HRL4IN: Hierarchical Reinforcement Learning for Interactive Navigation with Mobile Manipulators

Chengshu Li **Fei Xia** **Roberto Martín-Martín** **Silvio Savarese**
Stanford University
{chengshu, feixia, robertom, ssilvio}@stanford.edu

Abstract: Most common navigation tasks in human environments require auxiliary arm interactions, e.g. opening doors, pressing buttons and pushing obstacles away. This type of navigation tasks, which we call *Interactive Navigation*, requires the use of mobile manipulators: mobile bases with manipulation capabilities. Interactive Navigation tasks are usually long-horizon and composed of heterogeneous phases of pure navigation, pure manipulation, and their combination. Using the wrong part of the embodiment is inefficient and hinders progress. We propose **HRL4IN**, a novel **H**ierarchical **RL** architecture for **I**nteractive **N**avigation tasks. HRL4IN exploits the exploration benefits of HRL over flat RL for long-horizon tasks thanks to temporally extended commitments towards subgoals. Different from other HRL solutions, HRL4IN handles the heterogeneous nature of the Interactive Navigation task by creating subgoals in different spaces in different phases of the task. Moreover, HRL4IN selects different parts of the embodiment to use for each phase, improving energy efficiency. We evaluate HRL4IN against flat PPO and HAC, a state-of-the-art HRL algorithm, on *Interactive Navigation* in two environments - a 2D grid-world environment and a 3D environment with physics simulation. We show that HRL4IN significantly outperforms its baselines in terms of task performance and energy efficiency. More information is available at https://sites.google.com/view/hrl4in.

Keywords: Hierarchical Reinforcement Learning, Mobile Manipulation, Interactive Navigation

1 Introduction

In recent years, agents learning with reinforcement have achieved new levels of proficiency, solving from complex games [1, 2] and video games [3, 4], to real-world robotic tasks [5, 6]. In robotics, this success has mostly been restricted to either navigation [7, 8, 9, 10] or manipulation with stationary arms [2, 11, 12, 13, 14]. The goal in navigation is to change the agent's location without changing the environment's configuration, whereas the goal in manipulation with stationary arms is to change the environment's configuration without changing the agent's location. However, many tasks in real human environments require an agent capable of achieving both, sometimes even simultaneously.

A common case is robot navigation in realistic human environments. This task requires interactions with the environment such as opening doors or pushing obstacles away. We call the family of navigation tasks that require interactions *Interactive Navigation*. Solving interactive navigation problems requires a mobile base with interaction capabilities, i.e. a *mobile manipulator*. With this embodiment, the agent can change its location and/or the configuration of the environment, by using only the base, the arm(s) or a combination of both. A naive use of the entire embodiment for every phase of the task is inefficient: large parts of the task can be solved with only the base or the arm. Therefore, an efficient solution for Interactive Navigation needs to, first, understand what phases of the task are pure navigation, manipulation or both, and second, solve these subtasks efficiently.

The structure mentioned above is well suited for a hierarchical reinforcement learning (HRL) solution, where the high level learns the phase type and sets a subgoal, and the low level learns to achieve it. However, most existing HRL approaches focus on subgoals that are in the same space as the final goal, e.g. intermediate locations towards a final location [15, 16, 17] or intermediate joint configurations towards a final one [18]. In our problem setup, however, the subgoals are *heteroge-*

3rd Conference on Robot Learning (CoRL 2019), Osaka, Japan.

neous: while the final goal is to move the base to the desired location, the intermediate subgoals may require the agent to move its end-effector to a given pose, e.g., to the location of a door handle.

In this work, we present **HRL4IN**, a hierarchical reinforcement learning solution for heterogeneous subtask spaces that can be applied to solve Interactive Navigation tasks. Our solution learns the most efficient use of pure navigation, pure manipulation, or their combination. The key for efficiency is to allow the high level to decide not only on the subgoals for the low level, but also on the capabilities that the low level is allowed to use to achieve it.

In summary, the main contributions of our work are: first, we propose HRL4IN, a novel learning algorithm suited for mobile manipulators. Second, our solution is, to the best of our knowledge, the first deep HRL approach for continuous control that shows success in heterogeneous tasks - tasks with multiple phases where the goals lay on different spaces. Third, we show that HRL4IN can solve Interactive Navigation tasks, navigation tasks that require interaction with the environment. We believe that explicitly defining and addressing this new type of navigation task is relevant and practically impactful for robots that aim to operate in human environments.

2 Related Work

Hierarchical Reinforcement Learning: One of the hardest types of tasks for reinforcement learning is long-horizon tasks with sparse reward. Exploring the environment through random sequences of actions requires a prohibitive number of samples until a solution can be found. To alleviate this problem, researchers have proposed hierarchical RL to incorporate temporal abstraction [17].

Bacon et al. [19] proposes an option-critic architecture that learns temporally extended options and their termination conditions, together with the policy over them. Heess et al. [20] pretrains the low-level policy on simple tasks and then trains the high-level policy to output appropriate signals that modulate low-level behaviors. Our proposed method, HRL4IN, is closer to the hierarchical approaches where a high-level policy explores by setting subgoals for a low-level policy to achieve [16, 21, 15]. The subgoals can be set directly in the same space as the original goal [15], or in an embedding space [21]. While the former is more data efficient, the latter has the potential to learn more informative representation. Nachum et al. [22] propose a way to balance between representation efficiency and capabilities.

Similar to [15, 18], our subgoals represent a desired observation that the high-level policy wants to induce. However, we also exploit the fact that a heterogeneous task can be solved more efficiently by setting subgoals in different spaces during different phases of the task. Specifically, our high-level policy not only outputs a subgoal, but also selects a part of the embodiment to use for the current phase. Finally, unlike [16], we do not manually construct subgoals apriori, which may require extensive domain knowledge, allowing the high-level policy to learn subgoals from scratch.

Interactive Navigation: While the literature on autonomous robot navigation is vast and prolific [23, 24, 25], less attention has been paid to navigation problems that require interactions with the environment, what we call Interactive Navigation. In the robot control literature, several papers have approached the problem of opening doors with mobile manipulators [26, 27, 28, 29]. However, these approaches focus on this single phase and not on the entire Interactive Navigation task.

