcement learning (RL) can solve long horizon control tasks with continuous state-action spaces in robotics [1], [2], such as robot manipulation [3] or human-robot interaction [4]. 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 unseen test environments. Therefore, hierarchical reinforcement learning (HRL) splits the decision making problem into several subtasks at different levels of abstraction [5], [6], often learned separately via curriculum learning [7], [8], [9], or end-to-end via off-policy and goal-conditioning [10], [11], [12]. However, these methods share the 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 environments 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, which has been used in robotics [13], can be beneficial for end-to-end learned hierarchies. More sepcifically, it allows the integration of suitable inductive biases that can be leveraged to achieve generalization across agents and environments. Thus, we propose a 2-level hierarchy (see Fig. 2) 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,

[1]AIT Lab, Department of Computer Science, ETH Zurich, 8092 Zurich, Switzerland sammy.christen@inf.ethz.ch

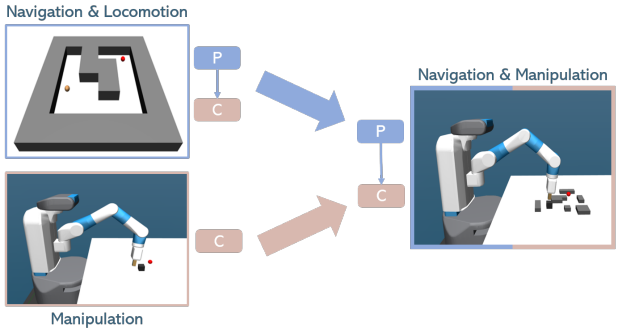


Fig. 1. Our functionally decomposed hierarchical architecture allows training higher-level **P**lanning policies with simple agents and combining them with more complex **C**ontrol policies. Example transfer of the planning of a simple 2DoF ball agent trained on a locomotion task to a manipulation robot to solve more complex tasks such as pushing a cube around obstacles.

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 [14]. Second, the 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 Fig. 1).

In our framework, which we call HiDe, a goal-conditioned control policy π_c 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 that are provided by the planning layer policy π_p. The planning layer has access to task-relevant information, e.g., an occupancy grid, 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 [10], [11] causes an explosion of the state-space complexity and hence leads to failure as we show in Section V-A. We propose a specialized, efficient planning layer, based on MVProp [15] with an added learned dynamic agent-centric attention window which transforms the task-relevant prior into a value map. The action of π_p is the position that maximizes the masked value map and is fed as a subgoal to the control policy π_c. 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 robot arm pushing a block. However, we show as a proof of concept that HiDe can also work in non-Euclidean space and plan 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 [16]. We then show that our method can scale to 3x longer horizons and generalize to randomly configured layouts. Furthermore, 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, combined 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 zero-shot transfer of individual policy layers across different agents and domains.

II. Related Work

1) Hierarchical Reinforcement Learning: Learning hierarchical policies has seen lasting interest [5], [14], [17], [18], but many approaches are limited to discrete domains. Sasha et. al [9] introduce FeUdal Networks (FUN), inspired by [14]. 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 [10], [11], [12], [19], [**?**]. Nachum et. al [11], [12] present HIRO and HIRO-LR, an off-policy HRL method with a two level 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 from environment images. In contrast to our approach, both methods use a dense reward function. Levy et. al [10] introduce Hierarchical Actor-Critic (HAC) that can jointly learn hierarchical policies from sparse rewards via different hindsight techniques. 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. Similar to [20], [21], we introduce a modular design to decouple the functionality of individual layers. This allows us to define different state-action and goal spaces per layer. In contrast to [20], our method is trained end-to-end and scales to longer horizons. Our method is closest to HIRO-LR, which also has access to a top-down view map. Although the learned representation of HIRO-LR can generalize to a mirrored environment, the policies still need to be retrained. Contrarily, HiDe generalizes without retraining through the explicit use of the map for planning.

