Machine Vision and Applications (2024) 35:6
https://doi.org/10.1007/s00138-023-01492-1

ORIGINAL PAPER

Learning to explore by reinforcement over high-level options

Juncheng Liu[1] · Brendan McCane[1] · Steven Mills[1]

Received: 26 August 2022 / Revised: 28 July 2023 / Accepted: 1 November 2023 / Published online: 30 November 2023

Abstract
Autonomous 3D environment exploration is a fundamental task for various applications such as navigation and object searching. The goal of exploration is to investigate a new environment and build a map efficiently. In this paper, we propose a new method which grants an agent two intertwined options of behaviors: "look-around" and "frontier navigation." This is implemented by an option-critic architecture and trained by reinforcement learning algorithms. In each time step, an agent produces an option and a corresponding action according to the policy. We also take advantage of macro-actions by incorporating classic path-planning techniques to increase training efficiency. We demonstrate the effectiveness of the proposed method on two publicly available 3D environment datasets, and the results show our method achieves higher coverage than competing techniques with better efficiency. We also show that our method can be transferred and applied on a rover robot in real-world environments.

Keywords Exploration · Option-critic · Reinforcement

1 Introduction

When a robot is placed in a new environment, it is very important that the surroundings are mapped as quickly as possible in an unsupervised manner so that subsequent tasks can be more easily completed. Specifically, an optimal policy should be able to give a sequence of actions that maximizes the coverage of an environment given a limited time or energy budget. This process is called autonomous exploration, and it has been studied for many years.

Traditional methods include frontier-based exploration [21] where agents consistently chase after frontiers until the whole scene is visited. Most existing learning-based work tackles this problem either by active SLAM (simultaneous localization and mapping) or reinforcement learning (RL). Traditional methods consider the problem as a partially observable Markov decision process (POMDP) [19], while reinforcement learning algorithms, which have attracted more attention in recent years, employ various kinds of rewards such as curiosity, coverage or novelty [13] to encourage the exploration of unknown places (Fig. 1).

Compared to traditional visual-SLAM methods, learning-based methods are able to leverage the structural regularities of environments with semantic features. However, one main drawback of reinforcement learning-based exploration is the inefficient use of training data and extremely long training times. This is partly because the action set of an agent is too low level to train by a limited number of training episodes in an end-to-end manner, though theoretically feasible. One solution is imitation learning [4], but expert demonstration is usually hard to obtain and hard to interpret. Recently, [3] proposed to use high-level macro-actions (a series of atomic actions) instead by incorporating classic path-planning techniques. By doing so, the agent is only trained to select optimal "goal points" rather than atomic actions, which significantly improves training efficiency. However, our insight is that navigating to a new goal point will come across a frontier point as shown in Fig. 2. Therefore we propose to produce frontier points instead of arbitrary points for the following reasons: (1) a frontier goal is a sub-goal of an arbitrary point and hence is quicker to navigation to; (2) a frontier point can be reached using path-planning since it is within the explored regions where the occupancy map is already constructed.

In this paper, we propose an autonomous exploration algorithm which integrates two exploration options implemented using the option-critic architecture: navigating to a selected frontier point; and investigating the local environment. An

✉ Juncheng Liu
ljc91122@icloud.com

1 Department of Computer Science, University of Otago, 133 Union St East, Dunedin, Otago 9016, New Zealand

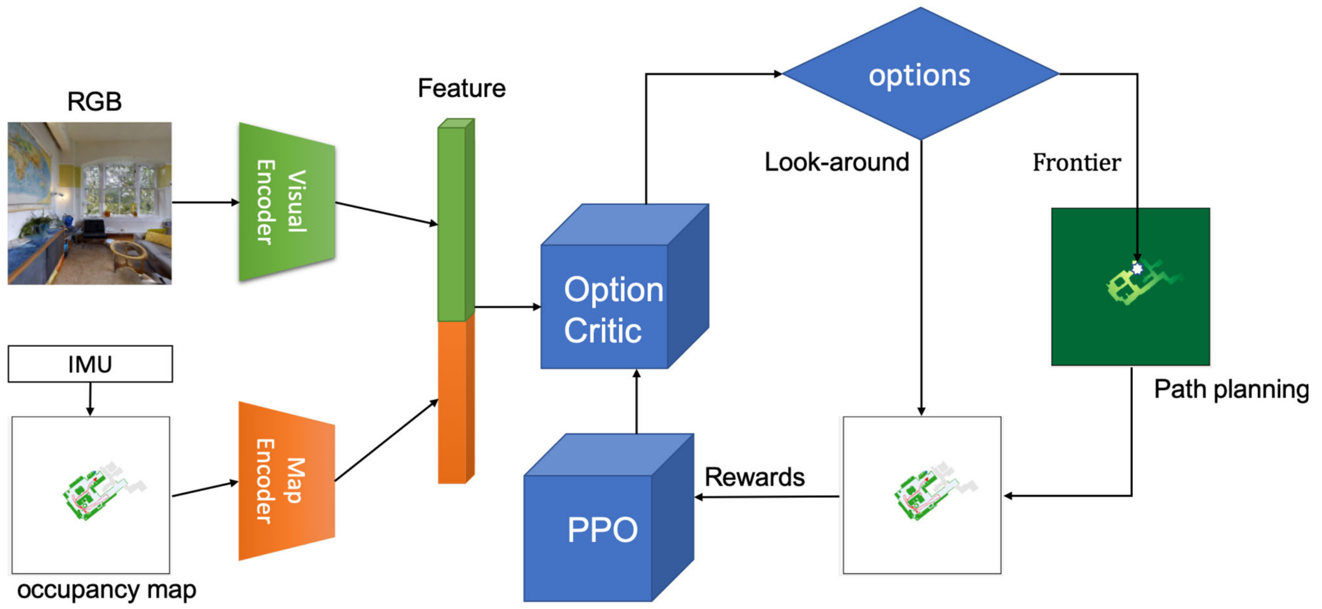

Fig. 1 Algorithm overview. Our method takes as inputs the RGB-D frames and the maintained maps. The feature vectors of them are concatenated and fed into the policy network. The policy chooses an option from "Frontier" and "look-around" according to the estimated values. An action is also produced by the chosen option. To navigate to a frontier, path-planning techniques are employed

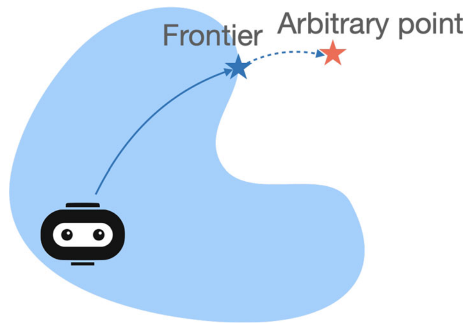

Fig. 2 Frontiers and arbitrary points. Our policy network produces frontier point instead of arbitrary points. Our insight is navigating to a new arbitrary point will come across at least one frontier point if new regions are discovered. Explored region is marked as blue and rest as white (color figure online)

overview of our method is shown in Fig. 1, and we make the following contributions:

- We show, for the first time, exploration in large spaces using RL across high-level task options that are interleaved in real time;
- We combine the classic frontier-based method with RL and its effectiveness is well studied;
- We extend the state of the art for exploratory navigation on the Gibson and Matterport3D datasets; and
- We show that our method produces not only quantitatively more efficient, but also qualitatively more compact exploration trajectories.