Stilman and Kuffner [30] studied interactive navigation from a geometric motion planning perspective. In their problem setup, the agent has to reason about the geometry and arrangement of obstacles to decide on a sequence of pushing/pulling actions to rearrange them to allow navigation. They named this type of problems Navigation Among Movable Objects (NAMO) [31, 32, 33, 34]. Their solution requires knowledge of the geometry of the objects to plan, and the search problem is restricted to 2D space. Our work has two main differences from the previous line of work on NAMO. First, most NAMO solutions assume full observability of the environment as it is required for sampling-based motion planning. On the contrary, HRL4IN only assumes partial observability through virtual depth images. Second, NAMO solutions find a suitable plan in configuration space and rely on a separate controller for the execution of the plan. HRL4IN generates full policies that can not only overcome local minimum in the reward function but also directly output agent commands. The aforementioned properties of our solution render it model-free and end-to-end trainable.

The work by Konidaris et al. [35] is close to us because it approaches Interactive Navigation as a hierarchical reinforcement learning problem. Their algorithm learns to sequence actions to solve simple tasks and reuses these sequences to solve tasks composed of simple subtasks. However,

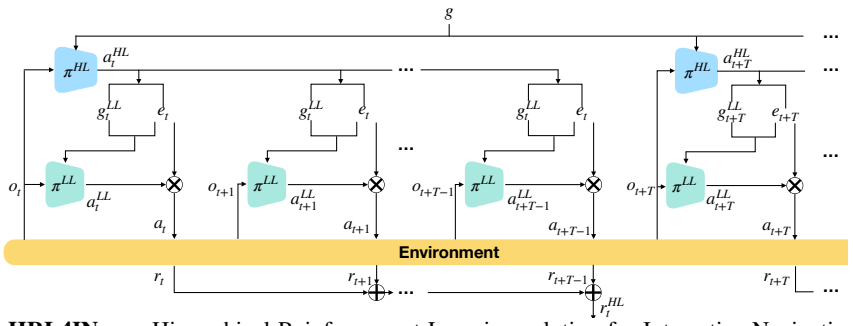


Figure 1: **HRL4IN**, our Hierarchical Reinforcement Learning solution for Interactive Navigation; both the high-level (blue) and the low-level (green) policies receive an observation from the environment; the high-level policy generates a subgoal and an embodiment selector for the low-level policy for the next T steps, conditioned on the final goal; the low-level policy outputs the robot commands based on the observation, conditioned on the action of the high-level policy; the robot commands are masked and executed by the environment, which gives a reward and the next observation; the sum of these rewards during the T steps will be used to train the high-level policy. More details can be found in Algorithm 1 in Appendix B.1.

different from ours, their solution requires a set of predefined controllers as actions for the policy during training. The options are learned in the low-level space and reused during testing. HRL4IN directly controls lower level commands for the mobile manipulator, without hard-coded controllers.

Task and Motion Planning: Interactive Navigation and NAMO can alternatively be solved by task and motion planning (TaMP) methods. Task and motion planning solves long-horizon problems by combining task planner and motion planner. Task planner can reason over large sets of states and motion planner can compute exact paths for the robot to execute. An integration of task and motion planner often uses a hierarchical approach [36]. Much of the work in TaMP factorizes the task into sub-problems and abstractions that the solutions learn to use. Existing TaMP approaches are capable of solving complicated mobile manipulation tasks [37, 38]. Konidaris et al. [39] learns symbolic representations by interacting with an environment and observing the effects of the actions. Wolfe et al. [36] build a system that reuses optimal solutions to state-abstracted sub-problems across the search space. Our work differs from the above as we don't explicitly formulate our problem as TaMP but rather implicitly allow the high-level policy to learn to do the planning and the low-level policy to learn to do the control.

While most of the previous work on abstraction selection refers to state abstraction [40] (mapping primitive state space S to an abstract state space $h(S)$), our work performs abstraction selection in the action space. The work by Konidaris and Barto [41] relates to us as they also learns abstractions for the action space.

3 Background

We assume that the underlying problem can be modelled as a discrete-time goal-conditioned MDPs $\mathcal{M} = (\mathcal{S}, \mathcal{G}, \mathcal{A}, R, \mathcal{T}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{G}$ is the goal space, $\mathcal{A}$ is the action space, $R(s, g, a)$ is the reward function, $\mathcal{T}(s'|s, a)$ is the transition dynamics, and $\gamma \in [0, 1)$ is the discount factor. In our work, $\mathcal{G}$ is the space of final goals for the high level and subgoals for the low level. At each time step t, an agent observes state s_t, executes action a_t sampled from policy $\pi(a_t|s_t, g_t)$, receives from the environment a new observation s_{t+1} and a reward r_t. In reinforcement learning, the goal of the agent is to maximize expected discounted return $\mathbb{E}_{a\sim\pi}[\sum_{t=0}^{T-1} \gamma^t r_t]$. To do so, a family of RL solutions use the policy gradient theorem [42] to optimize for the policy parameters θ. In this work, we adopt Proximal Policy Optimization Algorithms (PPO) [43], a stable and widely adopted on-policy model-free policy gradient algorithm as the building block and baseline for HRL4IN.

4 HRL4IN: Hierarchical RL for Interactive Navigation

Interactive navigation tasks are intrinsically multi-phase. In some phases the agent only needs to navigate using its base, or to interact with the environment using its arm. In other phases, it needs to move its base and arm in coordination. Using the base or arm when not needed can be counter-productive: it is less energy efficient and increases the probabilities of undesired collisions.

Based on these insights, we propose HRL4IN, a hierarchical reinforcement learning solution for Interactive Navigation that learns to select the elements of the embodiment that are necessary for each phase of the task. To do so, the high-level policy not only sets subgoals for the low-level policy, but also selects which part of the embodiment to use.

The main structure of HRL4IN is depicted in Fig. 1. Our solution is composed by two policies, a high-level (*HL*) and a low-level (*LL*) policy. Both policies use the same observations $o_t \in \mathcal{O}$ that contain both elements that can be changed by the agent (*mutable*) such as the poses of its base and arm, and elements that cannot be changed by the agent (*immutable*) such as the door location. A full description of the observation space is included in Sec. 5.2 and Appendix A. In the following, we explain the main characteristics of the high-level and the low-level policies.

4.1 High-Level Policy

The high-level policy π^{HL} acts at a different time scale from that of the original MDP: it acts at time steps t' that correspond to every T time steps of the low-level policy, or fewer, if the low-level policy converges to the subgoal. The high-level policy receives an observation $o_{t'} \in \mathcal{O}$ and generates an action $a^{HL}_{t'} \in \mathcal{A}^{HL}$, conditioned on the final goal, $g \in \mathcal{G}$.

