Structured Exploration Through Instruction Enhancement for Object Navigation

Matthias Hutsebaut-Buysse[0000−0001−6091−294X],
Tom De Schepper[0000−0002−2969−3133],
Kevin Mets[0000−0002−4812−4841], and
Steven Latré[0000−0003−0351−1714]

University of Antwerp - imec
IDLab - Department of Computer Science
{firstname.lastname}@uantwerpen.be

Abstract. Finding an object of a specific class in an unseen environment remains an unsolved navigation problem. Hence, we propose a hierarchical learning-based method for object navigation. The top-level is capable of high-level planning, and building a memory on a *floorplan*-level (e.g., which room makes the most sense for the agent to visit next, where has the agent already been?). While the lower-level is tasked with efficiently navigating between rooms and looking for objects in them. Instructions can be provided to the agent using a simple synthetic language. The top-level intelligently enhances the instructions in order to make the overall task more tractable. Language grounding, mapping instructions to visual observations, is performed by utilizing an additional separate supervised trained goal assessment module. We demonstrate the effectiveness of our method on a dynamic configurable domestic environment.

Keywords: Hierarchical Reinforcement Learning · Object Navigation · Embodied AI

1 Introduction

Finding objects in unseen environments is a hard navigation task. In order to be successful, an agent needs to be capable of mastering a number of skills. First, the agent needs to be capable to explore the environment in a structured manner: it should figure out the layout of the previously unseen environment, keep a memory of past actions, and remember visited regions. Second, the agent needs to be capable to understand the instruction: map an instruction to an actual visual representation. Third, the agent needs to be capable to make decisions on multiple abstraction levels: navigate to the other side of the building versus navigating through a doorway.

These problems have been studied individually intensively in various settings [19, 20, 17, 4]. However, constructing an agent capable of simultaneously performing these feats, remains an open challenge. In this paper we study how we can build an agent capable of simultaneously handling long-term planning through abstraction, low-level locomotion and basic language grounding.

Current navigation solutions typically utilize a *sense-plan-act* approach, in which different modules interact with each other. These solutions however tend to be brittle, are prone to error propagation, and often require a lot of manual engineering [12, 13]. End-to-end Reinforcement Learning (RL) systems have recently been proposed, as an alternative learning through interactions based solution, to handle these issues [21]. Unfortunately, as we will demonstrate, RL agents are often unable to reason on multiple levels of abstraction, have difficulties with mapping language instructions, and often explore poorly.

In contrast, our approach allows the agent to plan and explore on multiple levels of abstraction (e.g., on room-level and actuator-level) through utilizing a hierarchical approach. The proposed agent can be trained using only the reward-signal received from the environment, and only requires an egocentric RGB observation. This is in contrast to prior approaches, which often also require the pose of the agent as input.

In order to communicate between the two layers we propose *instruction enhancements*. In this system, the top-level is allowed to enhance the instruction it received from the environment. For example, if the original instruction is: *"Find the red ball"*, the top-level might choose to enhance this instruction to: *"Find the red ball, in the kitchen"*. This allows the top-level to plan on a higher level of abstraction (Which room makes sense to visit next? Where have I already been?). In turn, the enhanced instruction makes the task more tractable to complete by the lower-level.

Because both traditional and learning-based approaches are still unsolved, we take one step back from the typically used photo-realistic simulators [19, 17], and utilize a visually simpler setting [3], while keeping most of the navigation and generalization complexities. In this setting we demonstrate why a flat, non-hierarchical RL agent, does not manage to make any progress, and how our hierarchical approach is capable of exploring the environment in a more principled way. We also demonstrate the generalization capabilities of the agent to find previously unseen objects in new unseen environment configurations.

The contributions of this work are three-fold: (1) We introduce a dual layer hierarchical approach, capable of simultaneously learning structured room-level exploration, and low-level navigation. (2) In order to communicate between layers we propose to enhance instructions, allowing loose coupling of layers and generalization to novel instructions. (3) The introduction of a goal assessment module, which is capable of addressing whether the current state satisfies the instruction, and thus allows offloading language grounding, and integration of prior knowledge in a learning-based setup.

2 Background

Reinforcement Learning (RL)
A sequential decision-making problem can be modelled as a Partially Observable Markov Decision Process (POMDP), represented by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \Omega, \mathcal{O}, \gamma \rangle$.

On each time step t, the agent samples an action $a_t \in \mathcal{A}$ from its policy $\pi(a_t|o_t, g_t)$, and the environment produces in turn an observation $o_t \in \Omega, o_t \sim \mathcal{O}(s_t)$ of the internal state $s_t \in \mathcal{S}$ according to an unknown transition function $\mathcal{P}(s_{t+1}|s_t, a_t)$. The agent has access to a reward signal $r_t(s_t, a_t, g_t)$, which can be utilized to learn the value of the sequence of previously taken actions. In the goal-conditional RL setting studied in this paper, the reward-signal depends on an additional goal-signal g_t (the instruction). This goal-signal remains constant during each task instance (an episode). Episodes are terminated after a predetermined step-limit is reached, or the agent utilizes a special *done*-action.