2) 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 [22], [23]. Eysenbach et. al [24] 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 [15], [25], [26], [27]. Tamar et. al [25] establish a connection between CNNs and Value Iteration [28]. They propose *Value Iteration Networks* (VIN), where model-free RL policies are additionally conditioned on a fully differrentiable planning module. MVProp [15] extends this by making it more parameter-efficient and generalizable. Our planning layer is based on MVProp. However, we do not rely on selecting an action from a fixed neighborhood mask (see Section IV). 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 [29] 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 [30] use a goal-conditioned policy for learning a TDM-based planner on latent representations. Srinivas et. al [27] propose Universal Planning Networks (UPN), which also learn how to plan an optimal action trajectory via a latent space representation. Müller et. al. [13] 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.

III. Background

A. 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 $\pi : \mathcal{S} \times \mathcal{G} \rightarrow \mathcal{A}$, which maximizes the expected return. We use 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:

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

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

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

B. Hindsight Techniques

In HAC, Levy et. al [10] 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 [31], *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. Hence, a transition with a penalty reward is added to the replay buffer if the current lower-level policy π_{i-1} cannot reach the provided subgoal within H attempts. We find all three techniques effective and apply them to our model during training.

C. Value Propagation Networks

Tamar et. al [25] introduce value iteration networks (VIN) for path planning problems. Nardelli et. al [15] 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:

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

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):

$$\pi(s,(i_0,j_0)) = \underset{i',j' \in N(i_0,j_0)}{\operatorname{argmax}} v_{i',j'}^{(k)} \quad (4)$$

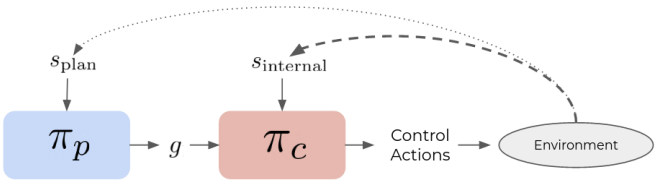


Fig. 2. Our HRL architecture. The planning layer policy π_{p} receives global environment information s_{plan} for planning and outputs a goal g for the lower layer. The low-level control learns a goal-conditioned control policy π_{c} from proprioceptive state information s_{internal} to reach the provided goal.

IV. Hierarchical Decompositional Reinforcement Learning

We introduce a 2-layer hierarchical architecture, HiDe, which allows for an explicit functional *decomposition* across layers (see Fig. 2). 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 goal-conditioned 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 III-B 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 V).

A. 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 the related work [10], [11], [12], 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 Fig. 3. We utilize a value propagation network (MVProp) [15] 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 [15] 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

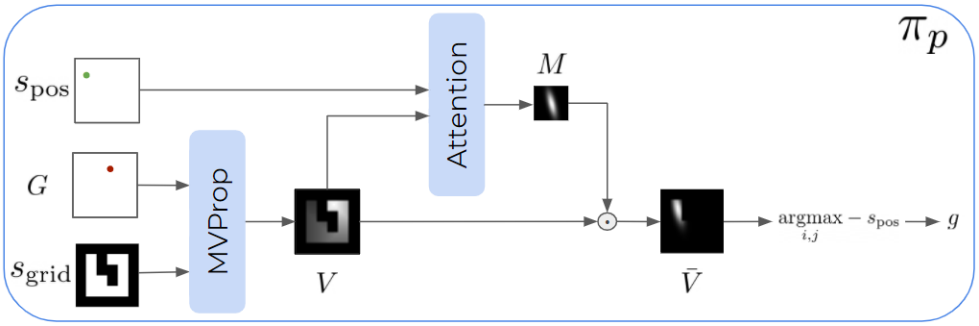


Fig. 3. Planning layer $\pi_{\text{p}}(s_{\text{pos}},s_{\text{grid}},G)$ illustrated for the 2D case. Given an occupancy grid s_{grid} and goal G, the MVProp network computes a value map V. An attention mask M, using the agent's position s_{pos} restricts V to a local subgoal map $\bar{V}$. The policy π_{p} selects max value and assigns the control policy π_{c} with a sugboal g relative to the agent's position. For the 3D case, see Equations 5-6 and Fig. 4c.

