Supplementary Materials for "Learning Active Camera for Multi-Object Navigation"

In the supplementary, we provide more implementation details and more experimental results of our EXPO camera policy. We organize the supplementary as follows.

- In Section A, we provide the differences of active cameras from panoramic cameras.
- In Section B, we provide more discussion on how to coordinate two types of policies.
- In Section C, we provide more architecture details on our EXPO camera policy.
- In Section D, we provide more details on updating the top-down map.
- In Section E, we provide more experimental details, including training settings, environment details, and evaluation metrics.
- In Section F, we provide experimental results on multi-object navigation with different number of goal objects.
- In Section G, we provide experimental results to explain the choice of step budget for evaluation.
- In Section H, we provide more discussions on oracle information used in experiments.
- In Section I, we provide analysis on failure cases.
- In Section J, we discuss potential future researches and social impact.

A Differences of active cameras from panoramic cameras

In this paper, we propose an active camera to coordinate camera and navigation actions for perceiving a novel environment more efficiently. An alternative is to use panoramic cameras for capturing panoramic images. However, panoramic images are often in a very large size and contain a lot of redundant information, requiring a huge amount of resources to analyze. It may not be practical to equip panoramic cameras on mobile platforms. In this case, designing an active camera would be a more practical way to capture necessary environmental information required for navigation at a much cheaper cost.

An agent with multiple fixed pinhole cameras could make for an interesting comparison with our active-camera agent. To compare our active-camera agent with the multi-fixed-camera agent, we equip an agent with four fixed pinhole cameras facing different directions (*i.e.,* front, left, right, and back). Specifically, we concatenate the features of these four images and feed them to an end-to-end navigation baseline (*i.e.,* MultiON). We train this baseline for 30 million frames, which is the same as our active-camera agent. In Table A, the multi-fixed-camera agent performs better than the single-fixed-camera agent but still worse than our single-active-camera agent. We speculate this is because the end-to-end learned neural network is hard to map this larger amount of information (four RGBD images) to correct actions. We note that the multi-fixed-camera results in Table A are different from the results reported in Openreview during the rebuttal phase. This is because we train the agent using different numbers of parallel threads. We found that using less threads for training significantly decreases performance. The possible reason is that the model may be unstable during training if capturing training data from only a few environments. In Table A, all variants use the same number of parallel threads for training (*i.e.,* 36).

Table A: Comparisons of different camera types based on an end-to-end agent (*i.e.,* MultiON).

	MatterPort3D			
Camera Types	SPL	PPL	Success	Progress
Single-Fixed-Camera (MultiON)	33.0	43.8	44.1	60.5
Multi-Fixed-Camera	37.75	48.3	49.1	65.6
Single-Active-Camera (Ours)	**38.7**	**49.5**	**51.1**	**67.3**

We also have conducted multi-fixed-camera experiments on the SLAM-based method. The information captured by multiple cameras helps agents build a map efficiently. This map provides more information for path-planning. The results are shown in Table B. The multi-fixed-camera agent

outperforms our active camera agent. Because the SLAM-based multi-fixed-camera agent does not need to learn a neural network to map RBGD observation to navigation action, it will not suffer from learning difficulty problems as in e2e-based agents. It is worth noting that although using a single camera, our active camera policy helps the agent achieve comparative performance compared with an agent with four fixed cameras.

Table B: Comparisons of different camera types based on slam-based agents.