Similar to previous HRL approaches [15, 18], one component of the high-level action, $a^{HL}_{t'}$, is the subgoal $g^{LL}_{t'} \in \mathcal{G}^{LL}$. $g^{LL}_{t'}$ indicates a desired relative change of certain mutable components of the observation, denoted as $x_{t'}$, yielding a subgoal $x_{t'} + g^{LL}_{t'}$ for the low-level policy to achieve. The subgoal is valid for T time steps of the low-level policy, unless it achieves the subgoal within T time steps (x_t is close enough to $x_{t'} + g^{LL}_{t'}$), in which case HRL4IN queries the high-level policy for the next high-level action. To represent the subgoal with respect to the current observation x_t, the original subgoal $g^{LL}_{t'}$ is updated by $g^{LL}_t = x_{t'} + g^{LL}_{t'} - x_t$ for every subsequent time step.

The other component of the high-level action, $a^{HL}_{t'}$, is an embodiment selector $e_{t'}$. The embodiment selector is a discrete variable that decides which capabilities the low-level policy can use to achieve the current subgoal: navigation (base), manipulation (arm), or both. The selector plays two crucial roles. First, it is used to compute an *action mask*, m^{act}, that deactivates some components of the embodiment that are not necessary for the current phase (i.e., base or arm) by forcing the velocity of certain joints to be 0. Second, it is used to compute a *subgoal mask*, m^{sg}, that determines the dimensions of the observation that the low-level policy should bring closer to the subgoal to obtain *intrinsic reward*. Using the embodiment selector, the high-level policy decides on the phase type within the Interactive Navigation task for a period of T steps: navigation, manipulation or both.

The reward for a high-level action is the sum of the rewards from the environment during the T or fewer than T time steps of its execution.

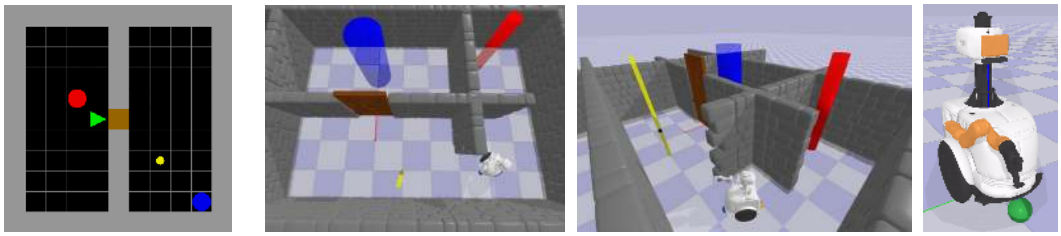
4.2 Low-Level Policy

The low-level policy, π^{LL}, acts at every discrete time step t of the MDP. It also receives an observation $o_t \in \mathcal{O}$ and generates an action, $a^{LL}_t \in \mathcal{A}^{LL}$, conditioned on the last high-level action, $a^{HL}_{t'} \in \mathcal{A}^{HL}$, which includes the subgoal $g^{LL}_{t'}$ (transformed to be relative to the current observation as explained above) and the action and subgoal masks derived from the embodiment selector $e_{t'}$.

The low-level actions, a^{LL}_t, are robot commands for the base and the arm. In our Interactive Navigation experiments, the exact form of these actions depends on the environment (see Sec. 5.1). The low-level actions are masked by the action mask derived the embodiment selector $e_{t'}$, which *zeroes out* the action components that are not necessary for the current task phase: $a_t = a^{LL}_t \otimes m^{act}_{t'}$.

We define the current subgoal distance D_t as the distance between the current observation x_t and the subgoal $x_{t'} + g^{LL}_{t'}$, masked by the subgoal mask $m^{sg}_{t'}$ derived from the high-level embodiment selector $e_{t'}$: $D_t = \sqrt{\sum_{i=0}^{N-1} m^{sg}_{t'}[i] \cdot (x_t[i] - (x_{t'}[i] + g^{LL}_{t'}[i]))^2}$, where N is the dimension of $g^{LL}_{t'}$. We train the low-level policy using intrinsic reward defined by $r^{LL}_t = D_t - D_{t+1}$. Intuitively, the intrinsic reward encourages the low-level policy to make progress towards the subgoal.

For each subgoal, the low-level policy has T time steps to achieve it. We consider a subgoal achieved if $D_t = 0$ (discrete state space) or $D_t < t_{sg}$ (continuous state space). When the subgoal is achieved or the low-level policy runs out of time, HRL4IN queries the high-level policy for the next $a^{HL}_{t'}$.



(a) (b)

Figure 2: Environments of our experiments; (a) `Interactive ToyEnv`: the agent (green triangle) has to navigate from the initial location (red circle) to the end location (blue circle) by opening the door that connects the rooms; (b) `Interactive Gibson Environment`: top-down (left) and perspective (middle) views of the environment where the agent, embodied as a JackRabbot mobile manipulator (right), has to navigate from the initial location (red cylinder) to the end location (blue cylinder) also by opening the door that connects the rooms; In HRL4IN the high-level policy guides the low-level policy by setting subgoals, depicted as a yellow dot in (a) and a yellow cylinder with a black sphere for the desired end-effector position in (b).

5 Experimental Evaluation

5.1 Environment Setup

In order to demonstrate the effectiveness of our approach, we build two environments with different levels of abstraction and physical complexity: `Interactive ToyEnv` and `Interactive Gibson Environment`. More details are explained below and included in Appendix A.

Interactive ToyEnv: `Interactive ToyEnv` is a 2D grid-world environment with discrete state and action space. It consists of $k \times k$ cells, where each cell could be free space, wall, door, or occupied by the agent. The agent is a simplification of a mobile manipulator that can navigate around the environment and interact with the door using its arm. At each time step, the agent can choose among navigation and/or manipulation actions that result in a deterministic outcome. Navigation actions include `turn-left`, `turn-right`, `go-forward` and `no-op`. Manipulation actions include `slide-up` (the door), `slide-down`, and `no-op`. Manipulation actions are only effective (i.e., change the door state) when the agent is in the cell in front of the door and is facing the door. Several (5) consecutive `slide-up` are necessary to fully open the door. The observation space includes the global map, the door state, the goal, and proprioceptive information such as the pose of the agent.

Interactive Gibson Environment: `Interactive Gibson Environment` is a 3D environment with continuous state and action space. It is an evolution of the Gibson Environment [44] that offers fast rendering and physics simulation. The main difference is that in `Interactive Gibson Environment`, additional objects (e.g. doors) apart from the agent can be added and their dynamics can be simulated, which allows us to train agents for Interactive Navigation tasks. To simplify the contact physics, we assume the agent has a grasping primitive that grabs the door handle when the end-effector is close enough (distance under $t_{grasp} = 0.1\,\mathrm{m}$). Grasping creates a ball joint between the end-effector and the door handle that constrains translation but allows rotation. Once the door is fully opened, the end-effector ends the grasp primitive and releases the door handle.