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_{grid} 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:

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

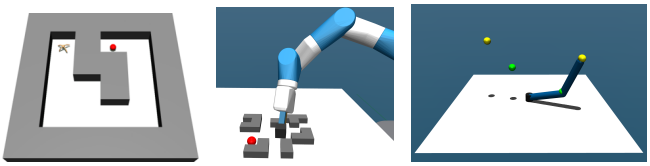
where $\bar{v}_{i,j,k}$ corresponds to the value at voxel (i,j,k) on the masked value map $\bar{V}$. Note that the subgoal selected by the planner is relative to the agent's current position s_{pos}, resulting in better generalization as we show in Section V-B. While we present the more general 3D case 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_{p}, 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 Fig. 7.

Since our planning layer operates in 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 [32] 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 performance. However, since the former setting allows for the use of DQN [33] 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.

B. Control Layer

The control layer learns a goal-conditioned control policy. Unlike the planning layer, it has access to the agent's 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. The agent-centric goal is provided by the planning layer. We use the hindsight techniques (cf. Section III-B) 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 [33] for the planning layer.



(a) Maze navigation (b) Manipulation (c) 3D Reacher

Fig. 4. a) The complex navigation training environment from Section V-A.2. The red sphere indicates the goal. b) The robot manipulation task from Section V-C.2. c) The 3D reacher task presented in Section V-D. The green and yellow spheres indicate the goals for the robot's elbow and tip.

V. Experiments

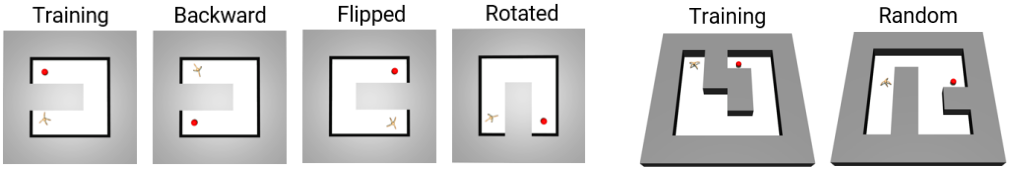
We evaluate our method on a series of continuous control tasks which are implemented in MuJoCo [34]. First, we compare to various baseline methods in navigation tasks (see Fig. 5a) in Section V-A.1 and show in Section V-A.2 that HiDe can scale beyond 3x larger environments (see Fig. 5b). In Section V-B, we provide an ablation study for our design choices. Section V-C demonstrates that our approach indeed leads to functional decomposition by transferring layers across agents and domains (see Fig. 4b). We show in Section V-D that HiDe can be extended to planning in 3D and use non-image based priors, such as a joint map (see Fig. 4c). The code with pretrained models along with videos of the experiments is made publicly available[2].

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

TABLE I

SUCCESS RATES FOR THE MAZE NAVIGATION EXPERIMENTS.

	Simple Maze				Complex Maze				
Method	Training	Backward	Flipped	Rotated	Training	Random	Backward	Flipped	Rotated
HAC	82±16	0±0	0±0	0±0	0±0	0±0	0±0	0±0	0±0
HIRO	91±2	0±0	0±0	0±0	68±8	36±5	0±0	0±0	0±0
HIRO-LR	83±8	0±0	0±0	0±0	20±21	15±7	0±0	0±0	0±0
RRT+LL	25±13	22±12	29±10	24±7	13±9	48±3	7±4	5±5	8±8
RRT-S+LL	47±16	37±18	47±17	46±6	32±13	62±5	20±12	11±10	19±17
RRT-S+LL*	77±7	74±9	61±12	79±5	69±5	79±6	54±13	63±11	59±10
HiDe	**94±2**	**85±9**	**93±2**	**84±8**	**87±2**	**85±3**	**79±8**	**79±12**	**78±17**



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

Fig. 5. Maze environments for the tasks reported in a) Section V-A.1 and V-B, b) Section V-A.2. The red sphere indicates the goal. Agents are always trained in the *Training* environment and tested in the *Backward*, *Flipped*, *Rotated* and *Random* environments.