The goal of RL consists of finding a policy π capable of maximizing the sum of rewards R_t, discounted by a factor $\gamma \in [0, 1]$, through environment interactions:

$$R_t = \mathop{\mathbb{E}}_{\pi, \mathcal{P}} \left[\sum_{t=0}^{T} \gamma^t r_t \left(o_t, g_t, a_t, o_{t+1} \right) \right] \tag{1}$$

Proximal Policy Optimization (PPO)
In order to learn a policy the on-policy Proximal Policy Optimization (PPO) algorithm [18] can be utilized. PPO utilizes an importance-weighted advantage on samples collected in the environment during a rollout phase. A proximity clipping term is used as a trust region optimization method in order to allow updates to use experiences collected during a rollout multiple times. This is done in order to improve sample efficiency.

Hierarchical Reinforcement Learning (HRL)
Exploration within policy-gradient methods such as PPO is achieved through sampling actions from a stochastic policy. However, solely depending on this mechanism to find solutions for complex tasks is often not tractable [15].

Within a two-level goal-conditioned hierarchical approach, a meta-controller $\pi_m(z_t|o_t, g_t)$ maximizes the extrinsic reward signal r_t indirectly by generating high-level actions $z_t \in \mathcal{Z}$ (often called skills or sub-behaviors). These high-level actions are executed for c steps by a second low-level policy $\pi_c(a_t|o_t, z_t)$, often called a controller. The controller maximizes an intrinsic reward signal by directly outputting primitive actions $a_t \in \mathcal{A}$.

3 Approach

The proposed Structured Exploration Through Instruction Enhancement (SETIE) approach consists of three parts: (a) the meta-controller $\pi_m(z_t|o_t, g_t)$ which performs high-level planning, by working on a lower temporal resolution, (b) the

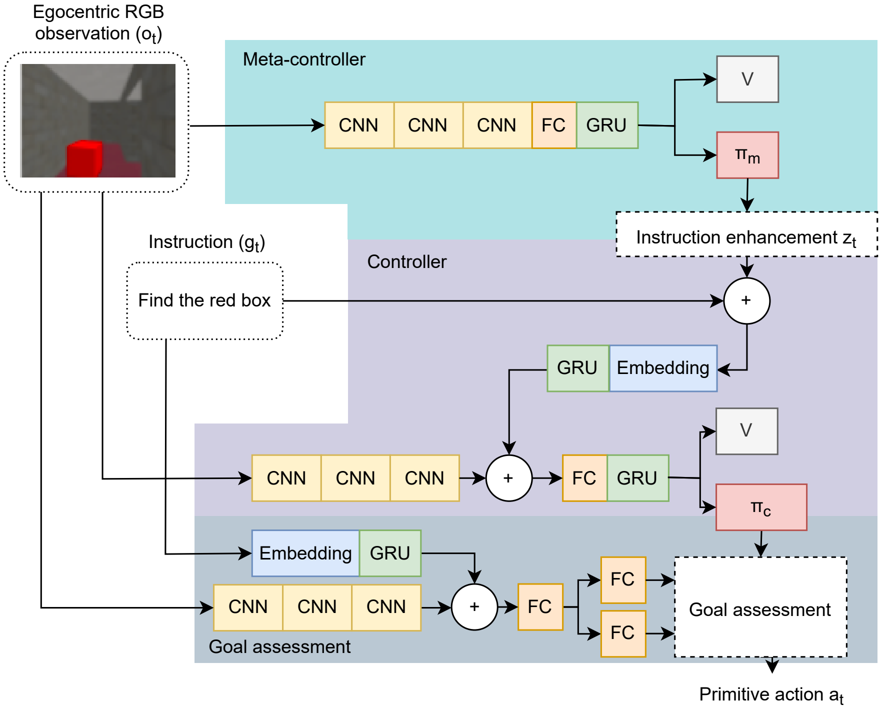

Fig. 1: SETIE architecture: the meta-controller handles structured exploration between different rooms from egocentric observations by enhancing the instruction. This output is used by the controller, in order to return primitive actions (navigation). The goal assessment module is used for language grounding.

controller $\pi_c(a_t|o_t, g_t, z_t)$ which handles low-level navigation, and (c) the goal assessment module $G(o_t, g_t) \rightarrow \{1, 0\}$ which handles language grounding. A visual representation of the architecture is displayed in Figure 1.