In `Interactive Gibson Environment`, the agent controls the mobile manipulator JackRabbot composed by a six degrees of freedom (DoF) Kinova arm mounted on a non-holonomic two-wheeled Segway mobile base. The agent commands actions for the mobile manipulator in continuous action space: desired velocities for each joint, including the two wheels and five joints of the arm (seven DoF in total). The observations to the agent include a depth map, and proprioceptive information such as the poses of the base and the end-effector, the joint configurations, and the goal position.

5.2 Experimental Setup

In our experiments, we use PPO [43] as the policy gradient algorithm for both the high-level and the low-level policies of HRL4IN. When training both policies at the same time, the changing behavior of the low-level policy poses a non-stationary problem for the high-level policy: the same high-level action could result in different low-level actions, and therefore different state transitions and rewards, depending on the current training stage of the low-level policy. We believe an on-policy algorithm like PPO will stabilize the training process and encourage policy convergence. Details about the network structure and hyperparameters of HRL4IN and PPO can be found in Appendix B.2 and B.3.

To evaluate the performance of HRL4IN, we compare with two baselines: flat PPO and Hindsight Actor-Critic (HAC), a state-of-the-art HRL framework. Since the official implementation of HAC published by the authors does not support discrete action, we only compare HRL4IN with HAC in `Interactive Gibson Environment`. We run experiments in `Interactive Gibson Environment` and `Interactive ToyEnv` with 5 and 7 different random seeds, respectively.

Interactive ToyEnv: The goal in `Interactive ToyEnv` is defined by the discrete coordinates of a desired agent's location, $g = (x, y)_{WF} \in \mathbb{N}^2$, where WF indicates a global world frame. The agent starts at a random location in the left room with a random orientation. Subgoals are defined as the desired relative change in the agent's x, y location, orientation and door state, $g_t^{LL} \in \mathbb{N}^4$. The embodiment selector can assume three values: `base-only`, `arm-only`, and `base+arm`.

The reward is $r_t = \mathbf{w}^T\mathbf{r}$, where the reward vector is defined as $\mathbf{r} = [r_{success}, r_{energy}]$ and the weights to sum them up are $\mathbf{w} = [w_{success}, w_{energy}]$. The reward weights are defined in Appendix A. In the reward vector, $r_{success} = \mathbb{1}\{(x, y)_t = g\}$ indicates goal convergence, $r_{energy} = 0$ if there is no actuation, $r_{energy} = 1$ if there is arm **or** base actuation, and $r_{energy} = 2$ if there are base **and** arm actuations. An episode ends when the agent reaches the goal or runs out of time. We use time scale $T = 4$ for the high-level policy.

Interactive Gibson Environment: The goals in `Interactive Gibson Environment` is defined by the continuous coordinates of a desired agent's base location, $g = (x, y)_{WF} \in \mathbb{R}^2$. The agent starts at a fixed location with a random orientation and is assumed to converge to the goal when the distance between the agent base position and the goal is smaller than $t_g = 0.5\,\mathrm{m}$. Subgoals g_t^{LL} are defined as the desired relative change in the agent's end-effector position in either 3D or 2D (xy-plane) space, depending on the embodiment selection. The embodiment selector can assume two values: `base-only` and `base+arm`. The threshold to assume subgoal convergence is $t_{sg} = 0.05\,\mathrm{m}$.

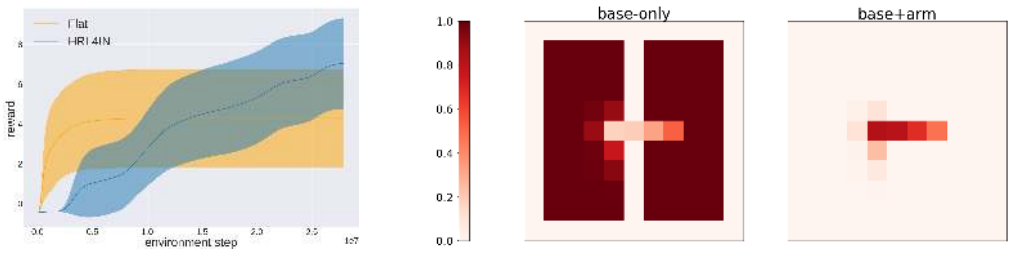
The environment returns different rewards in three different phases of the Interactive Navigation task: moving towards the door, opening the door, and moving towards to the goal. The reward is $r_t = \mathbf{w}^T\mathbf{r}$, where the reward vector is defined as: $\mathbf{r} = [r_{success}, r_{progress}, r_{energy}, r_{collision}]$ and the weights to sum them up are $\mathbf{w} = [w_{success}, w_{progress}, w_{energy}, w_{collision}]$. The reward weights are defined in Appendix A. In the reward vector, $r_{success} = \mathbb{1}\{|(x, y)_t - g|^2 < t_g\}$ indicates goal convergence; $r_{progress}$ is a small positive value if the agent is making progress for the current stage (e.g. getting closer to the door handle or opening the door partially); $r_{energy} = \mathbb{1}\{\dot{q}_{base} > t_{energy}\} + \mathbb{1}\{\dot{q}_{arm} > t_{energy}\}$, where $\dot{q}_{base}$ and $\dot{q}_{arm}$ are joint velocities of the base wheels and the arm joints; $r_{collision} = 1$ if the agent collides with the environment or itself. An episode ends when the agent reaches the goal, runs out of time or tips over. We use time scale $T = 50$ (equal to 5 seconds of world time) for the high-level policy.

5.3 Results

Interactive ToyEnv: Our experimental results for the `Interactive ToyEnv` are depicted in Fig. 3. While being initially less sample efficient, HRL4IN achieves higher reward than flat PPO. HRL4IN learns to solve the task for 5 out of 7 random seeds while flat PPO only does so for 3 out of 7 random seeds. Given the sparse reward, HRL4IN is able to explore more efficiently than flat PPO through temporal extended commitments towards subgoals. The plot also shows that once flat PPO learns to solve the task, the policy consistently repeats the successful strategy. HRL4IN, on the other hand, has less stable training process since the same high-level action could result in different executions by the low-level policy that is being trained simultaneously. Our best performing model achieves 100% success rate with average episode length of 19.2 steps. The optimal policy, computed analytically using the shortest path, takes on average 14 steps, so our solution is $< 1.5\times$ optimal.

We also analyze the probability of using different parts of the embodiment at different locations of the rooms (Fig. 3b). We observe that the high-level policy of HRL4IN learns to use only base at almost all locations except those near the door, where it uses both base and arm. It is energy-efficient to use only base when far away from the door because the agent cannot interact with it. The agent never uses arm-only because unless the agent is 1) at the position in front of the door and 2) facing it, it needs to use both base and arm to make progress towards the goal and arm-only is overall too restrictive. Fig. 4 illustrates three different phases of a sample trajectory of our best model.