A. 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 Rotated The training environment rotated by 90°.

Maze Random A set of 500 randomly generated mazes with random start and goal positions.

We always train in the Maze Training environment. The rewards in the experiments are sparse, i.e., 0 for reaching the goal and -1 otherwise (except for HIRO/HIRO-LR with an L2-shaped reward). We test the agents on the above task configurations. We intend to answer the following questions:

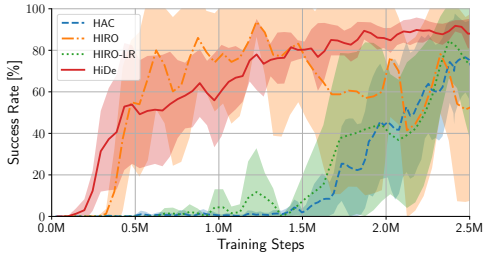
1) Can our method generalize to unseen test cases and environment layouts?
2) Can we scale to larger environments with more complex layouts (see Fig. 5b)?

We compare our method to state-of-the-art HRL approaches including HIRO [11], HIRO-LR [12], HAC [10], and a set of more conventional navigation baselines using RRT [16]. HIRO-LR is the closest related work, since it receives an occupancy grid and is a fully learned hierarchy.

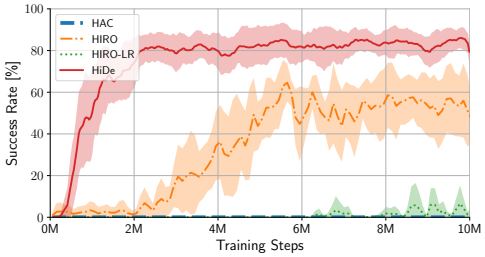
Our preliminary experiments showed that HAC and HIRO cannot solve the task when provided with an occupancy grid, likely due to the increase of the state space by factor of 14. We therefore only show results of HAC and HIRO where they are able to solve the training task, i.e., without accessing an occupancy grid. 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 [16], which finds a path from top-down view via tree-search and does not require training. We also introduce RRT-S+LL and RRT-S+LL* with an added safety margin to the planner such that it does not select subgoals close to the walls. In RRT-S+LL*, we add more noise to the starting states, which leads to more robust control policies. All methods were trained for 2.5 million steps (simple maze, Fig. 5a) and 10 million steps (complex maze, Fig. 5b). The presented results are averaged over 5 seeds in all experiments.

1) Simple Maze Navigation: Table I (left) summarizes the results for the simple maze tasks shown in Fig. 5a. All HRL models successfully learned the training task (Maze Training). The models' generalization abilities are evaluated in the unseen Maze Backward, Flipped and Rotated tasks. While HIRO, HIRO-LR and HAC manage to solve the training environment with success rates between 91% and 82%, they overfit to the training task, indicated by the 0% success rates in the unseen test cases. HIRO-LR, which uses the top-down view implicitly to learn a goal space representation, also fails. For the RRT+LL baseline, the success rate stays between 22% and 29% for all tasks, with failures arising mostly due to the agent falling over. The success rate slightly improves in RRT-S+LL due to the added safety margin. Adding more noise to the starting states helps to further increase the performance (RRT-S+LL*). However, although the planner can generalize to different tasks, it cannot learn to cooperate with the low-level control as in our hierarchy. Our method is able to achieve 93% and 84% 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 and the emerging curriuculum for the control policy.

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. Fig. 5b), 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 I (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



(a) Simple Maze Navigation Training Curves



(b) Complex Maze Navigation Training Curves

Fig. 6. Learning curves with success rates for the training tasks averaged over 5 seeds for a) the simple maze experiment from Section V-A.1, b) the complex maze experiment from Section V-A.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.

TABLE II

EFFICIENCY COMPARISON. WE MEASURE THE NUMBER OF CONTROL ACTIONS REQUIRED TO REACH THE TARGET IN SUCCESSFUL TRIALS.