3.1 Meta-controller

The meta-controller $\pi_m(z_t|o_t, g_t)$ is responsible for learning high-level navigation of the environment solely from partial state observations (through an egocentric RGB camera). This task consists of two sub-tasks: (1) discovering the layout of the current environment, determining which rooms are connected to which other rooms. Commonsense reasoning (the garage is less likely to be connected with the bathroom) together with a trial-and-error approach can be used in order to solve this task. (2) Keeping an implicit memory of which rooms have already been visited in order to explore the environment in a structured manner. Because the meta-controller reasons on a higher level of abstraction, the agent is capable to perform these tasks using a generic Gated Recurrent Unit (GRU) component [5] in its architecture.

The action-space of the meta-controller consists of a discrete set of *instruction enhancements.* This set of enhancements is provided up-front to the agent. Instruction enhancements should be defined on a higher level of abstraction, than the primitive actions utilized by the controller. By introducing this additional level of abstraction, the agent is able to explore in a structured manner (e.g., room by room).

The meta-controller does not interact with the environment itself, but can only influence the behavior of the controller through enhancing the instruction. For example the extrinsic instruction g_t could have been *"Find the green key"*, which the meta-controller can enhance to become *"Find the green key, in the dining room"*.

Within HRL, designing a sub-behavior space $\mathcal{Z}$ is a complex challenge. Most often this space is tightly coupled between the different levels. Utilizing language allows to decouple multiple levels. This allows the controller and meta-controller to be trained independently. Furthermore, language has also the potential to generalize to unseen instructions [11], and can make the intention of the agent clear to a human in the loop [2].

The meta-controller acts on a lower temporal resolution and is asked to provide a new instruction enhancement every c timesteps.

As the meta-controller has no direct influence on the environment, but only can act through the controller, its training needs to take into account potential unexpected behavior of a trained controller. Such quirks might be over-exploration of some rooms, while quickly moving through others. Accounting for these eccentricities can be done by utilizing a fully trained and frozen controller during training of the meta-controller. In this setting the meta-controller observes the environment, selects an instruction enhancement, and waits until the controller has taken c-steps, before sampling a novel enhancement. The reward of the meta-controller consists of the discounted sum of the extrinsic reward collected during the usage of the active instruction enhancement:

$$R_t(s_t) = 1/c \sum_{t=0}^{c} \gamma^t r_t \left(o_t, g_t, a_t, o_{t+1}\right) \tag{2}$$

A second option to train the meta-controller consists of assuming a perfectly behaving controller. In this setting the (simulated) environment will carry out the enhancements, and move the agent to different rooms, while respecting the floor plan. Utilizing this second approach allows both controller and meta-controller to be trained in parallel (as there is no dependency). In order to utilize this second training scheme a different reward function is required. For example, a reward function based on the room coverage can be utilized. In this setting each instruction enhancement which takes the agent to a previously unvisited room will lead to a positive reward (0.1), while other proposed enhancements will result in a negative slack penalty (-0.01).

While in the empirical evaluation of the presented method instruction enhancements consists of rooms to navigate between, other sets of enhancements can be used in different settings.

3.2 Low-level Controller

The controller $\pi_c(a_t|o_t, g_t, z_t)$ interacts with the environment through its primitive actions $a_t \in \mathcal{A}$. The controller expects on each timestep an egocentric RGB observation of the environment $o_t \in \Omega$ together with a task instruction $g_t \in \mathcal{G}$ and an instruction enhancement $z_t \in \mathcal{Z}$ provided by the meta-controller. The instruction informs the agent of its objective (e.g., *find the red ball*), and the instruction enhancement (e.g., *in the kitchen*) adds additional information on how the instruction should be carried out. The instruction enhancement will essentially navigate the agent to different rooms, resulting in episodic exploration of the different rooms in order to solve the main instruction. Both instruction and enhancement are provided using simple language sentences.

The action-space $\mathcal{A}$ of the controller consists of a discrete set of primitive movement steps (*move forward, turn left, turn right*) and a special *query*-action. This special query-action is invoked when the agent perceives itself near the goal object. Utilizing this action will invoke the goal assessment module.

Due to the utilization of instruction enhancements, the controller can be trained independent of the meta-controller. A straightforward way of training the controller, is to enhance the instructions by utilizing an oracle. When this oracle provides the most useful enhancement (e.g., which room should the agent visit next to find the goal) the extrinsic reward signal can be utilized to reward the agent. For example in the setting of object navigation, controllers can be rewarded by utilizing the improvement in geodesic distance between the agent and the goal object.

3.3 Goal Assessment Module

To signal that the agent thinks it has completed the objective, it needs to use a special *done*-action. Utilizing this action will typically end the episode. However, as we will empirically demonstrate in Section 4.2, incorrect usage of this action is one of the main failure modes appearing prior to the introduction of a goal assessment module. In contrast, when the *done*-action does not terminate the episode, the agent trains considerable faster.