Interactive Gibson Environment: Our experimental results for the `Interactive Gibson Environment` are depicted in Fig. 5. Similar to the pattern observed for the `Interactive ToyEnv`, the reward increases faster initially for flat PPO than for HRL4IN: flat PPO quickly starts to exploit



(a) Reward over time for HRL4IN and flat PPO; HRL4IN achieves higher reward across 7 random seeds despite being less sample efficient initially.

(b) Probability of using different parts of the embodiment; the high-level policy learns to set base-only subgoals everywhere except when the agent is near the door, where it sets base+arm subgoals. Arm-only usage is excluded because it's overly restrictive and never used.

Figure 3: Experimental results for `Interactive ToyEnv`

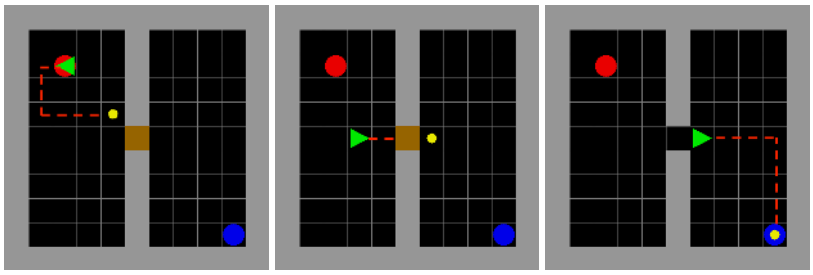


Figure 4: Three phases of a sample trajectory for `Interactive ToyEnv`; the agent starts at a random location (red) in the left room and is rewarded for reaching the goal (blue) in the right room; (*Left*) the agent first sets a base-only subgoal (yellow) to approach the door; (*Middle*) the agent then sets a base+arm subgoal to get to the front of the door and to open it; (*Right*) the agents sets a base-only subgoal to reach the final goal.

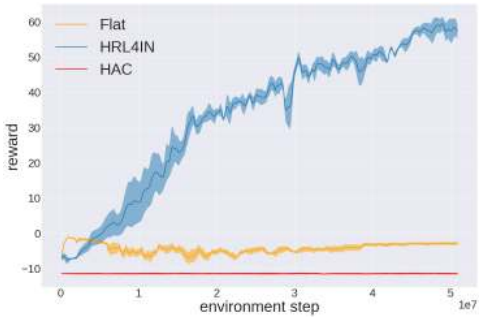
the experiences from the initial exploration and gets stuck in a local maximum of return that corresponds to the scenario in which the agent moves towards the wall separating the initial room and the final room (closer to the final goal). On the contrary, HRL4IN performs a deeper exploration in subgoal directions and achieves much higher reward in the long run. The other baseline HAC completely fails to solve the task. Our hypothesis for HAC's poor performance is that our task poses a difficult exploration problem in which there exists a discrepancy between the final goal distribution and the hindsight goal distribution. The subgoals that HAC learns to achieve during training does not contribute to achieving the final goal. We observe that most transitions sampled from the replay buffer include hindsight goals very close to the initial position and high virtual rewards, and few include real goals and low real rewards. Our best performing HRL4IN policies has close to 100% success rate after 50M training steps. A few failure cases are caused by the agent accidentally colliding with the door and closing it back on its way to the final goal.

We have run ablation studies to understand the effect of embodiment selection and different reward terms in `Interactive Gibson Environment` on model performance (Table 1). First, we ablate the embodiment selector for HRL4IN and find that it achieves comparable reward and success rate, but fails to save any energy. We compute energy in a symbolic sense since we can't accurately measure energy consumption from the simulator: the agent consumes 2 units of energy when using base and arm, and 1 unit of energy when using only base or arm. Furthermore, we ablate each reward term except $r_{success}$ and observe that all reward terms are necessary for training a successful, energy-efficient policy: without $r_{progress}$ and $r_{collision}$, the agent fails to solve the task; without r_{energy}, the agent fails to save energy or learn meaningful embodiment selection.

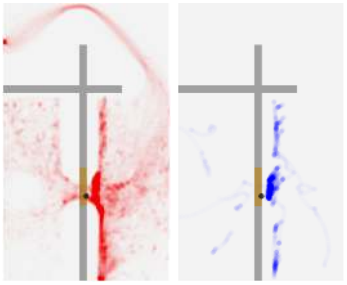
We also analyze the use of different parts of the embodiment at different locations of the environment (Fig. 5b). We plot the embodiment selections from 100 evaluation episodes on the room layout. We observe that the high-level policy learns to use only base most of the time to save energy. It sometimes uses base and arm when in front of the door to grasp the door handle. Then it tends to switch back to use only base again because now that the end-effector is attached to the door handle, moving the base is sufficient to open the door. Finally, it uses only base in the final room since the door is already open. Fig. 6 illustrates four different phases of a sample trajectory of our best model. More experimental results and policy visualizations can be found in Appendix C.

Model	Reward Terms	Success Rate	Reward	Energy-Saving
HRL4IN (full)	**All**	**0.963**	**64.3**	**0.453**
HRL4IN (w/o emb. sel.)	All	0.912	61.5	0.0
HRL4IN (full)	No r_{energy}	0.934	63.4	0.235
HRL4IN (full)	No $r_{collision}$	0.0	9.0	0.421
HRL4IN (full)	No $r_{progress}$	0.0	-1.2	0.498

Table 1: Ablation Study Results for `Interactive Gibson Environment`. We show here all reward terms and embodiment selection are necessary to achieve the best success rate and energy-saving.

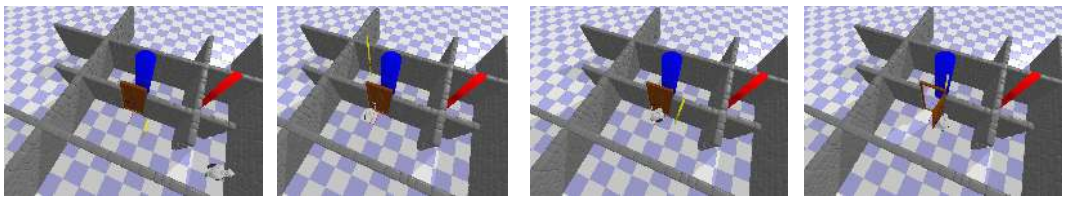


(a) Reward over time for HRL4IN, flat PPO and HAC; HRL4IN significantly outperforms both baselines in the final reward, indicating strong exploration capabilities of our algorithm.