2 Related work

[21] proposed a frontier-based exploration method that always navigates to the closest frontier point and then always performs a rotation inspection. Subsequently, new frontiers are produced and the process repeats and eventually the whole scene is explored. There is, however, no learning involved.

Recently there have been multiple attempts to solve the autonomous exploration problem by learning. The motivation is that an agent is able to act more efficiently by leveraging the structural regularity and semantics of the scene. Another benefit is by using learning techniques more freedom of sensory modalities is achieved. For instance, depth information can be inferred by neural networks if only an RGB camera is available. The first end-to-end method to output atomic actions from sensory observations is [4]. Their policy network takes as inputs an RGB-D frame and the current occupancy map and outputs an action which the agent executes. The reward is designed as the incremental coverage of the occupancy map. This simple method requires imitation learning to initialize the weights of the network. In addition to the spatial memory, the method also integrates a temporal memory by employing GRUs (gated recurrent units). However, it is still not clear that the method without imitation learning outperforms the classical frontier method [21].

While coverage is a common metric, there are also alternative ways of defining rewards: [11] encourages the agent to visit states that are not predicted confidently, which is also called a curiosity-driven strategy. [18] uses the inverse of the square root of visitation number as the reward, encouraging the agent to move to under-explored areas. We point readers to [13] for a detailed taxonomy and explanation of existing exploration paradigms.

As a major improvement, [3] leverages a hierarchical structure and classic path-planning techniques to train a policy more efficiently. Instead of using atomic actions, its policy outputs locations where the agent is supposed to navigate to. Compared to atomic actions, temporal abstracted actions need fewer training episodes and do not require imitation learning. Instead of an actual occupancy map, [12] uses an anticipated occupancy map. With training, the anticipated map becomes more and more accurate. This allows the policy access to a complete (albeit approximate) map at a relatively early stage.

All the aforementioned learning methods have only one option which involves navigating to a certain point. Our method combines the frontier method and the line of work which takes advantage of a temporally abstracted policy. However, we employ the option-critic architecture [1, 17] which grants our policy two different options, that is, two options of behaviors, giving the agent more flexibility and efficiency. This surprisingly simple addition of a "look-around" option, improves the performance of the agent significantly.

3 Methods

3.1 Problem definition

The goal of autonomous exploration is to establish a 2D occupancy map of an environment as quickly as possible. Given a set of actions $\mathcal{A}$ that an agent can perform, e.g., rotate, move forward, the algorithm should be able to find an optimal trajectory of actions $\tau = \{a_1, a_2, \ldots, a_T \mid a_i \in \mathcal{A}\}$ such that for a given time step T, the highest coverage rate of the environment is reached. We solve this problem by estimating a policy, π, trained by RL.

We consider a mobile agent equipped with an RGB-D camera. For simplicity, we also assume the agent has an IMU module which records the current location relative to the initial position. For a robot without such functionality, some off-the-shelf classic SLAM algorithms [7, 9] or neural network-based estimation [3] can be applied instead. At each time step t, the policy π takes as input the current state, s_t which comprises: an RGB frame; and occupancy maps, the format of which will be discussed in detail in the following. The output of the policy is an action $a_t \in \mathcal{A}$ that maximizes the coverage.

3.2 Overview

It has recently been found that using higher-level macro-actions significantly improves the training efficiency without involving imitation learning which usually requires many expert demonstrations [3]. Similar to this approach, we adopt classic path-planning techniques [16] for navigation, which saves the agent from learning straightforward navigation tasks. Specifically, instead of outputting an atomic action, our policy π estimates a goal point to which the agent will navigate. Different to ANS [3] that uses arbitrary points, we incorporate the concept of "frontiers" which have been shown as a very effective strategy for exploration [21].

Furthermore, instead of simply performing navigation, we add a complementary option "look-around investigation". Our insight is a "look-around" operation is sometimes more efficient than wandering around. Imagine when you are in a new environment, the first step you would take is very likely to look around rather than immediately moving to a new location.

3.3 Map formats

It has been demonstrated that complex map-based architectures significantly improve the performance of exploration [6, 12]. Furthermore, an explicit map facilitates classic path-planning techniques. We also take advantage of such maps to help memorize the occupancy of environments and trajectory of an agent. Specifically, we use 5 maps to record this information. All of these maps have a dimension of 512×512 and are concatenated as a 5-channel input consisting of: occupancy map, explored map, trajectory map, current location map, frontier map. The occupancy map records obstacles. Since our agent has a depth camera, we are able to observe a point cloud and align it with the agent's current location in real time. Therefore we can build an occupancy map on the fly. It is a 512×512 binary map indicating whether a certain location is free or an obstacle. The explored map indicates which areas have been observed, and which are unexplored, while the current location and the past trajectory of the agent are stored in the current location map and trajectory map separately.

We also maintain a frontier map in real time. The frontiers are defined as the boundaries between the explored and unexplored areas except for the obstacles (e.g., walls). The computation of this map is very fast since it only involves logical bit-wise operations on the explored and occupancy maps.

For the feature extractor, we use seven 2D convolutional layers with kernel size of 3 and stride of 2 followed by two fully connected layers activated by ReLU. In addition to the maps, the RGB frames are also processed by ResNet18 [5] and concatenated with the map feature.

3.4 Training policy

As previously mentioned, our policy has two options:

1. Navigation to a selected frontier, and
2. look around at the current location.

We denote these as ω_1 and ω_2, respectively. At each time step t the policy π outputs an option ω_t as well as the action a_t based on its individual policy π_{ω_t}.