In order to integrate the goal assessment module, the done-action is removed from the action-space of the controller. Instead, a *query*-action is added to this action space. This novel query action will not terminate the episode (soft termination), but will query the goal assessment module. If the goal assessment module deems that the instruction is satisfied, and the agent is close enough to the target object, the agent will utilize the original done-action.

Essentially, the controller is now able to focus on low-level navigation, and consult an expert (the goal assessment module) in order to handle language grounding of the instruction.

In order to allow the agent to find objects it did not see during training, a novel goal assessment model can be trained independent of the controller and meta-controller. Which is useful, as training a controller and meta-controller is typically more computational expensive.

In order to collect training data for the goal assessment module a random policy can be used, collecting both examples with goal objects, and observations without any visible objects. For positive samples the correct positive class is utilized 50% of the time, while in the remainder cases another random possible instruction is utilized, together with the negative class label. This allows balancing out positive and negative labels.

4 Empirical Evaluation

4.1 Environment Description

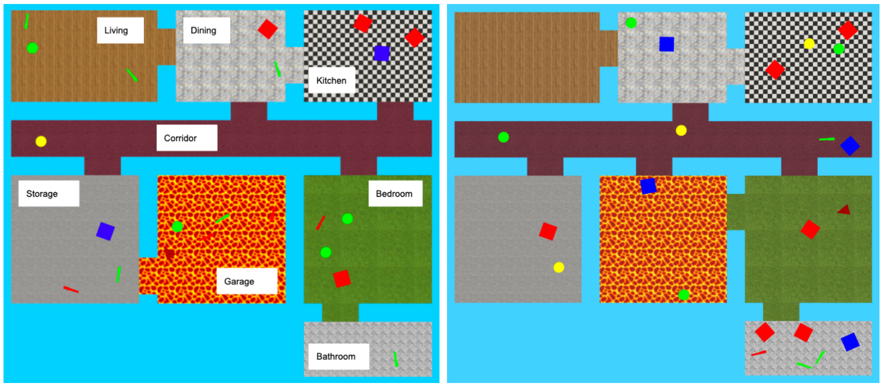

Fig. 2: Two different instances of the evaluation environment. Connections between rooms are randomized (with a holdout set of configurations). The agent has no access to this top-down map view.

In order to demonstrate the effectiveness of SETIE, a simulated domestic environment is utilized within the *MiniWorld* framework [3]. Two instances are represented in Figure 2. The environment consists of 7 different rooms (garage, storage, bedroom, bathroom, living room, dining room and kitchen) together with a corridor that connects some of these rooms (depending on the instance). Each room has a distinctive look. As not all rooms are connected, the agent will often need to backtrack to previously visited points in order to further explore the environment.

Throughout the environment different abstract objects are randomly placed. Objects are defined by a category and a color. The categories used are *box*, *ball* and *key*. In the experiments there is typically one goal object and multiple *distractor* objects. In each task instance there is only a single object which matches the goal object description. The task is communicated using language through the template of *"Find the [color] [shape]"*. The following objects are used during training: *red box, green ball, blue box, yellow ball, red key and green key.* There is no association between objects and rooms.

On each timestep the agent observes an egocentric RGB observation o_t of the environment. The reward function is densely defined, and consists of the improvement in the geodesic distance between the agent and the goal object. We use a slack penalty of 0.01 which is subtracted from the reward on each timestep. When reaching the goal object we award the agent with a success bonus of 10.

$$r_t(s_t, a_t, g_t) = (-\Delta_{geo\_dist} - 0.01) + 10 * \mathbb{1}_{success} \tag{3}$$

Regarding actions, the agent is capable of turning left and right for a fixed amount, moving a fixed distance forward, and utilizing a special *done*-action. In order to successfully complete an episode, the agent needs to use this done-action close to the goal object.

In each episode, the agent starts in a random position, and has no access to its current pose, the name of the room it is in, or a map of the environment. The connections between the different rooms are randomly enabled. However, each room is always accessible, and there are no uncommon connections (e.g., bathroom connected to kitchen). In total this results in 132 different possible floor plans. A holdout set of 30 floor plans is not utilized during training, but kept solely for evaluation purposes. This holdout set can be used in order to assess the generalization capabilities of the agent regarding floor plans.

4.2 Baselines: why do non-hierarchical approaches fail?

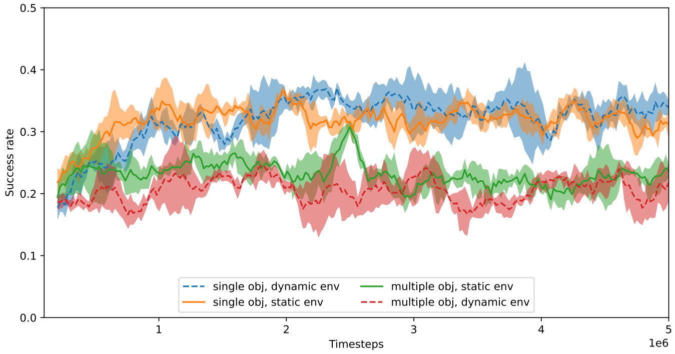

Fig. 3: Training performance of a non-hierarchical PPO agent with soft-termination. Results are averaged over 3 runs.