(b) Scatter plot of embodiment selections at different locations of 100 trajectories; the high-level policy learns to set base-only subgoals at most of the locations to save energy and set base-arm subgoals when in front of the door to grasp the door handle.

Figure 5: Experimental results for `Interactive Gibson Environment`.



(a) (b) (c) (d)

Figure 6: Four phases of a sample trajectory for `Interactive Gibson Environment`; (a) the agent navigates from the initial location (red cylinder) towards the door by following a base-only subgoal (yellow cylinder); (b) when in front of the door, the agent grabs the door handle with the arm following a base+arm subgoal (black sphere in the yellow cylinder); (c) the agent moves its base to open the door by setting a base-only subgoal behind itself (d) the agent follows a base-only subgoal to reach the final goal (blue cylinder).

6 Conclusion

We presented HRL4IN, a hierarchical Reinforcement Learning solution for Interactive Navigation tasks, tasks that require motion of the base and the arm of a mobile manipulator. The high-level policy of HRL4IN sets subgoals in different subspaces (navigation and/or manipulation) for the low-level policy to achieve. Moreover, the high level can also decide on the part of the embodiment (base, arm, base and arm) to use at each temporally extended phase to perform deep exploration. We evaluate HRL4IN against flat PPO and HAC in two environments, a grid-world environment with discrete action space, and a complex, realistic robotics environment with continuous action space. Our results indicate that HRL4IN achieves higher reward and solve the tasks more successfully and efficiently. In future work, we would like to 1) examine other relevant tasks that require interactive navigation and 2) transfer the learned policy to the real world.

Acknowledgments

We acknowledge the support of Toyota Research Institute ("TRI") but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity. Fei Xia would like to acknowledge the support of Stanford Graduate Fellowship.

References

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[2] S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[4] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

[5] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba. Learning dexterous in-hand manipulation. *CoRR*, 2018. URL http://arxiv.org/abs/1808.00177.

[6] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673, 2018.

[7] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3357–3364. IEEE, 2017.

[8] L. Tai, G. Paolo, and M. Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 31–36. IEEE, 2017.

[9] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis. Learning navigation behaviors end-to-end with autorl. *IEEE Robotics and Automation Letters*, 4(2):2007–2014, 2019.

[10] A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, and J. Davidson. Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5113–5120. IEEE, 2018.

[11] M. Večerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[12] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[13] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[14] B. Beyret, A. Shafti, and A. A. Faisal. Dot-to-dot: Explainable hierarchical reinforcement learning for robotic manipulation. 2019.

[15] O. Nachum, S. S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3303–3313, 2018.

[16] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016.

[17] R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[18] A. Levy, G. Konidaris, R. Platt, and K. Saenko. Learning multi-level hierarchies with hindsight. *International Conference on Learning Representations*, 2019.

[19] P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[20] N. Heess, G. Wayne, Y. Tassa, T. Lillicrap, M. Riedmiller, and D. Silver. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*, 2016.

[21] 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-Volume 70*, pages 3540–3549. JMLR. org, 2017.

[22] O. Nachum, S. Gu, H. Lee, and S. Levine. Near-optimal representation learning for hierarchical reinforcement learning. *International Conference on Learning Representations*, 2018.

[23] G. N. DeSouza and A. C. Kak. Vision for mobile robot navigation: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 24(2):237–267, 2002.

[24] F. Bonin-Font, A. Ortiz, and G. Oliver. Visual navigation for mobile robots: A survey. *Journal of intelligent and robotic systems*, 53(3):263–296, 2008.

[25] K. Chen, J. P. de Vicente, G. Sepúlveda, F. Xia, A. Soto, M. Vázquez, and S. Savarese. A behavioral approach to visual navigation with graph localization networks. *Robotics: Science and Systems (RSS)*, 2019.

[26] L. Peterson, D. Austin, and D. Kragic. High-level control of a mobile manipulator for door opening. In *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000)(Cat. No. 00CH37113)*, volume 3, pages 2333–2338. IEEE, 2000.

[27] A. J. Schmid, N. Gorges, D. Goger, and H. Worn. Opening a door with a humanoid robot using multi-sensory tactile feedback. In *2008 IEEE International Conference on Robotics and Automation*, pages 285–291. IEEE, 2008.

[28] A. Petrovskaya and A. Y. Ng. Probabilistic mobile manipulation in dynamic environments, with application to opening doors. In *IJCAI*, pages 2178–2184, 2007.

[29] A. Jain and C. C. Kemp. Behavior-based door opening with equilibrium point control. Technical report, Georgia Institute of Technology, 2009.

[30] M. Stilman and J. J. Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. *International Journal of Humanoid Robotics*, 2(04):479–503, 2005.

[31] M. Stilman and J. Kuffner. Planning among movable obstacles with artificial constraints. *The International Journal of Robotics Research*, 27(11-12):1295–1307, 2008.

[32] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour. Manipulation planning among movable obstacles. In *Proceedings 2007 IEEE international conference on robotics and automation*, pages 3327–3332. IEEE, 2007.

[33] J. Van Den Berg, M. Stilman, J. Kuffner, M. Lin, and D. Manocha. Path planning among movable obstacles: a probabilistically complete approach. In *Algorithmic Foundation of Robotics VIII*, pages 599–614. Springer, 2009.

[34] M. Levihn, J. Scholz, and M. Stilman. Hierarchical decision theoretic planning for navigation among movable obstacles. In E. Frazzoli, T. Lozano-Perez, N. Roy, and D. Rus, editors, *Algorithmic Foundations of Robotics X*. Springer, 2013.

[35] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto. Autonomous skill acquisition on a mobile manipulator. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[36] J. Wolfe, B. Marthi, and S. Russell. Combined task and motion planning for mobile manipulation. In *Twentieth International Conference on Automated Planning and Scheduling*, 2010.

[37] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 639–646. IEEE, 2014.

[38] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling. Ffrob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research*, 37(1):104–136, 2018.

[39] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018.

[40] H. Van Seijen, S. Whiteson, and L. Kester. Efficient abstraction selection in reinforcement learning. *Computational Intelligence*, 30(4):657–699, 2014.

[41] G. Konidaris and A. Barto. Sensorimotor abstraction selection for efficient, autonomous robot skill acquisition. In *2008 7th IEEE International Conference on Development and Learning*, pages 151–156. IEEE, 2008.

[42] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[43] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[44] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese. Gibson env: Real-world perception for embodied agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9068–9079, 2018.

