Visual Graph Memory with Unsupervised Representation for Visual Navigation: Supplementary Material

Obin Kwon Nuri Kim† Yunho Choi† Hwiyeon Yoo† Jeongho Park† Songhwai Oh

Department of Electrical and Computer Engineering, ASRI, Seoul National University *

{firstname.lastname}@rllab.snu.ac.kr, songhwai@snu.ac.kr

1. Introduction

We provide additional analyses on the proposed method in this supplementary material. Section 2 presents ablation study results about network architecture, auxiliary tasks, representation learning, scalability and dataset sizes. We also provide experimental results on coverage task. Additional qualitative Results of the proposed VGM are presented in Section 4. We present examples of the generated graphs using the VGM and comparisons with other baselines. Discussion about real-world implementation is presented in Section 3. Furthermore, examples of navigation episodes and failed cases are also provided in Section 4. In Section 5, we provide the implementation details of the proposed method and other baselines.

2. Ablation Study

Network Architecture. The navigation module consists of a single encoder (F_{enc}) and two decoders (F_{dec1}, F_{dec2}). We ablated each element in the navigation module, and the results are shown in Table 1. All models are tested with hard-level episodes (5m $\sim$ 10m).

First, we examined the influence of the F_{enc} in the encoder part. Comparing the *'No Encoder'* model and GCN models, we can validate that the memory encoder makes useful information by processing VGM. The success rate slightly increases from K=1 to K=3, but larger K does not yield a considerable improvement. As K increases, the neighborhood from which a single node can aggregate information gets larger. For example, if K = 5, a single node aggregates the embeddings from the neighborhoods within 5-hop distance. As K increases, the amount of information gathered to update a single node would increase, which may dilute the local feature of the node.

*This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2019-0-01190, [SW Star Lab] Robot Learning: Efficient, Safe, and Socially-Acceptable Machine Learning). † These authors contributed equally to this work.

	Ablation	SR	SPL
Encoder	**No Encoder**	0.562	0.436
	GCN K = 1	0.588	0.468
	GCN K = 3	0.609	0.456
	GCN K = 5	0.576	0.452
Decoder	**w/o F_{dec1} (w/o c_t)**	0.581	0.459
	w/o F_{dec2} (w/o c_{target})	0.551	0.427
	shared decoder	0.585	0.433
Navigation	**w/o r_t**	0.568	0.455
	w/o F_{vis}	0.000	0.000
	Proposed Model	0.609	0.456

Table 1: **Network architecture ablation results.** We ablated each element in the navigation module of the proposed navigation framework. (**SR**: success rate, **SPL**: success weighted by path length)

For the decoder part, we ablated each decoder function (F_{dec1}, F_{dec2}) to see the effect of the context vectors on the navigation agent. *'w/o F_{dec1}'* model uses embedding vector r_t from F_{vis} instead of the context vector c_t. The difference between r_t and c_t is that c_t includes aggregated information from the memory and r_t only contains the image information from the current observation. Similarly, *'w/o F_{dec2}'* model uses r_{target} instead of c_{target}. *'Shared F_{dec}'* model uses a single decoder network F_{dec} for both c_t, c_{target}. Comparing the results of *'w/o F_{dec1}'*, *'w/o F_{dec2}'* models and the proposed model, we can see that both context vectors are helpful to achieve better navigation performance. We find that the influence of c_{target} (F_{dec2}) is larger than c_t (F_{dec1}). The aggregated information about the target observation is more useful than the one about the current observation.

We further ablated the elements in the navigation module other than the memory processor. The navigation module has a CNN F_{vis}, which encodes the current and target observation image into the embedding vectors (r_t, r_{target}). This network is different from the F_{loc} in the Memory Module. We replaced the F_{vis} with F_{loc} for *'w/o F_{vis}'* model to use e_t and e_{target} instead of r_t and r_{target}. The context vec-

Ablation	Learning	SR	SPL
w/o Auxiliary Tasks	IL	0.48	0.34
with Superivsed Localization	IL	**0.56**	**0.42**
Proposed Model	IL	0.53	0.39
w/o Auxiliary Tasks	IL + RL	0.59	0.46
with Superivsed Localization	IL + RL	**0.61**	**0.50**
Proposed Model	IL + RL	**0.61**	0.46

Table 2: **Auxiliary tasks and representation learning ablation results.** (**SR**: success rate, **SPL**: success weighted by path length)

tors would be the aggregated information from the e_t and e_{target} as follows:

$$\begin{aligned} c'_t &= F_{dec1}(e_t, M) \\ c'_{target} &= F_{dec2}(e_{target}f, M). \end{aligned} \tag{1}$$