With soft-termination When utilizing a non-hierarchical PPO agent without any instruction enhancements, and with only a single object (a red box or blue box) the agent is capable of achieving an average success rate of $\sim 35\%$ after 5 million interactions with the environment (Figure 3). When also introducing the problem of language grounding, by adding multiple objects to the environment, the agent has an average success rate of $\sim 20\%$ after 5 million interactions.

No soft-termination (full problem setting) If we also remove the relaxation of soft termination of the environment we arrive at the full problem setting. In this setting, when the agent utilizes the *done*-action incorrectly, the episode is terminated. We analyzed the failure modes of the baseline agent in this setting (Figure 4):

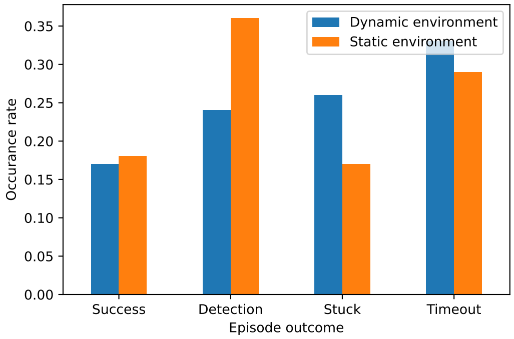

Fig. 4: Failure modes of the trained non-hierarchical baseline. If the floor plan remains fixed (static environment), the amount of episodes where the agent gets stuck decreases, however this in turn increases goal detection errors.

- **Detection:** agent used *done*-action but was in the wrong position.
- **Timeout:** agent did not manage to find the goal within the allowed amount of timesteps, the agent did not use the *done*-action at all.
- **Stuck:** distance between agent and goal object did not change in the final 10 steps.

When looking at these failure modes we noticed that the main reason for failure in a static environment setting, is related to the detection of goal objects. When also making the environment dynamic, both local navigation problems (getting stuck), and planning problems (timeout) start to occur more frequently.

4.3 Does enhancing the instruction make the task more tractable?

From the previous section, we can conclude that a non-hierarchical agent is not able to reliably solve the studied task. In order to validate whether enhanc-

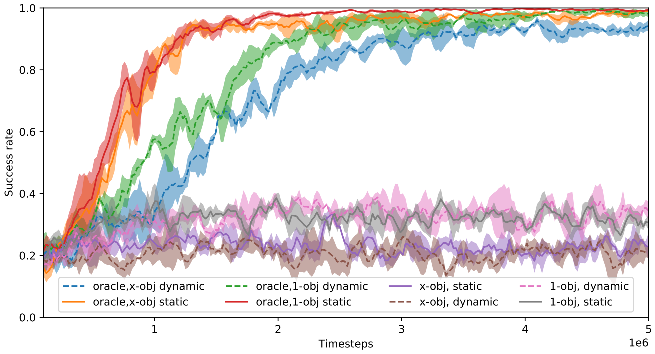

Fig. 5: Training performance of the controller, in this setting the agent is allowed to use the done-action multiple times (soft termination). Without information to which room the agent should move next (oracle), the agent is unable to learn a policy in the environment. Results are averaged over 3 runs.

ing the instruction will improve the performance, we trained an agent with its instructions enhanced through the use of an oracle.

The utilized oracle is aware of the shortest path to the goal object in terms of rooms to visit. Having access to such an oracle outside the training environment, is an unrealistic assumption. The learned meta-controller will however take over the role of this oracle, providing adequate enhancements.

Utilizing an oracle based on the shortest path also alleviates the requirement of a custom reward function. If the controller is able to correctly interpret and follow the instruction enhancement, it will also collect the most reward.

As the results plotted in Figure 5 indicate, enhancing the instructions allows the agent to almost entirely consistently solve the task both in the setting with a single object (1-obj) and multiple objects (x-obj). This validates the idea that enhancing the instruction allows the controller to carry out the low-level control task. In order to solve the entire task there is still the need to remove soft termination (Section 4.4), and actually train a meta-controller (Section 4.5).

4.4 What is the impact of soft termination?

In the previous experiments, the controller was trained using soft termination. This means that the agent is allowed to use the *done*-action multiple times in an episode. Normally, this would terminate the episode, however we found that allowing the agent to utilize this action multiple times during training significantly increased the sample efficiency and success rate (Figure 6). This training mechanism is especially crucial in the settings which require language grounding (multiple objects). We can allow this constraint due to the goal assessment module, which will filter out invalid done-actions when utilizing the entire architecture.