Methods	Simple Maze	Complex Maze
HIRO	184±100	323±28
RRT+LL	195±15	354±42
RRT-S+LL*	157±19	247±5
HiDe	169±7	250±32
HiDe-S	**141±13**	**209±5**

TABLE III

ABLATION STUDY SUCCESS RATES FOR THE SIMPLE MAZE.

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	**94±2**	**85±9**	**93±2**

methods if the state-space increases. The best performing RRT baseline, RRT-S+LL* reaches success rates between 54% and 69%, except for the Maze Random task. Note that HiDe does not have access to such a safety margin, yet learns to avoid the walls. The higher success rates in Maze Random compared to the other test cases is due 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 and also generalizes to all of the unseen test cases. The decrease in performance is due to the increased difficulty of the task. We analyze the convergence behavior between the different methods in Fig. 6. We find that in the simple maze, HIRO is competitive with HiDe, due to its reduced state space (no grid) and access to a shaped L2-based reward, but convergence slows with an increase in complexity. Contrarily, HiDe shows similar convergence behavior in both experiments. 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%. These results indicate that task-relevant information and efficient methods to leverage it are essential components to scale to larger environments. Most remaining failure cases for HiDe arise if the agent flips over.

3) Efficiency of Policies: To assess the efficiency of the policies, we evaluate the average number of steps it takes an agent to reach a target in successful trials. We compare against HIRO as the best performing learning-based method and the different RRT variants in both the simple and complex maze. The results are summarized in Table II. We find that adding safety margins to HiDe (HiDe-S) generates the best results and use it as an upper baseline. HIRO takes inefficient routes and tends to cut corners, likely due to the L2-based reward it relies on. The naive RRT+LL baseline shows similar behavior. Adding a safety margin and using more robust low-level policies (RRT-S+LL*) mitigates this and increases the performance. HiDe shows similar scores as the RRT-S+LL* baseline, although it does not have access to the safety margin, indicating that it implictly learns to avoid goals close to the walls. The remaining performance gap between HiDe and HiDe-S implies that HiDe may still sometimes provide goals too close to the walls, which could be improved upon, e.g., by adding an additional penalty to the planner for subgoals where the agents comes in contact with a wall.

B. 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 III, 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. We also observe qualitatively that behavior around corners is better compared to the fixed window, as it can learn to avoid corners (see Fig. 7). Moreover, it eliminates the need for tuning the window size per agent and environment. HiDe-RRT performs

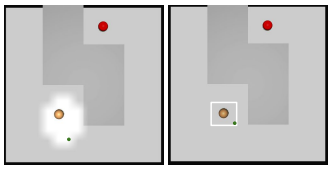


Fig. 7. A visual comparison of *(left)* our dynamic attention window with a *(right)* fixed neighborhood. The green dot corresponds to the selected subgoal. Notice how our window is shaped so that it avoids the wall.

significantly worse in all tasks, showing that our learned planner outperforms training with RRT and hence creates a beneficial curriculum for the control policy.

C. 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. We therefore perform a set of experiments to demonstrate transfer behavior.

1) Agent Transfer: For this experiment, we train different control agents with HiDe in the complex maze environment (see Fig. 5b). 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 and vice-versa. We compare against a recent method that allows for agent transfer [20] and uses a discriminator to improve the domain shift. We apply the zero-shot transfer and low-level imitation versions of [20], which we call "HMT Zero-Shot" and "HMT Discrim.", and omit the variant with fine-tuning of the high-level policies.

The results are illustrated in Table IV. For the transfer of the ball's planner to the ant (Ball PL $\rightarrow$ Ant CL), we observe that transfer with HiDe outperforms the HMT baselines and only leads to a marginal decrease in performance compared to the results reported in Table I. 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. When transferring the ant's planner to the ball agent (Ant PL $\rightarrow$ Ball CL), there is no decrease in performance for HiDe. On the other hand, the HMT variants only reach low success rates of 12-20% in the simple maze and fail in the complex maze, because training the ant planner seldomly succeeds. We hereby show that HiDe can achieve transfer of layers between agents and outperform a state-of-the-art method [20]. 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 ball agents and connect it as is with the control layer of the humanoid. We report results on two different sets of 500 random layouts, Complex Random is the same set from Section V-A and Simple Random is a set with less clutter. The success rates of 37% and 19% (cf. Table IV) indicate the fragility of the humanoid control policies and leave room for improvement. We will release our test environments to foster further research.