[45] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

Appendix A Environment Details

Interactive ToyEnv: We use $k = 11$ for our grid size. The room configuration is depicted in Fig. 2. The door state has a minimum value of 1 (closed) and a maximum value of 5 (open). The agent will increment the door state by 1 if it outputs `slide-up` at the front of the door while facing it and decrease by 1 if it outputs `slide-down`. The maximum episode length is 500.

The full observation includes:

- agent position
- agent orientation represented by yaw
- door state
- cosine and sine of yaw
- goal position
- whether the agent is next to the door
- a top-down global map with four channels: the first channel encodes the static environment, the second channel encodes agent's position and orientation, the third channel encodes the goal position and the last channel encodes the door position and state.

The high-level policy only sets subgoals for the first three elements of the observation because the other elements are either immutable or redundant information that solely helps the neural networks to learn policies more efficiently.

The reward weights defined in Sec. 5.2 are $w_{success} = 10.0$ and $w_{energy} = -0.001$.

Interactive Gibson Environment: We used a room size of approximately $6\,\mathrm{m} \times 9\,\mathrm{m}$. The room configuration is depicted in Fig. 2. The door is considered fully open if the door angle is more than $90°$. The maximum episode length is 1000.

The full observation includes:

- end-effector position $(x, y, z)_{ee}$
- base position
- linear and angular velocity
- roll, pitch, yaw of the base
- joint space configuration, velocities and torques for wheels and arm joints
- door angle and location
- goal location
- a flag that indicates collision or self-collision
- a flat that indicates if the end-effector is currently grabbing to the door handle
- cosine and sine of the robot's base orientation, joint configurations for revolute joints, and door angle
- a 64×64 depth map clipped to a range of $5\,\mathrm{m}$

For similar reasons as in the `Interactive ToyEnv`, the high-level policy only sets subgoals for the first element of the observation vector, the end-effector position $(x, y, z)_{ee}$. When the high level selects to use only the base embodiment ($e_{t'}$ =`base-only`), the z coordinate is *masked out* for the computation of intrinsic reward by the associated subgoal mask and the desired velocities for all joints of the arm are set to zero by the associated action mask. The reward weights defined in Sec.5.2 are included in Table 2a. The physics simulation parameters are included in Table 2b.

Weight	Value
$w_{success}$	30.0
$w_{progress}$	2.0
w_{energy}	-0.001
$w_{collision}$	-0.01

(a) Reward weights

Parameter	Value
Simulation time step	$1/40\,\mathrm{s}$
Action timestep:	$1/10\,\mathrm{s}$
Lateral friction coefficient	1.0
Rolling friction coefficient	0.0

(b) Physics simulation parameters

Table 2: Parameters for `Interactive Gibson Environment`

Appendix B Method Details

B.1 HRL4IN Algorithm

A detailed pseudo-code of our HRL4IN solution is included below in Algorithm 1:

Algorithm 1: HRL4IN Algorithm

Environment: `env`
Output : π_{HL}, π_{LL}
Parameters : $n_{update}, n_{step}, t_{sg}$

$n_{subgoal} \leftarrow 0$
$R_{extrinsic} \leftarrow 0$
for $update \leftarrow 0$ **to** n_{update} **do**
  for $step \leftarrow 0$ **to** n_{step} **do**
    if $n_{subgoal} = 0$ **then**
      $g^{LL}, e \leftarrow \pi_{HL}(obs)$ `// high level gives subgoal and embodiment type`
      $m^{act}, m^{sg} \leftarrow get\_masks(e)$
        `// determine action and subgoal mask from embodiment type`
      $g^{LL}_{cached} \leftarrow g^{LL}$
      $obs_{cached} \leftarrow obs$
      $g^{LL}_{abs} \leftarrow g^{LL} + obs$ `// convert subgoal from relative to absolute`
    $a_{LL} \leftarrow \pi_{LL}(obs, g^{LL}, m^{act}, m^{sg})$
    $a \leftarrow a_{LL} \otimes m^{act}$ `// apply action mask`
    $next\_obs, r \leftarrow$ `env.step`(a) `// collect experience from environment`
    $n_{subgoal} \leftarrow n_{subgoal} + 1$
    $R_{extrinsic} \leftarrow R_{extrinsic} + r$ `// accumulate extrinsic reward`
    if $||next\_obs - g^{LL}_{abs}||_2 < t_{sg}$ *or* $n_{subgoal} = T$ `// subgoal achieved or time out`
    then
      $ReplayBuffer_{HL} \leftarrow (obs_{cached}, g^{LL}_{cached}, R_{extrinsic}, next\_obs)$
      $n_{subgoal} \leftarrow 0$
      $R_{extrinsic} \leftarrow 0$
    $R_{intrinsic} \leftarrow ||(obs - g^{LL}_{abs}) \otimes m^{sg}||_2 - ||(next\_obs - g^{LL}_{abs}) \otimes m^{sg}||_2$
      `// calculate intrinsic reward`
    $ReplayBuffer_{LL} \leftarrow (obs, a_{LL}, R_{intrinsic}, next\_obs)$
    $g^{LL} \leftarrow g^{LL}_{abs} - next\_obs$ `// convert subgoal from absolute to relative`
    $n_{subgoal} \leftarrow n_{subgoal} + 1$
    $obs \leftarrow next\_obs$
  end
  $ReplayBuffer_{HL}$`.compute_advantage_estimates()`
    `// subroutine as defined in [43]`
  $ReplayBuffer_{LL}$`.compute_advantage_estimates()`
  π_{HL}`.update(`$ReplayBuffer_{HL}$`)` `// subroutine as defined in [43]`
  π_{LL}`.update(`$ReplayBuffer_{LL}$`)`
  $ReplayBuffer_{HL}$`.clear()`
  $ReplayBuffer_{LL}$`.clear()`
end

B.2 Network structure

We implement the high-level and low-level policies as Gated Recurrent Neural Networks (GRU [45]). For feature extraction, the observation space that contains spatial information (e.g. global map in `Interactive ToyEnv` and depth map in `Interactive Gibson Environment`) are processed by three convolutional layers and one fully connected layer with ReLU activation after flattening. Other observations are concatenated together and processed by one fully connected layer with ReLU activation. Finally, the features from two branches are concatenated together and fed into the GRU. The output features from the GRU are used to estimate the value function and

the action distribution. Note that the embodiment selection component of the high-level action is modeled as a discrete action (e.g. choose between base-only and base-arm). The action losses of the subgoal and the embodiment selector are aggregated together for back-propagation. We follow the same training procedure as PPO [43] and the training hyper-parameters can be found below.

B.3 Training Details and Hyperparameters

Table 3 summarizes the hyperparameters for HRL4IN.