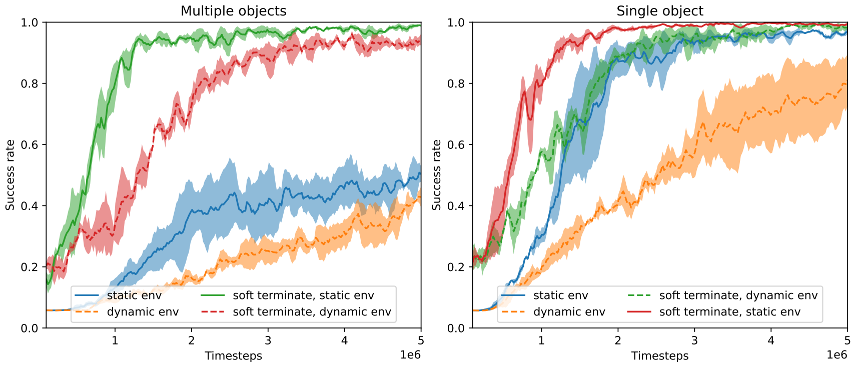

Fig. 6: Training performance of the controller, with oracle instruction enhancements. Allowing soft termination, greatly improves sample efficiency. Results are averaged over 3 runs.

4.5 Does a trained controller allow the meta-controller to solve the task?

In Figure 7 the results from training a meta-controller (through enhancing the instructions of a trained controller) in various configurations are plotted. The meta-controller has no problem exploring the environment when there is only a single static environment configuration used with a single goal object placed in it (SR ∼ 95%). When multiple objects are present in the static environment

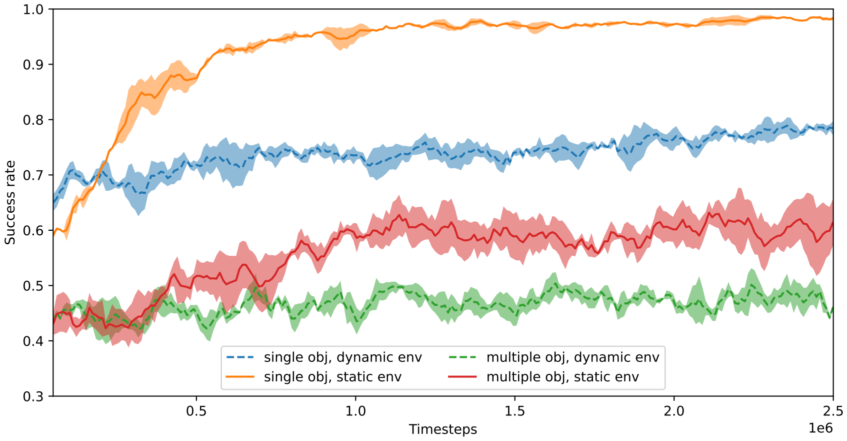

Fig. 7: Meta-controller training performance. Results are averaged over 3 runs.

setting, performance receives a significant hit (SR $\sim$ 60%), but the agent is still able to improve its performance.

When the agent needs to manage dynamic instances of environments it starts with a high success rate, and is able to steadily improve (SR $\sim$ 70%) in the setting with a single goal object. However, in the setting with both a dynamic environment configuration, and multiple objects the agent is not able to improve its initial performance (SR $\sim$ 45%).

4.6 Is the agent capable of exploring in a structured way?

The failure modes of the baseline agent indicated that a lot of episodes ($\sim$ 30%) failed due to the agent running out of allowed steps. This might indicate that the baseline agent is not able to explore the environment in a structured manner. In Table 1 we compare the percentage of the rooms the agent visited. From the results plotted in this table, we can conclude that the hierarchical approach is capable of covering a significantly larger proportion of the environment on average.

Agent	**Objects**	**Environment**	**Room coverage**
Hierarchical	Single	Static	51.0%
		Dynamic (holdout)	45.4%
		Dynamic (train)	45.5%
	Multiple	Static	50.2%
		Dynamic (holdout)	36.4%
		Dynamic (train)	36.7%
Flat (baseline)	Single	Static	27.8%
		Dynamic (holdout)	25.8%
		Dynamic (train)	26.3%
	Multiple	Static	12.5%
		Dynamic (holdout)	12.6%
		Dynamic (train)	12.6%

Table 1: Average room coverage observed during evaluation runs.

4.7 How well does the proposed hierarchical architecture performs?

In this section the performance of the architecture is analyzed in its entirety. We are especially interested in how well the agent is capable of handling unseen environment floor plans, and novel objects.