There are three main components in our policy networks parameterized by θ, η: the option-value function $V(s, \omega)$, intra-option policies $\pi_{\omega,\theta}$ and termination functions $\beta_{\omega,\eta}$. The option-value function $V(s_t, \omega_t)$ predicts the discounted returns of choosing option $\omega_t \in \{\omega_1, \omega_2\}$ given the current state s_t. The intra-option policies $\pi_{\omega,\theta}$ estimate the action distribution in the context of a state and an option. The termination functions $\beta_{\omega,\eta} \in (0, 1)$ give probabilities of terminating the current option. Moreover, there is a policy over options π_Ω which selects an option each time the current option is terminated. However we simply use a greedy policy for π_Ω, which always picks the option with the largest estimated value:

$$\pi_\Omega(s_t) = \arg\max_{\omega \in \{\omega_1, \omega_2\}} V(s_t, \omega). \tag{1}$$

In the following we will present the gradients of the intra-option policies $\pi_{\omega,\theta}$ and termination functions $\beta_{\omega,\eta}$ with respect to parameters θ and η. Given an option ω and state s, the gradient of an action is similar to that of Advantage-Actor Critic (A2C) [8]:

$$\sum_{a \in \mathcal{A}_\omega} \frac{\partial \pi_{\omega,\theta}(a \mid s)}{\partial \theta} A(s, \omega, a), \tag{2}$$

where $A(s, \omega, a)$ is the “advantage” of an action over the averaged rewards:

$$A(s, \omega, a) = r + \gamma V(s', \omega) - V(s, \omega), \tag{3}$$

s' is the next observation and γ is the discount factor. We use $\gamma = 0.99$ across all our experiments.

Different to [1], we do not assume the same action space for both options since a navigation target is a location (x, y), while look-around only needs an angle α. Therefore we use $\mathcal{A}_\omega$ to denote the action space of option ω.

Algorithm 1 Exploration with options

```
1: Choose the initial option $\omega_0$ according to $\pi_\Omega(s_0)$.
2: $\omega \leftarrow \omega_0$, $s \leftarrow s_0$
3: for $i = 1, \ldots, MaxSteps$ do
4:     Evaluation stage:
5:     Generate a macro-action $a \sim \pi_{\omega,\theta}(s)$
6:     if $\omega$ is "Frontier-navigation" then
7:         Navigate to $a$ in Environment using Path Planning and obtain $s', r$
8:     else
9:         Rotate by an angle $a$ in Environment and obtain $s', r$
10:    end if
11:    $s \leftarrow s'$
12:    Calculate $\beta_1, \beta_2$ and $V(s, \omega_1)$, $V(s, \omega_2)$.
13:    if Bernoulli($\beta_{\omega,\eta}$) then
14:        Update $\omega \leftarrow \pi_\Omega(s)$
15:    end if
16:    Training stage:
17:    $\theta \leftarrow \frac{\partial \log \pi_{\omega,\theta}(a|s)}{\partial \theta} A(s, \omega, a)$
18:
19:    $\eta \leftarrow \frac{\partial \beta_{\omega,\eta}(s)}{\partial \eta}(V(s, \omega) - \max_{\omega'} V(s, \omega'))$
20:    Update $\theta$ and $\eta$ based on Eq5.
21: end for
```

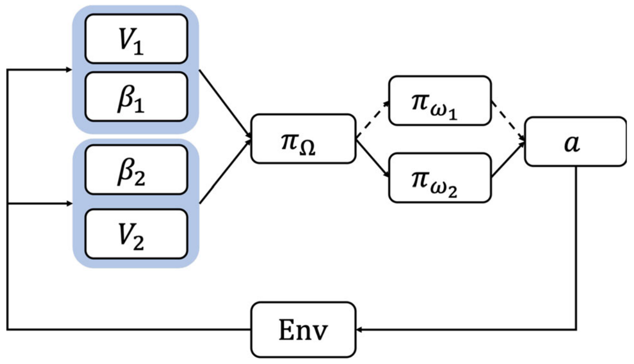

Fig. 3 Model diagram. After a macro-action is executed, values V and terminations β are computed. The next option is decided accordingly

η is updated as follows:

$$\sum_{\omega \in \{\omega_1, \omega_2\}} \frac{\partial \beta_{\omega,\eta}(s)}{\partial \eta}(V(s, \omega) - \max_{\omega'} V(s, \omega')), \tag{4}$$

where $V(s, \omega) - \max_{\omega'} V(s, \omega')$ acts in a similar way to advantage, which increases the termination probability of an option when its estimated value is suboptimal.

Finally, we update the value function parameters according to the TD-target:

$$r + \gamma((1 - \beta)V(s', \omega) + \beta \max_{\omega'} V(s, \omega')). \tag{5}$$

Our proposed approach is summarized in Algorithm 1, and its architecture is shown in Fig. 3. The method is divided into training and evaluation stages. During training, we employ a memory buffer which stores 20 different trajectories. The policy is then trained by these accumulated trajectories.

A description of the training process is as follows: The agent generates an RGB-D frame and a location signal at each time step. Our algorithm takes as inputs the RGB-D frame and the location signal and outputs a macro-action, that is, "look-around" or "point-navigation" using the intra-option policy $\omega_{\pi,\theta}$. After performing the action, the occupancy map is updated according to the observations. The increment of the map is then used as the reward signal to train the network parameters θ and η by using Eqs. 2 and 4, separately.

3.5 Action space

We use different action spaces for the two options. For the "Frontier-navigation" option the action space is 2D points $(x, y) \in (0, 1) \times (0, 1)$. The produced point is then scaled to the dimensions of our map (512 in all our experiments). The frontier which is closest to the point will be selected as target.

For the "look-around" option, its action space is simply a real number $\alpha \in (-1, 1)$. The number is then re-scaled to $(-\pi, \pi)$ which indicates the rotation angle.

Because there are two different action spaces for ω_1 and ω_2, we use two separate models for each of the policies. However, the two policy networks are jointly trained with and ultimately controlled by the intra-option policy π_Ω.

3.6 Rewards

Similar to [3, 4], we adopt the increase of coverage during exploration as rewards. This reward indicates how much new knowledge an action gains from the environment, and is also indicative of how efficient an action is. The new explored area can either be free space or obstacles. We use the ratio of increment to the total area of a scene instead of the absolute area considering that different scenes might have different scales.

4 Experiments

4.1 Experimental setup

We evaluate our proposed method in Habitat-Lab, a modular high-level library for end-to-end development in embodied AI [14] on two publicly available datasets: Gibson [20] and Matterport3D [2]. Both the datasets consist of 3D reconstructions of real-world indoor environments such as offices and homes. The average area of scenes in Matterport3D is larger than Gibson. For both datasets, there are training/test splits available. We use the *val* split for testing Gibson and the *test* set for Matterport3D. For each training/test scene (an episode), we use a maximum number of 1000 steps. All tasks required fewer than 1000 steps for the optimal solution. Setting the terminal time to 1000 steps was done to give algorithms a good chance of completing the task while trading off the time to complete the computational experiments. During our experiments we found that after 1000 steps all algorithms had either converged or were close to convergence. Furthermore, in practice, one must set either a maximum time or step budget, or a minimum coverage goal for a robot to achieve, and we have decided that a maximum time budget is more pertinent for real-world scenarios.

There are 14 test scenes in total which are not seen during training. Each scene has 71 different 2D rotations and initial locations of the agent for Gibson. In Matterport3D, there are 18 test scenes in this dataset in total and each one has 51 different rotations. The scenes in this dataset are generally much larger than the ones in Gibson and their layouts are more complicated, which makes exploration more difficult given the same number of time steps.