	MatterPort3D			
Camera Types	SPL	PPL	Success	Progress
Single-Fixed-Camera (OccAnt)	53.0	57.7	72.0	80.2
Multi-Fixed-Camera	**68.3**	**72.2**	**79.4**	**85.3**
Single-Active-Camera (Ours)	57.9	62.1	75.6	82.6

B More discussion on the coordination between two types of policies

When incorporating camera policy with an end-to-end navigation policy, we use one policy network (*i.e.,* Navigation-Camera Joint Policy) to jointly predict both camera and navigation actions, as described in Section 3.4 in the main paper. The paradigm is shown in Figure A. We feed this policy network both camera policy inputs (*i.e.,* map and heuristic direction) and navigation policy inputs (*i.e.,* RGB-D images, target, and previous action). In this way, the policy network better coordinates two types of actions. We have tried a variant that uses two separate policy networks to predict camera and navigation actions, respectively, which is similar to the paradigm in Figure 2 in the main paper. In Table C, this separated variant performs worse than our joint policy and even worse than the baseline without an active camera. We suspect this is because the navigation policy can not control camera direction to perceive desired observations for deciding navigation action. As a result, an agent performs poorly in navigation. These observations are consistent with the experimental results in Table 3 in the main paper. In Table 3, the agent performs worse when using rule-based camera actions because these camera actions are uncontrollable by the navigation policy either.

Table C: Comparisons between separated and joint navigation-camera policies when incorporating a camera policy with an end-to-end navigation policy.

	MatterPort3D			
Method	SPL	PPL	Success	Progress
MultiON	33.0	43.8	44.1	60.5
+ Active Camera (Separated)	29.1	40.4	37.3	54.1
+ Active Camera (Joint)	**38.7**	**49.5**	**51.1**	**67.3**
DD-PPO	16.7	29.2	22.2	40.9
+ Active Camera (Separated)	12.2	27.1	13.6	32.7
+ Active Camera (Joint)	**19.1**	**34.0**	**24.0**	**43.9**

When incorporating a camera policy with a SLAM-based navigation policy, two types of policies decide actions separately as shown in Figure 2 in the paper. Although the agent does not decide two types of actions jointly, two types of policies cooperate with each other from the following two aspects. 1) The camera policy cooperates with navigation action by taking it as input when deciding camera action. 2) The navigation policy cooperates with camera action by an off-the-shelf path-planning algorithm (*e.g.,* if the agent finds a better path toward goal objects after executing a camera action, the navigation policy will change the navigation action accordingly).

C More architecture details on camera policy

The general scheme of our active-camera agents, *i.e.,* , incorporating a camera policy with a modular SLAM-based [4, 9] navigation policy or an end-to-end learning-based [6, 7] navigation policy, are shown in Figure 2 and Figure A, respectively. We feed map features, heuristic direction features and navigation features to a policy network to predict camera action. These features are extracted from three encoders, respectively. Next, we describe the architecture details of these encoders and the policy network.

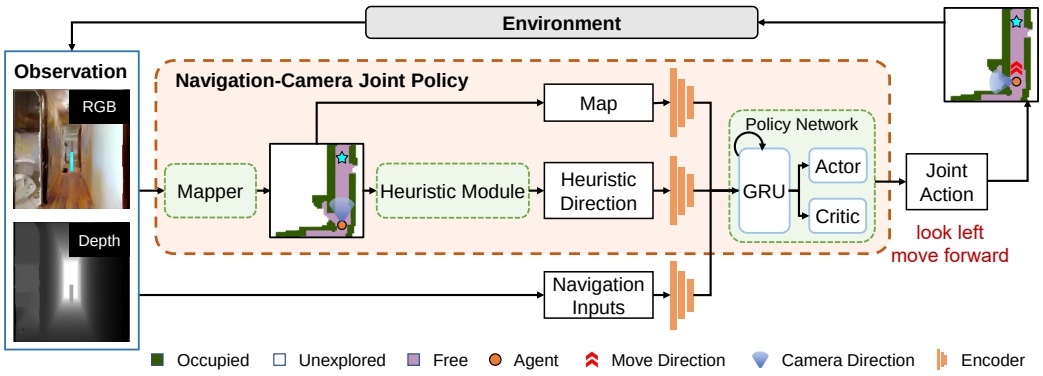

Figure A: General scheme of an active-camera agent when incorporating camera policy with an end-to-end navigation policy.