Hyperparameter	Value
Learning Rate for High-Level Policy	0.00001
Learning Rate for Low-Level Policy	0.0001
Freeze High-Level Policy for First N Updates	500
Intrinsic Reward Scaling	30.0
Discount Factor for the High-Level policy	0.99
Discount Factor for the Low-Level policy	0.99
Subgoal Init. Std. Dev. (Gibson Env)	$[0.8\,\mathrm{m}, 0.8\,\mathrm{m}, 0.1\,\mathrm{m}]$
Subgoal Min. Std. Dev. (Gibson Env)	$[0.05\,\mathrm{m}, 0.05\,\mathrm{m}, 0.05\,\mathrm{m}]$
Subgoal Init. Std. Dev. (Toy Env)	$[0.2, 0.2, 0.5, 0.5]$
Subgoal Min. Std. Dev. (Toy Env)	$[0.1, 0.1, 0.25, 0.25]$

Table 3: Hyperparameters for HRL4IN

Table 4 summarizes the hyperparameters for PPO.

Hyperparameter	Value
Learning Rate	0.0001
PPO Clip Parameter	0.2
PPO Epochs	4
Num of Mini-Batches for PPO	8
Value Loss Coefficient	0.5
Entropy Coefficient	0.01
Max Gradient Norm	0.5
Rollout Steps	256
Hidden layer size for the GRUs	512
Use Linear LR Decay	True
Use Generalized Advantage Estimation (GAE)	True
Tau for GAE	0.95
Primitive Action Init. Std. Dev.	0.33
Primitive Action Min. Std. Dev.	0.1

Table 4: Hyperparameters for PPO

We normalize the observation space and the action space to $[-1, 1]$ during training and inference.

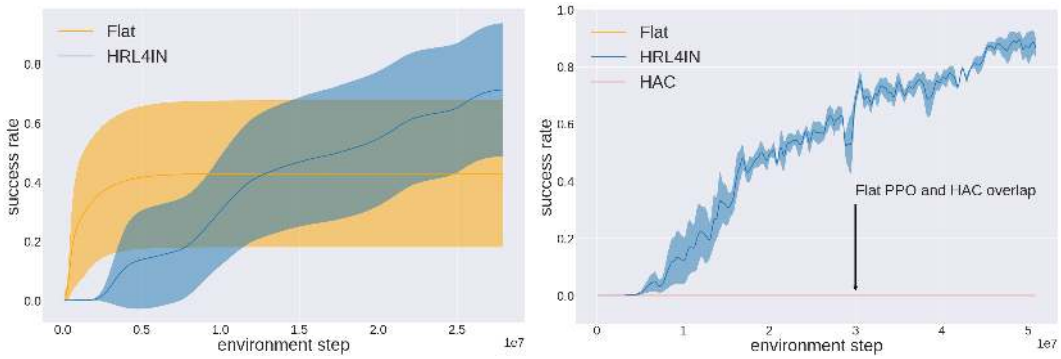
Appendix C Additional Results

In this section we show some additional results from the experiments described in Sec 5.

Fig. 7 shows additional training curves of success rate on `Interactive ToyEnv` and `Interactive Gibson Environment`, which complements Fig. 3a and Fig. 5a. It shows that our approach is able to achieve higher final success rate against both baselines in both environments.

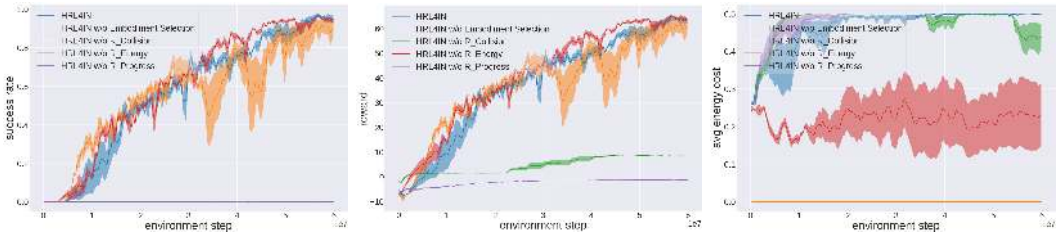
Fig. 8 depicts the training curves of our ablation studies, which complements Table 1. We observe that $r_{progress}$ and $r_{collision}$ are required for the agent to solve the task, and r_{energy} and embodiment selection are required for the agent to learn embodiment selection and solve the task efficiently.

Fig. 9 shows additional visualizations of subgoals set by the high-level policy of HRL4IN in `Interactive Gibson Environment`. These visualizations illustrate that the high level policy of HRL4IN learns to command meaningful subgoals to guide the low-level policy to achieve different types of subtasks, including pure navigation, reaching for the door handle and pulling the door open.



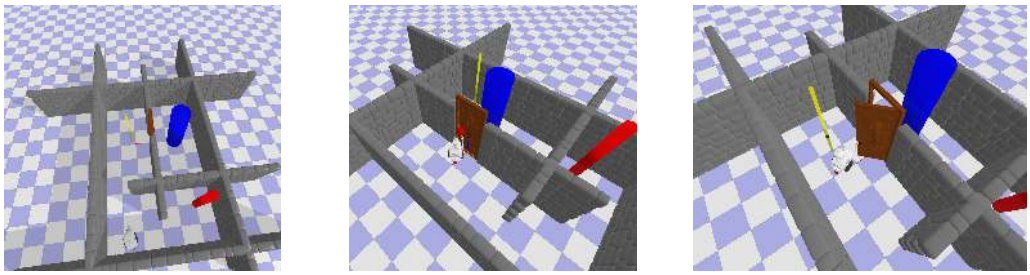
(a) (b)

Figure 7: (a) Success rate over time for HRL4IN and flat PPO in `Interactive ToyEnv`, HRL4IN achieves higher success rate despite requiring more samples during the initial phase of training; (b) Success rate over time for HRL4IN, flat PPO and HAC in `Interactive Gibson Environment`; HRL4IN achieves close to 100% success rate while the two baselines fail to solve the task.



(a) (b) (c)

Figure 8: Success rate, reward and energy-saving over time for HRL4IN and its ablated versions in `Interactive Gibson Environment`. $r_{progress}$ and $r_{collision}$ are essential for task performance; r_{energy} and embodiment selection are essential for energy efficiency.



(a) (b) (c)

Figure 9: More examples of subgoals set by HRL4IN in `Interactive Gibson Environment`: (a) a pure navigation (base-only) subgoal is set to guide the robot to move closer to the door; (b) a base+arm subgoal is set to guide the robot to touch the door handle; Although the subgoal is not set directly on the door handle, when the low-level policy tries to achieve it by stretching its arm, the end-effector grabs the door handle; (c) a base+arm subgoal is set behind the robot so that it can pull the door handle back and open the door.