As r_t is not given from F_{vis}, the navigation policy can only use the context vectors. We find that the model which does not have F_{vis} failed to learn a goal-searching policy (*w/o* F_{vis} in Table 1). We also removed only r_t from the navigation policy inputs to find out whether the reason for this failure is r_t. '*w/o* r_t' model in Table 1 has F_{vis}, and it uses the same context vectors as the original model, except that the navigation policy only uses the context vectors (c_t, c_{target}). We can see that *'w/o* r_t*'* model is able to learn the goal-searching policy well, even though the navigation policy can not directly see the embedding vector of the current observation. These ablation results show that F_{vis} and F_{loc} output very different embeddings even though they have the same architecture and same inputs. With the F_{loc}, the proposed navigation framework can localize and build a topological map about the environment. However, without the additional network F_{vis}, it failed to learn how to search for the goal in the environment or when to take a stop action. The embeddings from F_{loc} have useful information for localizing and mapping, but it seems that these embeddings do not contain navigation-related information.

Auxiliary Tasks. We provide the experiment results without the auxiliary losses in Table 2. The auxiliary tasks improve the success rate and SPL by a large margin when the models are only trained using imitation learning (about 10% in success rate, 14% in SPL). However, after the models are finetuned with reinforcement learning, the effect of the auxiliary losses becomes smaller (about 3% in success rate). We believe that the reason comes from the limited dataset of imitation learning. The model is trained with a limited number of demonstrations, so that the auxiliary tasks can be helpful to leverage the limited dataset. The auxiliary tasks are designed to teach the agent an ability to determine 1) the appropriate distance to take a stop action, and 2) whether the agent has passed the current location. When the model is training with reinforcement learning, the agent can learn to determine its state relative to the goal in the environment. With a number of trials and errors, the model without auxiliary tasks also can learn such abilities implicitly by the given reward.

Representation Learning. We also trained F_{loc} using supervised learning. We labeled the visibility and geodesic distance between each observation pairs using the 3d mesh of the simulator environment. If an observation pair is visible from each other and the geodesic distance is smaller than 3m, the label is set to 1. We trained F_{loc} to classify the observation pairs. *'with Supervised Localiztion'* model in Table 2 is the navigation agent trained with supervised F_{loc}. We can see that supervised localization further brings performance improvement from the unsupervised one. However, this improvement also becomes smaller after the models are finetuned with reinforcement learning. When the agent is only trained with imitation learning, supervised F_{loc} can provide better information than the unsupervised one. However, after the agent sufficiently interacts with the environment, the agent with unsupervised localization can learn to navigate well as the one with supervised localization.

Scalability. During the evaluation, the agent finds the target location creating 6.6 nodes on average, and the maximum number of generated nodes is 28. Even though the proposed method can summarize the environment into a compact-sized graph and finds the target efficiently, the number of nodes will increase as the agent keeps exploring the environment.

We limited the maximum number of nodes to a fixed number N to test its scalability in large environments. In this ablation test, the RL-finetuned agent is tested on 27 large environments (more than $100m^2$) with hard level episodes. These environments are not included in the training or validation splits from [9], and they are larger than the majority of the validation environments. When a new node is added and the number of nodes exceeds N, we erase the oldest node from the VGM. Table 3 shows the result. We also tested the baselines in the same environments. The limitation on the memory size decreases the success rate, but even with a fixed size of memory, the agent still can show better performance than other baselines. Note that we did not limit the memory size of the baselines. The Exp4nav model uses a whole metric map, and SMT collects all observations as a memory. Neural Planner and SPTM models use unlimited size of graph.

Dataset with limited size. We tested our method with limited dataset. Compared to the previous graph-based nav-

Baselines	SR	SPL	Memory Limit	SR	SPL
Exp4nav	0.414	0.348	**VGM** ($N_t \leq 1$)	0.482	0.299
SMT	0.489	0.356	**VGM** ($N_t \leq 5$)	0.526	0.390
Neural Planner	0.184	0.091	**VGM** ($N_t \leq 15$)	0.541	0.395
SPTM	0.121	0.079	**VGM** ($N_t \leq \infty$)	0.550	0.398

Table 3: **The evaluation results in large environments.** We tested the proposed navigation algorithm with different memory sizes. $N_t \leq N$ means that maximum number of nodes are limited to N. Other baselines are also tested in the same environments.