The map encoder consists of three convolution layers followed by a fully connected layer. We use ReLU as an activation function after each layer. The setting of three convolution layers (kernel size, stride) are $(4 \times 4, 2)$, $(3 \times 3, 2)$ and $(2 \times 2, 1)$, respectively. The fully connected layer outputs 512-dimension map features $\mathbf{v}_m$. The heuristic direction encoder is a learned embedding layer, transforming the input heuristic action into 32-dimension features $\mathbf{v}_h$. As for navigation encoder, its architecture depends on what kind of navigation policy we incorporate the camera policy with. If we use a SLAM-based navigation policy that output an navigation action, the navigation encoder is a learned embedding layer which transforms the action into 32-dimension features $\mathbf{v}_n$. If we use an end-to-end navigation policy, the encoder of this navigation policy [6, 7] would become our navigation encoder, transforming navigation input (typically RGB-D images) to latent features. With the features from these three encoders, we concatenate them to build the input $\mathbf{v} = [\mathbf{v}_m, \mathbf{v}_h, \mathbf{v}_n]$ for the policy network. The policy network consist of a one-layer gate recurrent unit (GRU), followed by two individual fully connected layers corresponding for predicting camera action and values of the agent's current state, respectively.

D Updating top-down map by a registration function $R(\cdot)$

As described in Section 3.2 in the paper, we use a registration function $R(\cdot)$ to update the global top-down occupancy map $\mathbf{M}^g$, *i.e.,* $\mathbf{M}^g = R(\mathbf{M}^l, \mathbf{M}^g, q)$, where $\mathbf{M}^l$ is a local map and q is the relative pose change of the agent from the first time step. After that, we crop an egocentric map from the global as camera policy input. We describe the details of this registration function below.

All these three types of maps are with the same resolution. Each pixel in a map represents $0.08 \times 0.08m^2$ area in the real world. The size of local map, global map and egocentric map are 61×61, 3000×3000, and 125×125, respectively. These maps represent $4.88 \times 4.88m^2$ area in front of the agent, $240 \times 240m^2$ area in the real-world, and $10 \times 10m^2$ area around the agent, respectively. The global map is with a larger size and covers the whole environment. In contrast, the local map represents only a small area, which is projected from the visual observation within a 79° field-of-view in front of an agent. To merge these two maps, we initialize a blank map with the same size as the global map. We put the local map in the center. Then, we use affine transformation [3] on this map based on the pose q. This affine transformation transforms the local to its location on the global map. After that, we aggregate the transformed map with the global map by a max-pooling operation.

E More experimental details

More implementation details. We use PyTorch to implement our active-camera agent. We train it using 36 parallel threads on 3 Nvidia Titan X GPUs. Each thread contains one of the scenes from the training set. The agent is trained using PPO with 4 mini-batches and 2 epochs in each PPO update. We use Adam optimizer with a learning rate of 2.5e-4, a discount factor of $\gamma = 0.99$, an entropy coefficient of 0.001, a value loss coefficient of 0.5. We train the proposed camera policy together with the SLAM-based and learning-based methods for 2 and 30 million frames, respectively.

More details on environments. We use Habitat simulator[1] [5] along with two large-scale photorealistic 3D indoor environments (*i.e.,* Matterport3D [1] and Gibson [8]). Both environments consist various of scenes such as house, church, and factory. For Matterport3D, we use the dataset proposed in MultiON [6] for the multi-object navigation task. In this dataset, 79 scenes are divided into two disjoint parts, 61 scenes for training and 18 scenes for testing. For Gibson, we evaluate the transferability of our camera policy. As the MultiON paper does not provide a corresponding multi-object dataset, we use the test scene split provided by Active Neural SLAM[2] and follow the episode generation rule in MultiON (*i.e.,* the geodesic distance between two successive goals is in the range of 2m and 20m) to generate 861 multi-object episodes within 14 different scenes for testing. For each episode, we ensure that all objects and start positions are on the same floor.

More details on metrics. We follow MultiON [6] to evaluate multi-object navigation in terms of success rate and navigation efficiency. A good agent should successfully navigate to all goal objects through the shortest path. The details are described below.