Zero-shot transfer to unseen environment configurations The agent is allowed to utilize 102 different floor plans during training. In order to validate whether the agent is capable of functioning in an environment it did not see during training, there is also a test-set containing 30 floor plans the agent did not see during training.

Architecture	Objects	Static	Train	Test
Flat PPO baseline	Single	37% ± 3.71	42% ± 4.79	44% ± 3.76
Hierarchical + GA	Single	81% ± 5.20	76% ± 5.54	75% ± 4.67
Flat PPO baseline	Multiple	13% ± 3.41	15% ± 4.47	12% ± 2.66
Hierarchical + soft term.	Multiple	82% ± 6.44	69% ± 5.76	67% ± 5.27
Hierarchical	Multiple	15% ± 3.12	18% ± 2.81	15% ± 3.13
Hierarchical + GA	Multiple	52% ± 3.06	38% ± 4.58	40% ± 4.52

Table 2: Overall performance of the entire architecture. For each setting 10 runs of each 100 random episodes where used.

From the results plotted in Table 2 we can conclude that the hierarchical approach has a high success rate in the static environment. Especially, when there is no language grounding required.

In the setting with multiple objects, the hierarchical agent is now able to reach a high success rate when soft termination is allowed. When soft termination is disabled, the goal assessment module is capable of somewhat emulating this improved performance. However, there still remains room for improvement. When qualitatively looking at the mistakes made by the goal assessment module, we noticed that it often made mistakes if the goal object was barely visible in the single passed RGB observation.

In all cases, the agent was successfully capable of achieving a similar level of performance in the floor plan holdout set as in the training set.

Zero-shot transfer to unseen goal objects Because the instructions are formulated in natural language, we have an interface that makes it straightforward to test how well the agent handles combinations of colors and objects it did not see during training. The goal assessment module was retrained in order to be capable to detect the novel combinations of colors and shapes, while keeping all original navigation policies (controller and meta-controller).

Similar to the zero-shot environment transfer experiments, we empirically can validate from the results in Table 3 that the agent is able to successfully find combinations of colors and shapes the agent did not see before without having to re-train the controller and meta-controller.

Environment:	**Static**	**Train**	**Test**
Flat PPO Baseline	15% ± 1.69	15% ± 3.1	14% ± 4.21
Hierarchical	17% ± 2.54	15% ± 2.96	14% ± 3.52
Hierarchical + soft term.	78% ± 3.75	69% ± 2.75	66% ± 3.77
Hierarchical + GA	52% ± 4.36	39% ± 3.7	38% ± 4.58

Table 3: Overall performance of the entire architecture on a holdout set of goal objects. For each setting 10 runs of each 100 random episodes where used.

5 Related work

Object Navigation in RL
Prior proposed architectures either fully rely on end-to-end training [21], make use of self-supervised learning through auxiliar tasks [10, 22], or use a planning-style approach by inferring maps from observations [1, 6]. In contrast to our method, prior work relies on a pose sensor.

Structured exploration through HRL
HRL [9] is a core mechanism in object navigation. In these architectures a top-level meta-controller is trained to output relative goal position points which should be reachable by a trained PointGoal agent. For example the agent in [16] is capable of inferring a rough floor plan of the environment, the top-level outputs pointgoals in order to reach a desired area. This is similar to how we navigate the agent to different rooms in order to solve ObjectNav tasks.

Instead of using points as the interface between different levels of the architecture, natural language has also been proposed as the interface [11, 8], allowing the lower-level to generalize to unseen instructions.

Language grounding
The problem of language grounding has been approached solely from data [14], by adding auxiliary tasks and curriculum learning [7] and feature-wise affine transformation based on the instruction [4].

6 Conclusion

In this paper we study the problem of structured exploration in an object navigation setting. We demonstrate how the three sub-problems of: navigation, high level reasoning, and language grounding each contribute to the complexity of object navigation. A hierarchical approach is proposed in order to handle both the low-level navigation, and high-level planning. In order to have a loose coupling between the layers, language is used to enhance the original instruction in a way that makes it feasible for a low-level controller to partially tackle the overall task. To handle the third sub-problem of basic language grounding, a goal assessment module is introduced in order to guide the controller in assessing whether goal objects have been reached.

The effectiveness of the proposed architecture is empirically demonstrated in a simulated domestic environment. We demonstrate that the agent is able to better handle unseen environment configurations, and unseen goal objects compared to a non-hierarchical baseline.

In future work we plan on researching how we can further improve the performance in dynamic environments, make the set of instruction enhancements more dynamic, and how well the SETIE approach performs in real-world settings.

ACKNOWLEDGMENT

This research received funding from the Flemish Government under the “Onderzoeksprogramma Artificile Intelligentie (AI) Vlaanderen” programme.