Although we make assumptions about the input modalities, we note that our method is not limited to a robot with such a configuration. The flexibility of input modalities can be achieved by adopting neural network-based approximators. However, since this is not the main focus of the paper, we simply use the ground-truth data of the simulator. For each time step, the agent observes an RGB-D frame of size 256×256 and its location. The aforementioned maps are maintained and updated in real time using this information.

Our algorithm is implemented in PyTorch [10]. All experiments were done on a PC with Intel Core i7-6700K CPU (4.00GHz) and an NVIDIA Quadro P6000 GPU (24GB memory).

4.2 Baselines

We use 3 alternative baselines for evaluation and comparisons. All of the baselines as well as our method are trained with the same 750 episodes (except for the frontier method which does not need training) and tested on the same split. It is worth noting that the test scenes are not seen during training and we do not train the policy during testing. To speed up the training process, we use 8 simulators to collect trajectories in parallel. Since we employ the A2C [8] algorithm, every simulator is synchronized when executing actions. For each training/evaluation episode, we consistently use 1k atomic action steps across all methods but a different number of macro-actions due to the scene scale. We use 25 steps for the macro-actions for Gibson (hence 40 macro-actions per episode) and 50 steps (20 macro-actions per episode) for Matterport3D dataset because its scenes are much larger than those in Gibson. Each macro-action includes a series of 25/50 atomic actions such as move forward, rotate left/right. The policy is updated by the collected trajectories after every 20 macro-actions are executed using proximal policy optimization [15].

Frontier method This is the classic method for exploration without any learning being involved [21]. The main idea is to chase after the closest frontiers until the whole environment is explored. The simple policy performs well and is guaranteed to converge given sufficient time.

Atomic action method The simplest attempt to engage RL to the problem is just maximizing the coverage given a set of atomic actions (*turn left, turn right, move forward*) such as [4]. To make the comparison straightforward, we do not employ imitation learning as done in [4]. Furthermore, we assume the agent is provided with the ground-truth locations.

Arbitrary point method Instead of navigating to frontier points, [3] proposed to navigate to arbitrary points on a map. Similarly, we also simplify its original implementation for the straightforwardness of comparison: we do not include SLAM and local policy components. Please note that these modifications should not worsen its performance for the following reasons: the visual SLAM component predicts the odometry by taking ground-truth locations as training samples. Therefore directly using imu output should not lead to deterioration; The local policy is trained to imitate the path-planning algorithm. Similarly, using the path-planning output directly should not worsen its performance. We make these two modifications for a simplicity of comparison.

The major difference between our proposed method and the alternatives is our method has two options (navigation and look-around), while the others only have one.

4.3 Metrics

We use two metrics for evaluating the performance of all methods: coverage percentage with time steps and coverage percentage with trajectory length. The difference between them is that trajectory length only considers the number of "*move forward*" actions, while time steps take all of the three actions into consideration. The length provides an effective way of measuring the neatness and efficiency of a trajectory. The evaluation of this metric is shown in Table 2. For a robot equipped with a rotatable camera, rotating its arm usually costs less energy than moving forward.

4.4 Results and comparisons

The results for Gibson and Matterport3D datasets are shown in Figs. 4 and 5, respectively. For each dataset, we show the overall, large scenes, and small scenes separately. Overall, our method achieves the best performance in both metrics and datasets. After a total number of 1000 time steps, our method achieves a coverage of 95% across all test scenes of Gibson on average, followed by 92% for the arbitrary point method, 88% for the frontier method. The atomic action method achieves the lowest coverage rate of 75%. The ranking of the methods for Matterport3D closely agrees with that of Gibson.

Table 1 collects the coverage rates of the four aforementioned approaches after 500 and 1000 time steps in the two datasets, respectively. It can be seen that for the same method and number of steps, the coverage of Matterport3D is lower than that of Gibson by approximately 20%-30%. When considering the coverage with trajectory length (Table 2), our method is much better than other methods, especially in Gibson dataset.

The results suggest that temporally abstracted macro-actions are generally more effective than atomic actions. This is due to the fact that classic path planning is employed to obtain optimal paths, while learning this usually requires much more training. By using high-level behaviors the policy focuses on goals with higher abstraction level. We note that the frontier-based method achieves decent coverage without learning. Especially at the beginning stage, it even outperforms the arbitrary point method. However, with the exploration progressing, the learning-based method is able to explore more areas as it learns the regularities from the past trajectories and observations.

We observe that the trajectories produced by our method appear more organized and cleaner than that of the fron-

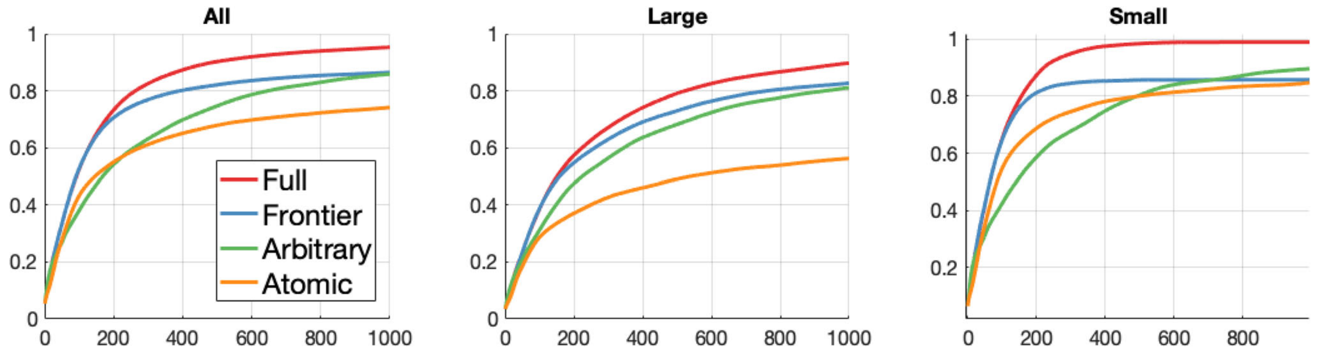

Fig. 4 Evaluation on Gibson. Plots show the averaged coverage as the episodes progress. Left: all scenes. Middle: large scenes. Right: small scenes