- **Success**: ratio of successfully navigating to all goal objects and calling FOUND in a correct order within an allowed time step budget.
- **Progress**: the fraction of goal objects being successfully found.
- **SPL**: Success weighted by Path Length. Concretely, $\mathrm{SPL} = s \times d/max(d, \bar{d})$, where s indicates the value of Success metric, d is the shortest geodesic distance from the starting point to all objects in order, $\bar{d}$ indicates the geodesic distance traveled by the agent.
- **PPL**: Progress weighted by Path Length. This is an extended version of SPL based on progress. Concretely, $\mathrm{PPL} = p \times d'/max(d', \bar{d}')$, where p indicates the value of Progress metric, d' and $\bar{d}'$ is similar to d and $\bar{d}$ in SPL metric but only account for the path through all founded objects.

F More results on 1-ON, 2-ON and 3-ON episodes

In order to evaluate the efficiency of our camera policy for the navigation tasks with different difficulties, except for 3-ON episodes (*i.e.,* navigating to 3 different objects in an environment), we follow MultiON [6] to evaluate on 2-ON and 1-ON episodes on both Matterport3D and Gibson datasets.

Performance on Matterport3D dataset. In Table D our camera policy consistently improves the navigation performance based on different baselines. These results suggest that actively moving camera following our camera policy helps agents explore the environment and locate objects more efficiently. These abilities benefit both the single object navigation task and the multi-object navigation task.

Table D: Object navigation results (%) on 1-ON, 2-ON and 3-ON episodes on Matterport3D dataset.

Method	SPL			PPL			Success			Progress		
	1-ON	2-ON	3-ON	1-ON	2-ON	3-ON	1-ON	2-ON	3-ON	1-ON	2-ON	3-ON
OccAnt	60.9	55.2	53.0	60.9	58.3	57.7	89.2	79.4	72.0	89.2	84.3	80.2
+Naive Camera Policy	58.6	52.2	48.7	58.6	55.4	53.1	85.7	75.5	69.1	85.7	80.6	76.8
+Our Camera Policy	**66.9**	**61.4**	**57.9**	**66.9**	**64.1**	**62.1**	**90.3**	**81.8**	**75.6**	**90.3**	**86.0**	**82.6**
Mapping+FBE	49.1	44.0	40.1	49.1	46.9	45.4	82.6	**72.6**	62.3	82.6	77.6	72.5
+Naive Camera Policy	45.5	38.7	35.2	45.5	42.5	41.2	79.3	65.4	55.3	79.3	72.3	66.9
+Our Camera Policy	**55.0**	**47.2**	**44.6**	**55.0**	**51.1**	**49.8**	**84.7**	71.8	**64.2**	**84.7**	**78.2**	**74.1**
MultiON	55.9	43.6	33.0	55.9	49.3	43.8	77.9	59.9	44.1	77.9	68.9	60.5
+Naive Camera Policy	59.6	44.6	32.4	59.6	52.0	45.1	80.2	59.2	41.9	80.2	69.9	60.2
+Our Camera Policy	**62.1**	**50.8**	**38.7**	**62.1**	**56.4**	**49.5**	**84.2**	**67.5**	**51.1**	**84.2**	**75.9**	**67.3**
DD-PPO	43.5	27.5	16.7	43.5	35.5	29.2	62.3	38.0	22.2	62.3	50.2	40.9
+Naive Camera Policy	47.5	27.7	16.7	47.5	37.7	30.4	61.1	35.2	20.8	61.1	48.4	39.2
+Ours Camera Policy	**51.2**	**31.8**	**19.1**	**51.2**	**41.2**	**34.0**	**66.3**	**41.5**	**24.0**	**66.3**	**53.8**	**43.9**

[1] https://aihabitat.org/

Transferability results to Gibson dataset. In Table E, we show the Transferability results to the Gibson dataset. Most of the results share the same trend as the results on the Matterport3D dataset, *i.e.,* our EXPO camera policy improves the object navigation performance in terms of Success, Progress, SPL, and PPL based on different baselines. These results demonstrate that our method can generalize well to different domains. This allows us to train the agent on the collected photorealistic datasets and then easily deploy it in the real-world environment.