	Large		All	
Covered Ratio	t = 500	t = 1000	t = 500	t = 1000
Exp4Nav [3]	0.66	0.71	0.76	0.79
SMT [4]	0.74	0.78	0.84	0.86
ANS [1]	0.83	0.89	0.91	0.95
VGM (ours)	0.82	0.83	0.87	0.89

Table 4: Comparison and evaluation results of the baselines and our model.

igation method NTS[2], ours has an end-to-end structure which can utilize more data by training with RL. When our method is trained with a similar number of data as NTS (300 images per scene, only IL with 200 episodes), VGM performance drops (RL 0.61, IL 0.53 $\rightarrow$ IL 0.49) on the hard difficulty level.

Coverage Task We have also tested the proposed navigation framework on the coverage task. The experiment results are shown in Figure 4. The maximum time step of single episode is set to 1,000 and the table also shows the covered ratio with half of an episode (t=500) to see how fast each method can cover the environment. We first tested each method in large environments (with an area of $55m^2$ or higher) and extended to all environments in the validation split. We can see that the proposed method can quickly cover a large environment similar to the metric map-based methods (ANS, Exp4Nav). However, since the VGM does not use dense information such as the metric map (ANS, Exp4Nav) or all observations (SMT), the proposed method lacks meticulous coverage so the final covered ratio is lower than the ANS. The ANS model can densely cover the environment but requires relatively accurate geometric information and a high computational cost and memory to build a metric map. In contrast, our method sparsely covers the environment to build concise representation of the environment and it is more robust to environmental errors, such as pose estimation errors. Considering the trade-off, one can choose the memory model in accordance with the desired coverage performance.

3. Real-world Experiment

We think the main difficulty for the real-world experiment is RL because RL should be conducted in a strictly controlled environment for safety. However, as the proposed model shows acceptable performance with IL (Table 2), we are preparing to implement the IL model and study how to bring the RL features to the real world. On the other hand, recent papers [1, 7] have reported that direct sim-to-real transfer from Habitat Simulator is possible due to its photo-realistic feature. Therefore, we are also considering the direct implementation of the simulator-trained policy in the real world.

4. Qualitative VGM Visualization

Generated graph. Examples of generated graphs using VGM are shown in Figure 1a. In each example, each graph is generated from the same random exploration data. In Figure 1a, the trajectory of the agent is shown in the first column. The distance-based graph method uses the position of the agent. If the agent is far from all nodes in the graph, this method adds a new node. The Supervised Localization method builds a graph with supervised F_{loc}, which is trained in the ablation study (Section 2).

We can see that generated graphs are different across different methods. Comparing the Supervised Localization and the (unsupervised) VGM, the VGM can build a graph selecting the novel location in the environment as good as the supervised F_{loc}. The distance-based method only focuses on the distances between nodes. Each node in the graph is selected by comparing distances with previous nodes, not by considering how each location is novel to the previous nodes. The node distances of the VGM graph are different according to locations because VGM builds a graph based on similarities between observations. When the agent navigates through the corridor or doorway (pink area in Figure 1a), the appearance of observations will change abruptly as the agent enters a new room or encounter new furniture. In this case, new nodes will be densely added and the distances between nodes will be small. In contrast, when the agent moves in a large hallway (blue area in Figure 1a), the appearance of observations will not change much. So the distance between nodes becomes larger. In Figure 1b, we marked the positions of nodes generated through several navigation episodes in a single environment. We can see that a number of nodes are concentrated around a narrow corridor or doorway. We also visualized corresponding image observations of nodes to see which location is added as a node, in Figure 4.

We can adjust the distance between nodes by changing the value of the similarity threshold s_{th}. The visualizations of generated graphs according to various s_{th} are shown in Figure 1c. As we have stated in the paper, small s_{th} gener-

ates a sparse graph since small s_{th} associates a larger region as a neighborhood. A new node is added only when the new observation has a smaller similarity value than s_{th}.

Navigation episodes. We present further examples of navigation episodes in Figure 2a. The agent can find the target location after exploring other places. A video of these examples and comparisons with other baselines are provided in the supplementary video.

We also present examples of failed episodes in Figure 2b. The main reasons for the failure can be categorized into four types; *Not Close Enough, False Stop, Stuck on Obstacles*, and *Time over*. The agent often takes a stop action around the target but not close enough to be regarded as success (*Not Close Enough*). This type of failure happens in 14.4% of all failed episodes. The agent sometimes takes a stop action in the wrong location which looks similar to the target location (*False Stop*, 25.5%). Furthermore, the agent often gets stuck on obstacles (*Stuck on Obstacles*, 12.3%). Some obstacles are invisible to the agent due to the height of the camera and low-quality of 3D mesh. The most frequent reason for the failure is *Time Over* (44.6%), in which the agent had not found the target area after 500 steps of the navigation. The agent rarely overlooks the target location when it encounters the target area during the navigation. Most of the *Time Over* cases are when the agent had not sufficiently explored the environment to find the target area in a limited time. Adding an exploration reward in the reinforcement learning stage might address these failure cases.