References

1. Chaplot, D.S., Gandhi, D., Gupta, A., Salakhutdinov, R.: Object Goal Navigation using Goal-Oriented Semantic Exploration. In: Advances in Neural Information Processing Systems 33 (2020)
2. Chen, V., Gupta, A., Marino, K.: Ask Your Humans: Using Human Instructions to Improve Generalization in Reinforcement Learning. In: ICLR21 (2021)
3. Chevalier-Boisvert, M.: gym-miniworld environment for openai gym. https://github.com/maximecb/gym-miniworld (2018)
4. Chevalier-Boisvert, M., Bahdanau, D., Lahlou, S., Willems, L., Saharia, C., Nguyen, T.H., Bengio, Y.: BabyAI: First Steps Towards Grounded Language Learning With a Human In the Loop. In: ICLR19 (2019)
5. Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In: EMNLP14 (2014)
6. Gupta, S., Tolani, V., Davidson, J., Levine, S., Sukthankar, R., Malik, J.: Cognitive Mapping and Planning for Visual Navigation. International Journal of Computer Vision **128**(5), 1311–1330 (2020)
7. Hermann, K.M., Hill, F., Green, S., Wang, F., Faulkner, R., Soyer, H., Szepesvari, D., Czarnecki, W.M., Jaderberg, M., Teplyashin, D., Wainwright, M., Apps, C., Hassabis, D., Blunsom, P.: Grounded Language Learning in a Simulated 3D World. arXiv:1706.06551 [cs, stat] (2017)
8. Hu, H., Yarats, D., Gong, Q., Tian, Y., Lewis, M.: Hierarchical Decision Making by Generating and Following Natural Language Instructions. In: NeurIPS19 (2019)
9. Hutsebaut-Buysse, M., Mets, K., Latré, S.: Hierarchical reinforcement learning: A survey and open research challenges. Machine Learning and Knowledge Extraction **4**(1), 172–221 (2022)
10. Jaderberg, M., Mnih, V., Czarnecki, W.M., Schaul, T., Leibo, J.Z., Silver, D., Kavukcuoglu, K.: Reinforcement Learning with Unsupervised Auxiliary Tasks. In: ICLR17 (2017)
11. Jiang, Y., Gu, S., Murphy, K., Finn, C.: Language as an Abstraction for Hierarchical Deep Reinforcement Learning. In: NeurIPS19 (2019)

12. Karkus, P., Cai, S., Hsu, D.: Differentiable slam-net: Learning particle slam for visual navigation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 2815–2825 (2021)
13. Mishkin, D., Dosovitskiy, A., Koltun, V.: Benchmarking Classic and Learned Navigation in Complex 3D Environments. arXiv:1901.10915 [cs] (2019), http://arxiv.org/abs/1901.10915
14. Misra, D., Langford, J., Artzi, Y.: Mapping Instructions and Visual Observations to Actions with Reinforcement Learning. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing (2017)
15. Nachum, O., Tang, H., Lu, X., Gu, S., Lee, H., Levine, S.: Why Does Hierarchy (Sometimes) Work So Well in Reinforcement Learning? In: NeurIPS 2019 DeepRL Workshop (2019), http://arxiv.org/abs/1909.10618
16. Narasimhan, M., Wijmans, E., Chen, X., Darrell, T., Batra, D., Parikh, D., Singh, A.: Seeing the Un-Scene: Learning Amodal Semantic Maps for Room Navigation. In: ECCV20 (2020)
17. Savva, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., Parikh, D., Batra, D.: Habitat: A Platform for Embodied AI Research. In: Proceedings of the IEEE International Conference on Computer Vision (2019)
18. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs] (2017), http://arxiv.org/abs/1707.06347
19. Szot, A., Clegg, A., Undersander, E., Wijmans, E., Zhao, Y., Turner, J., Maestre, N., Mukadam, M., Chaplot, D., Maksymets, O., Gokaslan, A., Vondrus, V., Dharur, S., Meier, F., Galuba, W., Chang, A., Kira, Z., Koltun, V., Malik, J., Savva, M., Batra, D.: Habitat 2.0: Training Home Assistants to Rearrange their Habitat. In: Advances in Neural Information Processing Systems. vol. 34, pp. 251–266 (2021)
20. Weihs, L., Salvador, J., Kotar, K., Jain, U., Zeng, K.H., Mottaghi, R., Kembhavi, A.: AllenAct: A Framework for Embodied AI Research. In: CoRR2020 (2020)
21. Wijmans, E., Kadian, A., Morcos, A., Lee, S., Essa, I., Parikh, D., Savva, M., Batra, D.: DD-PPO: Learning Near-Perfect PointGoal Navigators from 2.5 Billion Frames. In: ICLR20 (2020)
22. Ye, J., Batra, D., Wijmans, E., Das, A.: Auxiliary Tasks Speed Up Learning PointGoal Navigation. In: Proceedings of the 2020 Conference on Robot Learning (2020)