2) Domain Transfer: In this experiment, we demonstrate zero-shot transfer of HiDe's planner from a simple ball agent, trained on a pure navigation task, to a robot manipulation agent (see Fig. 4b). 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 [35], 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. Fig. 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 2 sets of 500 random environment layouts, one set with few obstacles (Simple Random) and a more cluttered set (Complex Random). As in the navigation experiments in Section V-A.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% and 33% (cf. bottom row of Table IV) in the two random test sets. Most failure cases arise when the robot's end-effector gets stuck at an obstacle. Thus, our modular approach can achieve domain transfer and generalize to different environments, although leaving room for improvement.

TABLE IV

TRANSFER EXPERIMENT SUCCESS RATES.

	Method	Simple Maze	Complex Maze
Ball PL $\rightarrow$ Ant CL	HMT Zero-Shot [20]	42±18	16±14
	HMT Discrim. [20]	54±14	20±18
	HiDe	**84±11**	**72±2**
Ant PL $\rightarrow$ Ball CL	HMT Zero-Shot [20]	12±26	0±0
	HMT Discrim. [20]	20±45	0±0
	HiDe	**100±0**	**100±0**
		Simple Random	Complex Random
Ball PL $\rightarrow$ Humanoid CL	HiDe	**37±3**	**19±4**
Ball PL $\rightarrow$ Robot Arm CL	HiDe	**49±1**	**33±3**

D. Representation of Priors and 3D-Planning

While the majority of our experiments use 2D-grids for planning, we provide 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 Fig. 4c). Instead of a top-down view, we project the robot's joint angles onto a 3D-map which is used as input to our planner. The planner finds subgoals that bring the robot into the target pose. The low-level policy learns a controller that applies motor torques in order to reach the given subgoal poses of the planner. We train the 3D variant of our planning layer (see Section IV-A) and 3D CNNs to compute the value map. Our method can successfully solve the task (see supplementary video). If the planning space exceeds 3 dimensions, a mapping from higher dimensional representation space to a latent space could be a solution.

VI. DISCUSSION & CONCLUSION

In this paper, we introduce a novel HRL architecture, HiDe, that can solve long-horizon, continuous control tasks with sparse rewards. The architecture, which is trained end-to-end, consists of an 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 unseen settings and environments. Furthermore, we show that zero-shot transfer of planners between different agents can be achieved, enabling us to move a planner trained on a simple agent to a more complex agent, such as a humanoid. The key insight lies in a strict separation of concerns with task-relevant priors that allow for efficient planning and in consequence better generalization. Several issues remain. In the simulation, we operate under the ideal assumption that an occupancy grid is available, which may not be the case in real-world settings. A potential solution to the problem could be to first use SLAM with a simple robot to obtain an occupancy grid of the environment and subsequently apply HiDe with an added sensing module to estimate the position of the agent in the environment. Furthermore, when transferring the learned attention mask, the planner does not take into account the capabilities of the control agent. This leads to a decrease in performance (cf. Section V-C), which could be addressed by fine-tuning of the higher layer policies, such as suggested in [20]. Other interesting directions for future work include experiments on a real-world robot or multi-agent settings.

VII. Acknowledgements

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme grant agreement No. StG-2016-717054.

References

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations*, Y. Bengio and Y. LeCun, Eds., 2016.

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[3] N. Vulin, S. Christen, S. Stevšić, and O. Hilliges, "Improved learning of robot manipulation tasks via tactile intrinsic motivation," *IEEE Robotics and Automation Letters*, 2021.

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

[5] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning," *Artif. Intell.*, vol. 112, no. 1-2, pp. 181–211, Aug. 1999.