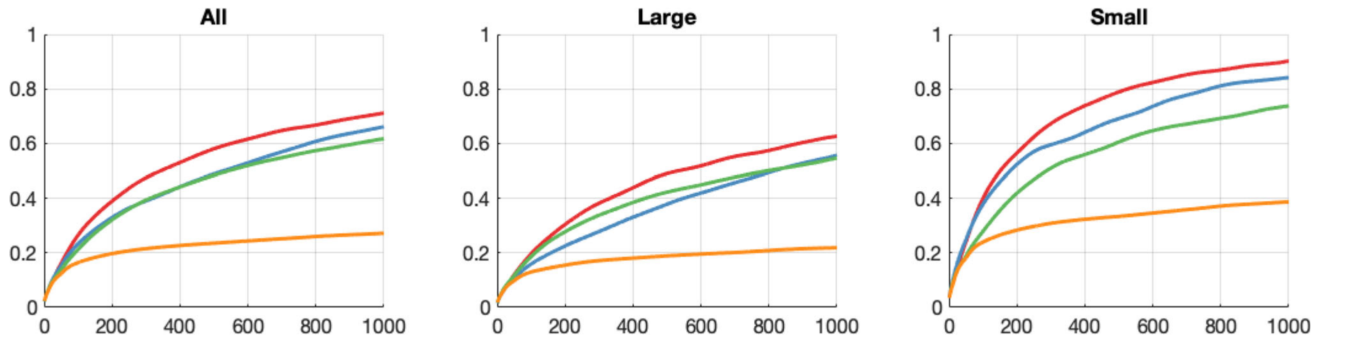

Fig. 5 Evaluation on Matterport3D. Plots show the averaged coverage as the episodes progress. Left: all scenes. Middle: large scenes. Right: small scenes

Table 1 Coverage after 500 and 1000 steps on Gibson and Matterport3D datasets (↑)

Methods	Gib500	Gib1k	Mat500	Mat1k
Full	**0.90**	**0.95**	**0.59**	**0.72**
Frontier	0.84	0.88	0.48	0.65
Arbitrary	0.82	0.92	0.48	0.62
Atomic	0.68	0.75	0.24	0.27

The symbol bold indicates the best results

Table 2 Average coverage incremental percentage per unit trajectory length on Gibson and Matterport3D datasets (↑). "A" for averaged. "L" for large scenes and "S" for small scenes

Methods	Gib/A	Gib/L	Gib/S	Mat/A	Mat/L	Mat/S
Full	**0.47**	**0.30**	**0.69**	**0.14**	**0.11**	**0.19**
Frontier	0.21	0.18	0.22	0.13	0.10	**0.19**
Arbitrary	0.12	0.12	0.12	0.09	0.09	0.11
Atomic	0.13	0.10	0.15	0.13	0.09	**0.19**

The symbol bold indicates the best results

tier method. This is the consequence of the option switching and planned frontier goals. For circumstances where a look-around action is more efficient than navigating to a point, the option-critic architecture enables switching between multiple options, leading to a shorter trajectory length.

We show the exploration trajectories in Fig. 13. This visually validates the neatness of the trajectories of our method. Our method is advantageous as it uses fewer *move forward* actions which are replaced by rotation wherever possible.

4.5 Ablations

4.5.1 Effectiveness of both components

As previously mentioned, we proposed two key improvements in our method: adding a frontier constraint when choosing the target point; adding a look-around option. To validate our method benefits from both improvements, we did an ablation study in which we remove one component at a time, which leads to four variants: full method, full method without frontier component (arbitrary point + look-around), full method without look-around option (arbitrary point + frontier), and the method without both components (the arbitrary point method). They are summarized in Table 3. Their corresponding performance can be seen in Fig. 6.

Table 3 Ablation study variants

	w/ frontier	w/o frontier
w/ look-around	full	w/o F
w/o look-around	w/o R	w/o FR

To examine the performance under different scene scales, we also report the small, large splits in addition to the overall results. The overall performance of the four variants suggests that both components help since the method without both components has the worst performance. Adding a frontier constraint for the target point is more advantageous in smaller scenes than large ones. This coincides with the fact that the frontier-based method performs better in smaller scenes than arbitrary point method as we consistently observed in Figs. 4 and 5. The look-around component is beneficial in both small and large scenes and is more important than the frontier component (w/o F outperforms w/o R by approximately 10% after 1k steps).

Therefore we draw the following conclusions for the ablation study: (1) both components help; (2) frontier constraint is more beneficial to smaller scenes; (3) look-around component contributes more (Fig. 7).

It can also be observed from Fig. 13 that the trajectories of the method without look-around option are generally more messy and the coverage is less after the same number of time steps. We show the selections and distribution of options from 10 distinct trajectories in Fig. 8. Generally speaking, the navigation option is selected more frequently than look-around. A common pattern of behavior that we observe across different scenes is that the agent starts exploring by first looking

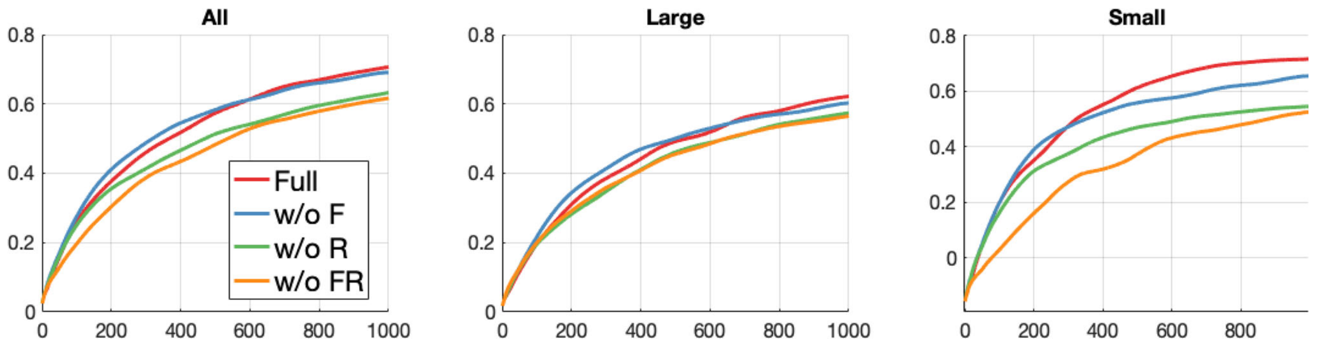

Fig. 6 Ablation study on MP3D. Full: full method. w/o F: full method without frontier component (arbitrary point + look-around). w/o R: full method without look-around option (arbitrary point + frontier). w/o FR: the method without both components (the arbitrary point method)