5. Implementation Details

5.1. Network Architecture

An image observation is a panoramic RGBD image with $64 \times 252 \times 4$ size. The image encoders F_{loc} and F_{vis} are both based on ResNet-18 [6]. They take an image observation and produce a 512-dimension vector. F_{enc} is a graph convolutional network, which uses 512-dimension vectors from the outputs of F_{loc} and F_{vis}. F_{enc} consists of $K = 3$ graph convolutional layers. F_{dec} is multi-head attention network, which has $J = 4$ heads.

5.2. Training details

Imitation learning. We collected 200 trajectories in each environment for training. As we want the agent to learn how to use VGM, demonstrations need to contain sufficient information to build a nontrivial VGM. We have collected demonstrations from the oracle agent, which sequentially searches for multiple targets in order. In the learning stage, the agent builds a VGM from the start of a demonstration. This memory remains during the whole demonstration sequence, even when the target changes. As the oracle agent in the demonstration data sufficiently roamed the environment to build a decent size of memory, we intended the agent to learn how to use the built memory for inferring the proper action to find the targets. Using this kind of demonstration, we could derive successful behavior of the agent, which is better than the one only trained with single-target search demonstration. Each target is separated from 3m to 10m, and up to 5 targets are sampled at the start of each trajectory. At each training iteration, we sampled a mini-batch of 16 training trajectories, and update the network parameters using the Adam optimizer with learning rate 0.0001, weight decay 0.00001. Early stopping is used for all imitation training models.

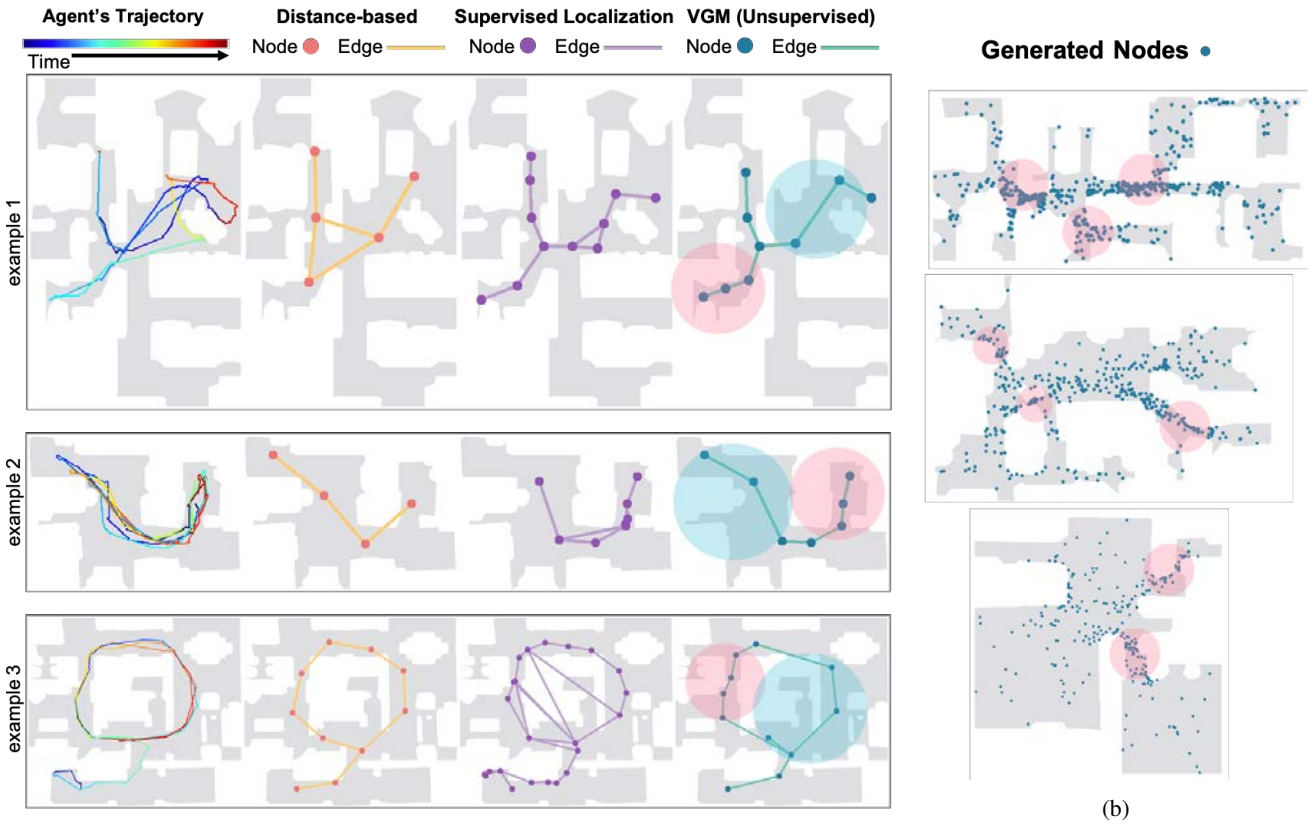
Reinforcement learning. In the reinforcement learning stage, we used Proximal Policy Optimization (PPO) [11]. Hyperparameter settings are shown in Table 5. All the end-to-end policy baselines (CNN+LSTM, Exp4Nav and SMT) and our proposed model are first trained using imitation learning and finetuned 10M frames in the reinforcement learning stage. ANS model is also trained with 10M frames. Handcrafted graph baseline algorithms (Neural Planner, SPTM) need local policies to navigate between the nodes. For the Neural Planner, we used the publicly released pre-trained point-goal navigation model from [12], which shows 0.944 SPL in the Gibson dataset. For the SPTM, we utilized the RL-finetuned CNN + LSTM model as the image-based local navigation policy. For the exploration policy of the graph baselines, we used the VGM coverage model as it shows better performance than the basic CNN+LSTM model.