[6] D. Andre and S. J. Russell, "State abstraction for programmable reinforcement learning agents," in *AAAI/IAAI*, 2002, pp. 119–125.

[7] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman, "Meta Learning Shared Hierarchies," in *International Conference on Learning Representations*, 2018.

[8] P.-L. Bacon, J. Harb, and D. Precup, "The Option-Critic architecture," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[9] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "FeUdal Networks for Hierarchical Reinforcement Learning," in *Proceedings of the 34th International Conference on Machine Learning*, 2017, p. 3540–3549.

[10] A. Levy, G. Konidaris, R. Platt, and K. Saenko, "Learning Multi-Level Hierarchies with Hindsight," in *International Conference on Learning Representations*, 2019.

[11] O. Nachum, S. S. Gu, H. Lee, and S. Levine, "Data-efficient Hierarchical Reinforcement Learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 3303–3313.

[12] O. Nachum, S. Gu, H. Lee, and S. Levine, "Near-Optimal Representation Learning for Hierarchical Reinforcement Learning," in *International Conference on Learning Representations*, 2019.

[13] M. Müller, A. Dosovitskiy, B. Ghanem, and V. Koltun, "Driving policy transfer via modularity and abstraction," in *CoRL*, 2018.

[14] P. Dayan and G. Hinton, "Feudal reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 5, 09 2000.

[15] N. Nardelli, G. Synnaeve, Z. Lin, P. Kohli, P. H. S. Torr, and N. Usunier, "Value Propagation Networks," in *International Conference on Learning Representations*, 2019.

[16] S. LaValle, "Rapidly-exploring random trees : a new tool for path planning," *The annual research report*, 1998.

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

[18] T. G. Dietterich, "Hierarchical reinforcement learning with the MAXQ value function decomposition," *Journal of artificial intelligence research*, vol. 13, pp. 227–303, 2000.

[19] D. Tirumala, H. Noh, A. Galashov, L. Hasenclever, A. Ahuja, G. Wayne, R. Pascanu, Y. W. Teh, and N. Heess, "Exploiting Hierarchy for Learning and Transfer in KL-regularized RL," *CoRR*, vol. abs/1903.07438, 2019.

[20] D. Hejna, L. Pinto, and P. Abbeel, "Hierarchically decoupled imitation for morphological transfer," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. Virtual: PMLR, 13–18 Jul 2020, pp. 4159–4171.

[21] G. Konidaris and A. Barto, "Building portable options: Skill transfer in reinforcement learning," in *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, ser. IJCAI'07. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007, p. 895–900.

[22] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *ML*, 1990.

[23] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba, "Benchmarking Model-Based Reinforcement Learning," *CoRR*, vol. abs/1907.02057, 2019.

[24] B. Eysenbach, R. R. Salakhutdinov, and S. Levine, "Search on the replay buffer: Bridging planning and reinforcement learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 15 246–15 257.

[25] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value Iteration Networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2154–2162.

[26] J. Oh, S. Singh, and H. Lee, "Value Prediction Network," in *Advances in Neural Information Processing Systems*, 2017, pp. 6118–6128.

[27] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn, "Universal Planning Networks: Learning Generalizable Representations for Visuomotor Control," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, J. G. Dy and A. Krause, Eds., vol. 80. PMLR, 2018, pp. 4739–4748.

[28] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 2nd ed. Athena Scientific, 2000.

[29] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive mapping and planning for visual navigation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2616–2625.

[30] S. Nasiriany, V. Pong, S. Lin, and S. Levine, "Planning with Goal-Conditioned Policies," in *Advances in Neural Information Processing Systems*, 2019, pp. 14 814–14 825.

[31] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight Experience Replay," in *Advances in Neural Information Processing Systems*, 2017, pp. 5048–5058.

[32] O. Chapelle and M. Wu, "Gradient descent optimization of smoothed information retrieval metrics," *Information retrieval*, vol. 13, no. 3, pp. 216–235, 2010.

[33] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *CoRR*, vol. abs/1312.5602, 2013.

[34] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

[35] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.