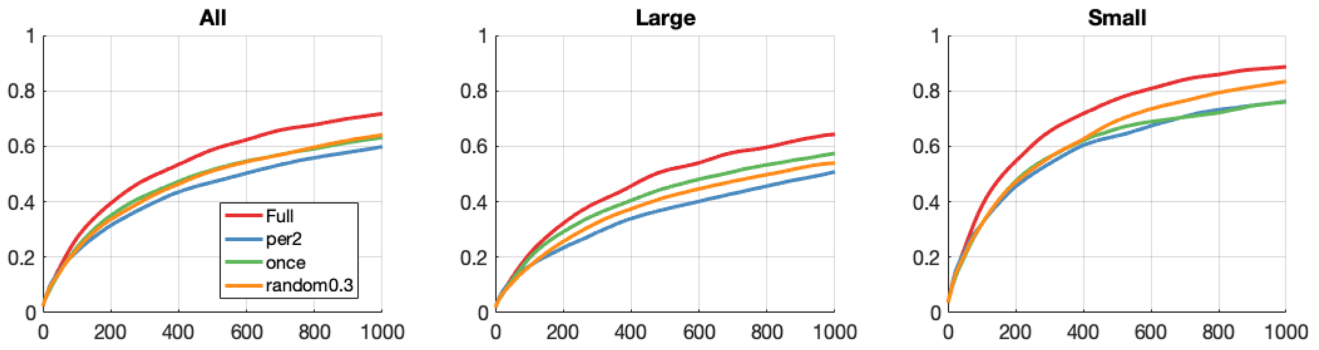

Fig. 7 Look-around option validation on MP3D. per2: perform a look-around after each navigation step. once: perform a look-around once at the initial stage. random0.3: perform a look-around by a probability of 0.3

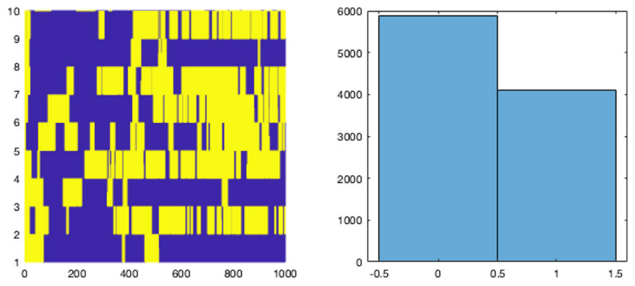

Fig. 8 Option selection and histogram. Left: 10 option selections in 10 trajectories. The x-axis shows the time steps and the y-axis has a row for each trajectory. Navigation is shown as purple and Look-around yellow. Right: frequency of options. Navigation is 0, look-around is 1

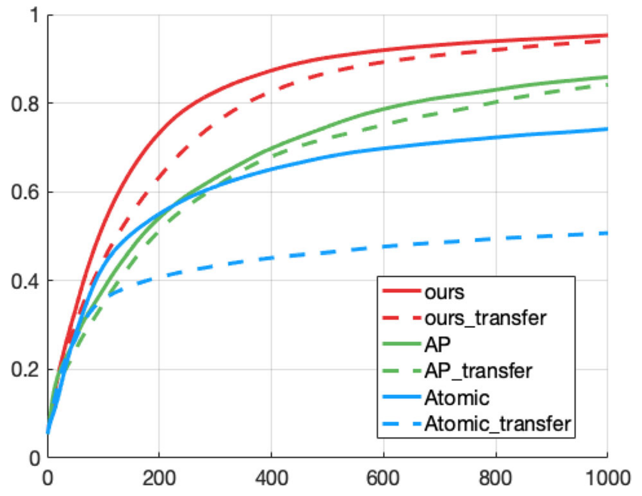

Fig. 9 Domain generalization performance. We evaluate the performance on Gibson with the model trained on Matterport3D. Solid and dotted lines indicate the corresponding original and domain-transferred performance, respectively

around. Then there is a period mostly dominated by navigating to a frontier point. Then, when most of the scene is explored, the agent tends to choose look-around more often as shown in left of Fig. 8.

4.5.2 Look-around validation

In the previous section, we showed that the look-around option implemented by the option-critic architecture is beneficial to the exploration task. To verify that an "intelligent" look-around is necessary, we tested several naive look-around baselines in the experiments: (1) a look-around is performed after each navigation step; (2) perform a look-around once at the initial stage; (3) perform a look-around randomly by a probability of 0.3. The results can be seen in Fig. 7. Overall, our learned look-around outperforms the baselines by approximately 10%. It can be observed that the "look-around after each navigation step" has the worst performance since it wastes time examining already well-explored

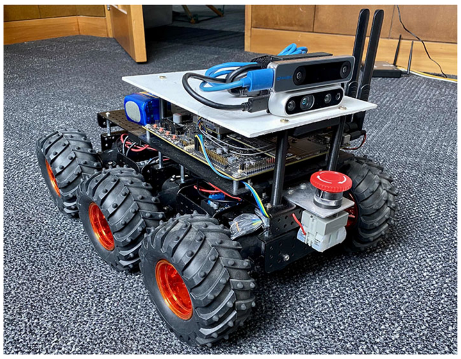

Fig. 10 Rover robot used in real-world transfer. The policy model is trained on simulator and deployed on the real robot. It is equipped with an IMU as well as an RGB-D module

surroundings, while our learned policy intelligently chooses to look around in more appropriate situations.

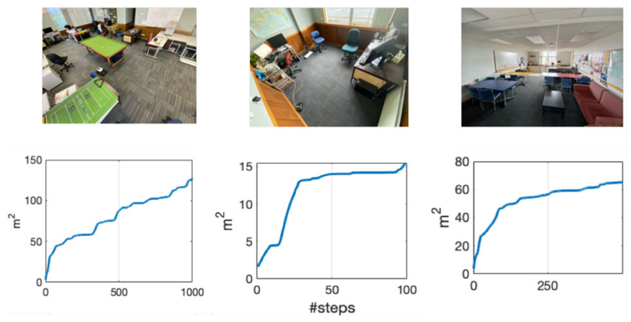

Fig. 12 Real-world transfer performance in 3 scenes. x-axis represents the number of steps, while the y-axis represents the explored area in m^2

4.6 Domain generalization

We also evaluate the generalization ability of the above methods. In this experiment, every method is tested on Gibson dataset with the model trained on Matterport3D. The purpose is to test if a method can generalize well from one domain to