Representation learning. The image encoder F_{loc} in memory update module is trained using prototypical contrastive learning (PCL) [8]. We sampled 10,000 images for each environment. For each observation image o_i, F_{loc} produces a feature embedding vector $e_i = F_{loc}(o_i)$. The loss function of PCL has the similar form as InfoNCE [5, 10], which is formulated as:

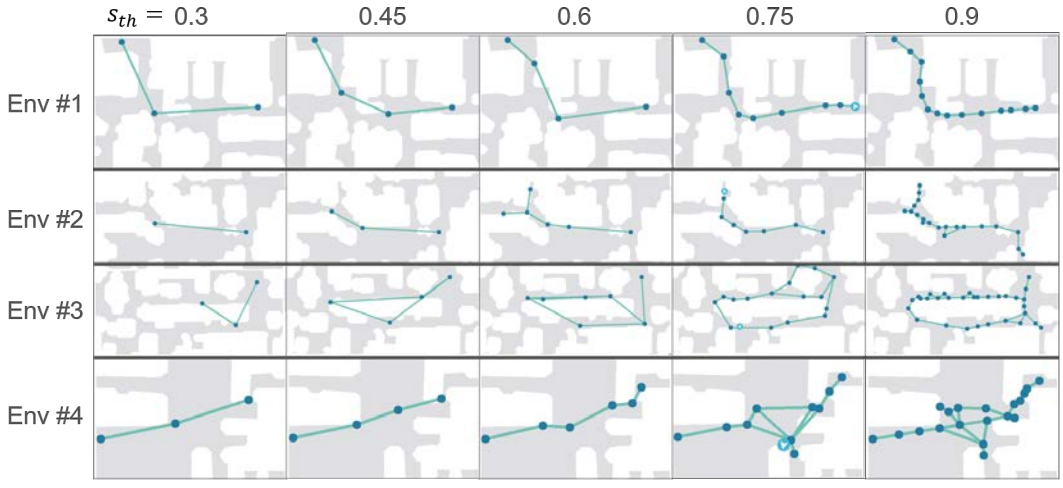
$$L_{\text{InfoNCE}}(e_i) = -\log \frac{\exp(e_i \cdot e_i'/\tau)}{\sum_{j=0}^{r} \exp(e_i \cdot e_j'/\tau)}, \tag{2}$$

where e_i' is the positive embedding of e_i, e_j' includes r negative embeddings, and τ is a temperature hyperparameter. A positive embedding is from an augmented version of o_i, and negative embeddings are from different images. We augment the panoramic image by vertically splitting the image into 12 patches and changing the order of the 12 patches. *e.g.*, $\{0, 1, 2, ...10, 11\} \longrightarrow \{3, 4, 5, ...11, 0, 1, 2\}$

PCL has an additional contrastive loss using prototypes of image embeddings. PCL iteratively conducts two steps which are similar to the expectation maximization (EM) algorithm. In the first step, we conduct k-means clustering for the image embeddings from the training dataset. Then,