Table E: Transferability results (%) to Gibson dataset on 1-ON, 2-ON and 3-ON episodes.

Method	SPL			PPL			Success			Progress		
	1-ON	2-ON	3-ON	1-ON	2-ON	3-ON	1-ON	2-ON	3-ON	1-ON	2-ON	3-ON
OccAnt	79.6	77.8	76.1	79.6	78.7	77.8	94.5	92.0	89.0	94.5	93.2	91.8
+Naive Camera Policy	73.9	72.0	69.9	73.9	72.9	71.9	91.7	88.4	84.8	91.7	90.0	88.3
+Ours Camera Policy	**82.4**	**80.8**	**78.9**	**82.4**	**81.6**	**80.7**	**94.7**	**92.7**	**89.9**	**94.7**	**93.7**	**92.4**
Mapping+FBE	67.7	65.0	62.1	67.7	66.3	64.9	91.5	86.3	80.9	91.5	88.9	86.2
+Naive Camera Policy	61.2	58.5	55.8	61.2	59.9	58.5	88.7	83.0	77.5	88.7	85.9	83.1
+Ours Camera Policy	**73.8**	**71.2**	**68.7**	**73.8**	**72.5**	**71.2**	**93.6**	**88.7**	**84.4**	**93.6**	**91.1**	**88.9**
MultiON	68.1	62.1	56.5	68.1	65.1	62.2	86.2	77.2	68.4	86.2	81.7	77.3
+Naive Camera Policy	69.2	61.2	54.1	69.2	65.2	61.6	86.0	74.5	64.5	86.0	80.0	75.2
+Ours Camera Policy	**73.1**	**66.7**	**59.6**	**73.1**	**69.9**	**66.8**	**88.9**	**79.3**	**69.1**	**88.9**	**84.1**	**79.0**
DD-PPO	53.6	39.8	30.1	53.6	46.8	41.2	**72.1**	52.6	39.4	**72.1**	62.6	54.8
+Naive Camera Policy	56.9	42.3	31.3	56.9	49.6	43.4	71.7	52.4	38.8	71.7	62.1	54.5
+Ours Camera Policy	**57.7**	**44.9**	**33.9**	**57.7**	**51.3**	**45.3**	71.9	**54.6**	**40.5**	71.9	**63.3**	**55.3**

G Selection of time step budget for evaluation

Exploring the environment and locating objects efficiently are important abilities for multi-object navigation. We propose EXPO camera policy to help agents coordinate their camera and navigation actions for exploring the environment more efficiently. To better evaluate the efficiency, we evaluate the navigation success rate given a limited time step budget. If given an infinite time step budget for an agent, it is trivial for the agent to transverse the entire environment and then go to goal objects.

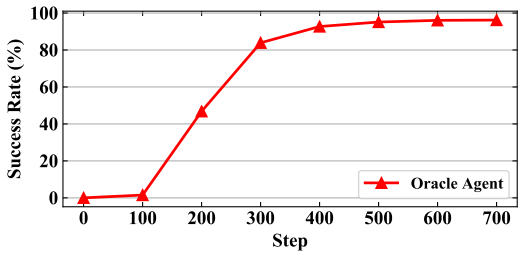

Figure B: Success rate of multi-object navigation task for an oracle agent. The success rate reaches 97% at 500 time steps and then remains the same.

To determine the value of time step budget, we consider an oracle agent with a ground-truth occupancy and object map. In other words, the oracle agent knows the position of objects and thus it does not need to explore the environment and find them. We evaluate how many time steps the oracle agent need for finishing the multi-object navigation task. In Figure B, the oracle agent successfully finishes more than 97% episodes using 500 time steps and the success rate remains the same when the time step is greater than 500. According to these results, we select 500 as the time step budget for evaluation because 500 time steps are sufficient for an oracle agent to navigate to all goal objects. We hope our EXPO camera policy helps the agents without map knowledge narrow the performance gap to the oracle agent.