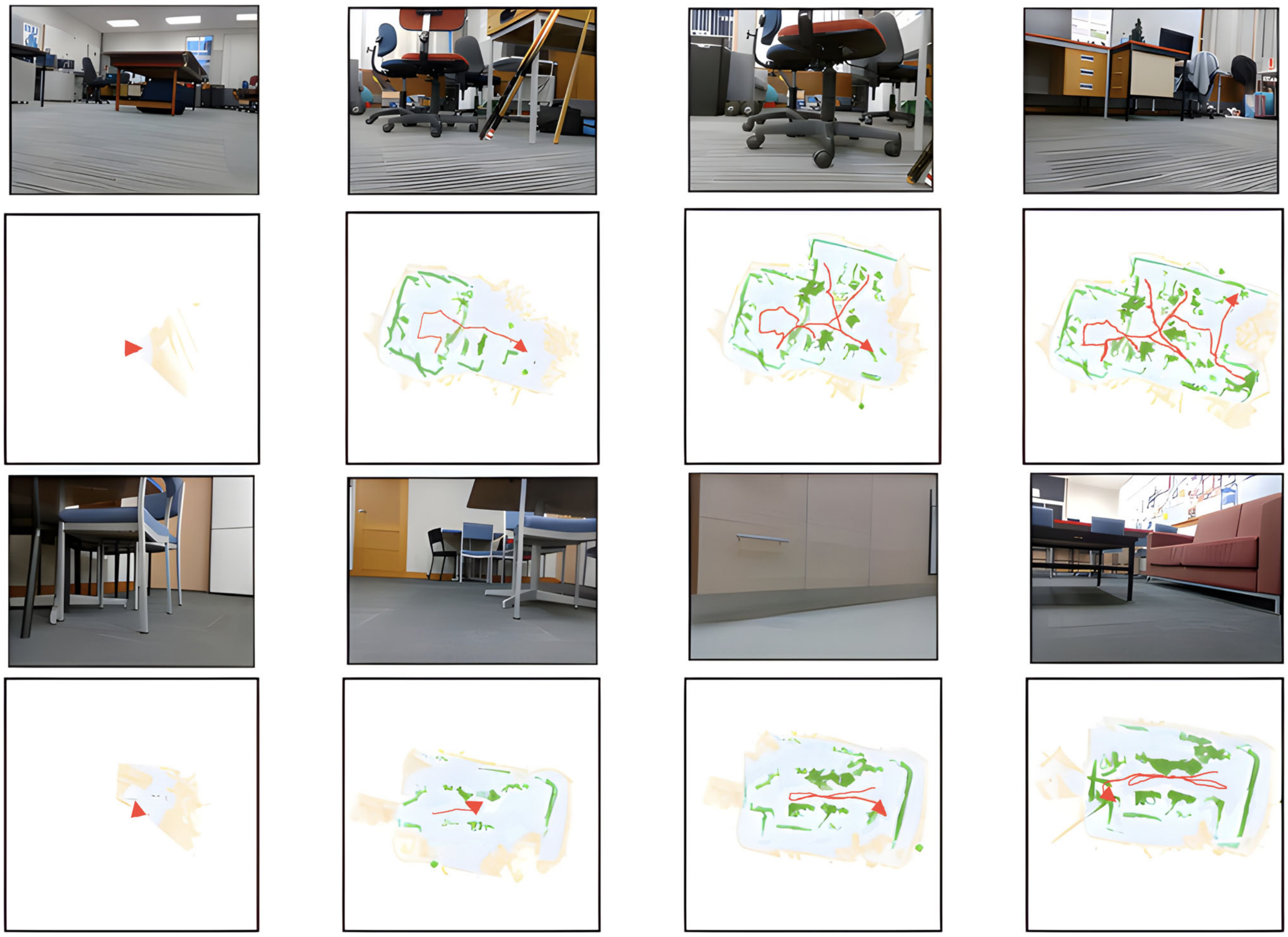

Fig. 11 Real-world transfer. Figure showing the progress of agent exploring two different scenes. For each scene we show the RGB frames and the constructed maps at the initial stage, 30%, 60% and 100% of each corresponding number of time steps

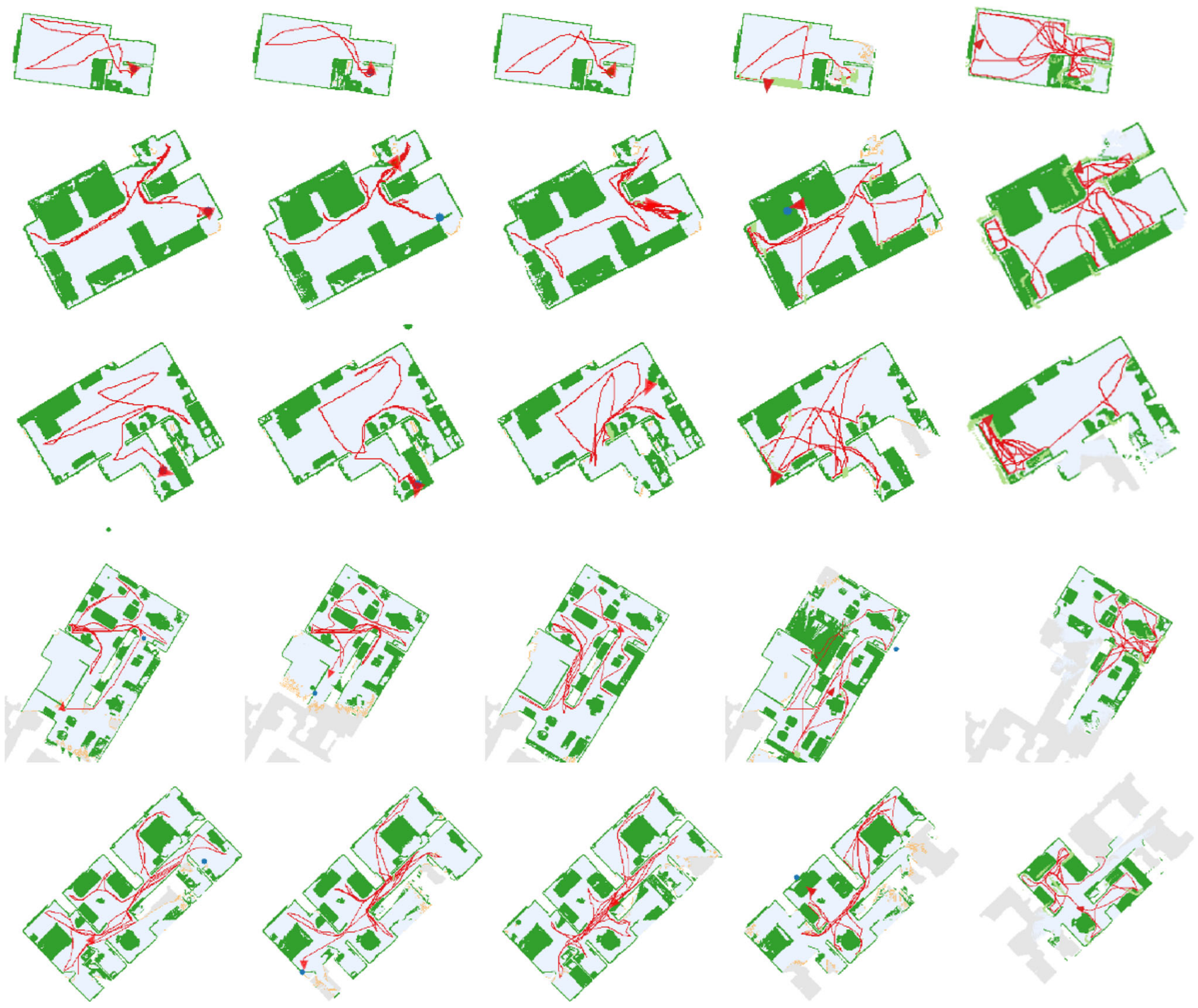

Fig. 13 Trajectories of different methods after 1000 time steps. From left to right columns: our approach, look-around option disabled, frontier method, arbitrary point method and atomic action method. Each row represents a selected scene. Green areas represent obstacles, while light blue represents free space. Red lines indicated the trajectories of agent. Orange dots are the frontiers. Unexplored areas are in gray. We can observe from the figure that our proposed method covers most area compared to alternatives after 1k steps. Besides, the trajectories of the agents trained by our method are more compact. This means the movements are more efficient and hence less energy consumption (color figure online)

another. Since the frontier method does not involve learning, it should remain the same as in previous experiments. The results can be seen in Fig. 9 which shows the coverage with time steps of the three aforementioned methods. We observe a performance drop for all methods. Our method and arbitrary point method have a comparable drop rate, while the atomic action method deteriorates significantly. This suggests RL methods with more abstract actions are more robust to environment change and hence better transfer ability.