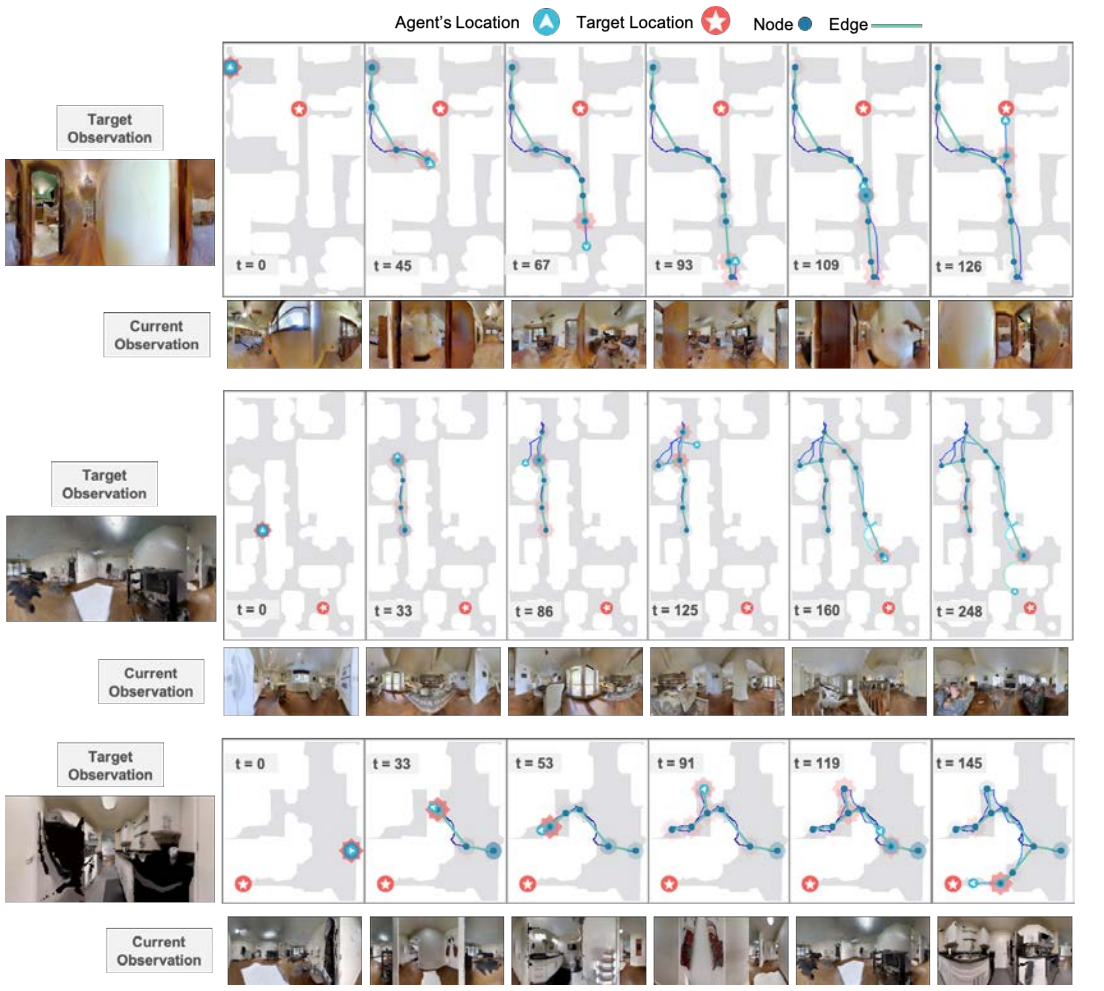


(a) (b)

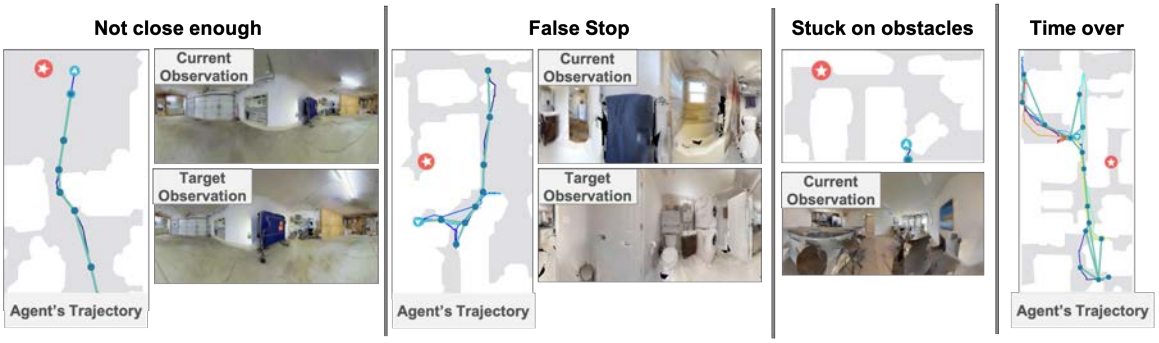


(c)