H More discussions on oracle information used in experiments

In our experiments, to indicate whether an agent has reached goal objects, we use an oracle FOUND action (*i.e.,* automatically execute FOUND action once the agent reaches goal objects). We argue that this simplification is reasonable because the agent can easily judge whether a goal object appears near it with the help of a well-trained semantic segmentation model and a depth image. After simplifying this problem by using the oracle FOUND action, we are allowed to focus more on evaluating the exploration ability of active-camera agents.

I Failure cases analysis

We analyze failure cases where our active-camera agent takes more time for reaching goal objects compared with baselines. We observe that most of these cases are because our agent selects the wrong way for exploration. This is inevitable for all agents because goal objects are placed randomly in our experiments. We believe this problem can be solved if we have more information about the object's location (*e.g.,* finding a specific object like food, which is likely located in the kitchen). We can involve some constraints to guide our agent to explore relevant areas actively. We leave this in future works. Even so, as seen in Figure 3 in the paper, given the same time step, our agent explores more areas and finds all objects more quickly, which demonstrates its superior exploration ability. We strongly recommend readers watch the supplementary video for more examples.

J Potential future researches and social impact

In this paper, we present an active-camera agent. This agent can explore the environment and finish multi-object navigation task more efficiently by dynamically moving RGB-D camera sensors. In the real world, we may equip robots with different sensors, such as microphones and cameras. How to coordinate the movement of these sensors is worth exploring. Besides, we have assumed perfect camera pose localization in our experiments. However, there exist actuation and sensor noise in the real world. Leveraging a neural network [2] to solve this problem is an interesting research direction. Besides, even though we have evaluated the transferability of camera policy among different datasets (transferring from MatterPort3D to Gibson) in our paper, evaluating the robustness of the camera policy among different robot types and different embodied tasks is also a valuable research direction.

In the future, these agents may be able to help people deliver parcels and even take care of patients. On the other hand, an imperfect robotic agent may cause accidents such as breaking some things and hitting pedestrians. A possible way to avoid these accidents is to first develop such a robotic agent in a simulator environment and then deploy it into a controlled real environment for evaluation.

References

[1] A. X. Chang, A. Dai, T. A. Funkhouser, M. Halber, M. Nießner, M. Savva, S. Song, A. Zeng, and Y. Zhang. Matterport3d: Learning from RGB-D data in indoor environments. In *3DV*, pages 667–676, 2017. 4

[2] D. S. Chaplot, D. Gandhi, S. Gupta, A. Gupta, and R. Salakhutdinov. Learning to explore using active neural SLAM. In *ICLR*, 2020. 4, 6

[3] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. In *NeurIPS*, pages 2017–2025, 2015. 3

[4] S. K. Ramakrishnan, Z. Al-Halah, and K. Grauman. Occupancy anticipation for efficient exploration and navigation. In *ECCV*, pages 400–418, 2020. 2

[5] M. Savva, J. Malik, D. Parikh, D. Batra, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, and V. Koltun. Habitat: A platform for embodied AI research. In *ICCV*, pages 9338–9346, 2019. 4

[6] S. Wani, S. Patel, U. Jain, A. X. Chang, and M. Savva. Multion: Benchmarking semantic map memory using multi-object navigation. In *NeurIPS*, 2020. 2, 3, 4

[7] E. Wijmans, A. Kadian, A. Morcos, S. Lee, I. Essa, D. Parikh, M. Savva, and D. Batra. DD-PPO: learning near-perfect pointgoal navigators from 2.5 billion frames. In *ICLR*, 2020. 2, 3

[8] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese. Gibson env: Real-world perception for embodied agents. In *CVPR*, pages 9068–9079, 2018. 4

[9] B. Yamauchi. A frontier-based approach for autonomous exploration. In *CIRA*, pages 146–151, 1997. 2