4.7 Real-world transfer

To evaluate how well the trained model transfers in a real-world environment, we deploy our proposed algorithm on a rover robot equipped with an Intel RealSense RGB-D camera D435 and a tracking camera T265 as shown in Fig. 10. We discretize the action space of the motors so that the 3 atomic actions used in the simulator can be applied.

We test our method on three real environments: a large office, a small office and a kitchen. Since they have different scales ($120m^2$, $15m^2$, $60m^2$ approximately), we use different total numbers of steps for each individual scene (1000, 100 and 500 steps). The RGB frames and the constructed maps of two environments at the initial stage, 30%, 60% and 100% of each corresponding number of time steps, are shown in Fig. 11. A demonstration video showing this process can be found in supplementary materials. We observe very similar behavior patterns as we see in the simulator: the agent always performs a look-around at the initial stage; after most spaces are explored the agent tends to select look-around option more frequently. After certain number of steps, most space is explored and the corresponding occupancy maps are constructed.

Thumbnail pictures of the three scenes and their corresponding coverage with time steps curves are shown in Fig. 12. The curves agree with the ones we tested in simulator, which suggests our method is effectively transferred to a robot in real-world environments. One advantage of using abstract high-level actions is that it is able to accommodate different implementation details of atomic actions. For instance, the "turn-left" may have a different rotation angle compared to the one in simulator; however, it does not have an influence on the policy network. Note that real robots might have different sizes and shapes; therefore, we need to set obstacle boundaries accordingly so that robots can move and rotate freely without stalling (Fig. 13).

5 Conclusion

In this paper, we proposed a method that integrates two options for autonomous exploration. One option is navigating to a selected frontier using classic path-planning techniques. The other complementary option is looking around the surroundings by a certain angle. We validated its effectiveness on two publicly available datasets and real-world environments. The results outperform the three baselines we tested. We also validated that the method benefits from both options by an ablation study.

To conclude, we have the following findings in our research: it is more effective and efficient to have multiple options for exploration. Note that our method is able to integrate any number of options even though we only used two in our implementation. Classic frontier-based method is more advantageous in the early stage. Employing path-planning techniques requires fewer episodes, making training more efficient. Temporally abstracted actions are generally more effective than atomic actions. All of these are consistent across all our experiments.

Future work includes integrating more options and exploring how these options can cooperate and what is the minimum set of options that achieves the best performance. Another further direction is a policy outputting a higher hierarchical abstracted action such as a whole planned trajectory. We expect this to take longer-term planning into consideration which makes the trajectory even shorter and hence more action-efficient.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s00138-023-01492-1.

Acknowledgements This project was funded by Science for Technological Innovation (https://www.sftichallenge.govt.nz/.) under the spearhead project: Adaptive learning robots to complement the human workforce.

References

1. Bacon, P.L., Harb, J., Precup, D.: The option-critic architecture. In: Proceedings of the AAAI Conference on Artificial Intelligence (2017)
2. Chang, A., Dai, A., Funkhouser, T., et al.: Matterport3d: learning from RGB-D data in indoor environments. arXiv preprint arXiv:1709.06158 (2017)
3. Chaplot, D.S., Gandhi, D., Gupta, S., et al.: Learning to explore using active neural slam. arXiv preprint arXiv:2004.05155 (2020)
4. Chen, T., Gupta, S., Gupta, A.: Learning exploration policies for navigation. arXiv preprint arXiv:1903.01959 (2019)
5. He, K., Zhang, X., Ren, S., et al.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
6. Henriques, J.F., Vedaldi, A.: Mapnet: an allocentric spatial memory for mapping environments. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 8476–8484 (2018)
7. Klein, G., Murray, D.: Parallel tracking and mapping for small ar workspaces. In: 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality. IEEE, pp. 225–234 (2007)
8. Mnih, V., Badia, A.P., Mirza, M., et al.: Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning, PMLR, pp. 1928–1937 (2016)
9. Mur-Artal, R., Montiel, J.M.M., Tardos, J.D.: Orb-slam: a versatile and accurate monocular slam system. IEEE Trans. Rob. **31**(5), 1147–1163 (2015)
10. Paszke, A., Gross, S., Massa, F., et al.: Pytorch: an imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., et al (eds) Advances in Neural Information Processing Systems 32. Curran Associates, Inc., pp. 8024–8035, http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (2019)
11. Pathak, D., Agrawal, P., Efros, A.A., et al.: Curiosity-driven exploration by self-supervised prediction. In: International Conference on Machine Learning, PMLR, pp. 2778–2787 (2017)
12. Ramakrishnan, S.K., Al-Halah, Z., Grauman, K.: Occupancy anticipation for efficient exploration and navigation. In: European Conference on Computer Vision. Springer, pp. 400–418 (2020)
13. Ramakrishnan, S.K., Jayaraman, D., Grauman, K.: An exploration of embodied visual exploration. Int. J. Comput. Vis. **129**(5), 1616–1649 (2021)
14. Savva, M., Kadian, A., Maksymets, O., et al.: Habitat: A platform for embodied AI research. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 9339–9347 (2019)
15. Schulman, J., Wolski, F., Dhariwal, P., et al.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)

16. Sethian, J.A.: A fast marching level set method for monotonically advancing fronts. Proc. Natl. Acad. Sci. **93**(4), 1591–1595 (1996)
17. Sutton, R.S., Precup, D., Singh, S.: Between mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning. Artif. Intell. **112**(1–2), 181–211 (1999)
18. Tang, H., Houthooft, R., Foote, D., et al.: # exploration: a study of count-based exploration for deep reinforcement learning. In: 31st Conference on Neural Information Processing Systems (NIPS), pp. 1–18 (2017)
19. White, C.C.: A survey of solution techniques for the partially observed Markov decision process. Ann. Oper. Res. **32**(1), 215–230 (1991)
20. Xia, F., Zamir, A.R., He, Z., et al.: Gibson env: real-world perception for embodied agents. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 9068–9079 (2018)
21. Yamauchi, B.: A frontier-based approach for autonomous exploration. In: Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'. IEEE, pp. 146–151 (1997)