Figure 1: **(a) Examples of generated graphs.** We compare the graphs generated using various methods. The first column shows trajectories of an agent in the map. Each graph is generated using the trajectory shown in the first column. Nodes are densely generated in a corridor or doorway (pink area) because the appearance of observations changes abruptly as the agent enters a new room or encounter new furniture. In contrast, nodes are generated sparsely in a large hallway (blue area) because the appearance of observations does not change much as the agent moves around. **(b) Visualization of all generated nodes.** We collected several navigation episodes in each environment, and visualized the generated nodes in the map. Pink areas denote the concentrated region of generated nodes. **(c) Generated graphs across various values of the similarity threshold s_{th}.** Small s_{th} generates a sparse graph since small s_{th} associate a larger region as a neighborhood.



(a)



(b)

Figure 2: **(a) More examples of a target searching episode.** Note that these maps are for visualization and the agent is not given any metric maps. Each node in the map denotes the node elements which is added throughout the path of the agent. Additionally, the attention scores of nodes are shown as blue circles and red stars. The blue circle means the attention score in the context of current observation in F_{dec1}. The red star means the attention score in the context of the goal observation in F_{dec2}. More vivid the color, the higher the attention score. Best shown in color. **(b) Examples of failed episodes.** Main reasons for the failure can be categorized into four;*Not Close Enough, False Stop, Stuck on Obstacles, Time over*. The trajectories of the agent and the observations for each category are shown in the figure.

paramter	value
batch size	128
buffer size	256*4
rollout step.	256
epoch	2
ϵ (clip ratio)	0.2
learning rate	0.00001
value loss coefficient(c_1)	0.5
entropy coefficient (c_2)	0.3
gamma (generalized advantage estimation)	0.99
tau (generalized advantage estimation)	0.95

Table 5: Hyperparameters for PPO training.

the centroids of k clusters are set as prototypes $\{c_i\}_{i=1}^{k}$. We cluster the embeddings $M = 3$ times for different numbers of k to encourage various resolutions of clusters. In the second step, the network parameters are updated in the direction that each e_i gets closer to its own cluster c_s. The illustrated overview of the second step is shown in Figure 3a The prototypical contrastive loss is formulated as :

$$L_{\text{cluster}}(e_i) = -\frac{1}{M}\sum_{m=1}^{M}\log\frac{\exp(e_i \cdot c_s^{k_m}/\phi_s^{k_m})}{\sum_{j=0}^{r}\exp(e_i \cdot c_j^{k_m}/\phi_j^{k_m})}, \quad (3)$$

where c_s is the prototype of the cluster which o_i belongs to, $\{c_j\}$ includes r negative prototypes from other clusters and c_s. The superscript k_m means that each cluster is from k_m-means clustering. In our setting, we use $k_m \in \{2500, 5000, 10000\}$. In addition, PCL uses estimated concentrated level ϕ_s as a temperature hyperparameter. ϕ_s is calculated as the variance of embeddings in each cluster. The total loss of PCL is set as follows:

$$L_{\text{ProtoNCE}} = \sum_{i=1}^{n} L_{\text{InfoNCE}}(e_i) + L_{\text{cluster}}(e_i) \quad (4)$$

where n is the total number of images in a mini-batch. We used τ in $L_{\text{InfoNCE}}(e_i)$ as 0.2, the batch size is 1024, and the learning rate is 0.3.

We visualize the learned representation space of the trained F_{loc} via t-SNE in Figure 3b. We collected the features of the validation image set using F_{loc}. The observation images in the validation image set are sampled from the unseen environment. Each point in the top-down map represents the feature embedding of the observation in the specific location. We can see that the observations from the similar area has similar representations while each area is distinct from other areas. We also clustered features to see the distribution of the clusters that PCL learned. Each colored area in the map denotes that the observation images in the region are assigned to the same cluster (see Figure 3b). We can see that the observations from the similar area are assigned to same cluster, and they are distinct from the other clusters.

References

[1] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning To Explore Using Active Neural SLAM. In *International Conference on Learning Representations (ICLR)*, 2020. 3

[2] Devendra Singh Chaplot, Ruslan Salakhutdinov, Abhinav Gupta, and Saurabh Gupta. Neural Topological SLAM for Visual Navigation. In *IEEE Conference on Compuuter Vision and Pattern Recognition (CVPR)*, 2020. 3

[3] Tao Chen, Saurabh Gupta, and Abhinav Gupta. Learning Exploration Policies for Navigation. In *International Conference on Learning Representations (ICLR)*, 2019. 3

[4] Kuan Fang, Alexander Toshev, Li Fei-Fei, and Silvio Savarese. Scene Memory Transformer for Embodied Agents in Long-Horizon Tasks. *IEEE Conference on Compuuter Vision and Pattern Recognition (CVPR)*, 2019. 3

[5] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum Contrast for Unsupervised Visual Representation Learning. In *IEEE Conference on Compuuter Vision and Pattern Recognition (CVPR)*, 2020. 4

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *IEEE Conference on Compuuter Vision and Pattern Recognition (CVPR)*, 2016. 4

[7] Abhishek Kadian and et al. Sim2real predictivity: Does evaluation in simulation predict real-world performance? *IEEE Robotics and Automation Letters (RA-L)*, 2020. 3

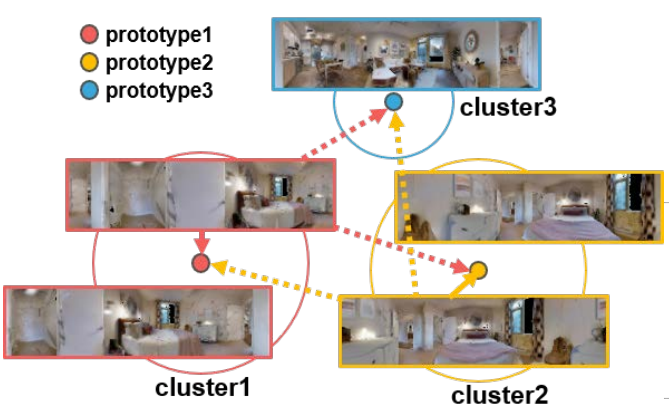
[8] Junnan Li, Pan Zhou, Caiming Xiong, Richard Socher, and Steven C.H. Hoi. Prototypical Contrastive Learning of Unsupervised Representations. In *International Conference on Learning Representations (ICLR)*, 2021. 4

[9] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A Platform for Embodied AI Research. In *IEEE International Conference on Computer Vision (ICCV)*, 2019. 2

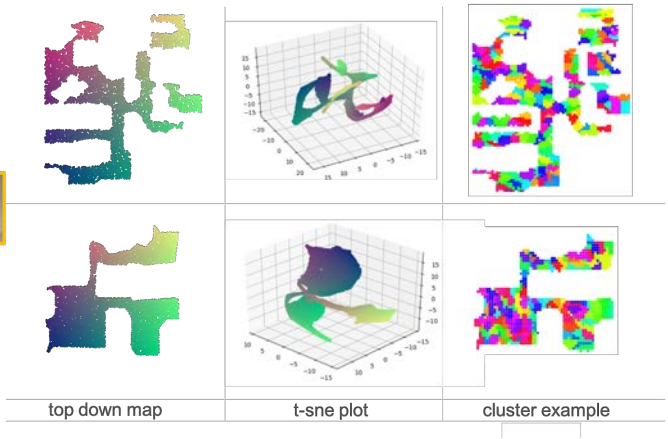
[10] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation Learning with Contrastive Predictive Coding. *arXiv preprint arXiv:1807.03748*, 2018. 4

[11] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 4

[12] Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. DD-PPO: Learning near-perfect pointgoal navigators from 2.5 billion frames. *International Conference on Learning Representations (ICLR)*, 2020. 4



(a) Overview of the PCL



(b) Visualization of the learned representation space

Figure 3: **(a) Overview of the PCL** In PCL, an image encoder is trained to encode an image similar to the prototype of its own cluster while the prototypes of other clusters are placed further. **(b) Visualization of the learned representation space** Each point in the top-down map represents the observation taken in the location. We visualize the representation space via t-SNE plot. We also clustered the features and visualized them on the map. Each colored area in the map denotes that the observation images in the area are assigned to the same cluster. Best shown in color.

Figure 4: **Corresponding Observations of Generated Nodes.** The graph is generated from random exploration data. We visualized corresponding image observation on each node. Even with the unsupervised representation, the generated graph well summarized the environment into a compact size.