Supplementary Material for Topological Semantic Graph Memory for Image-Goal Navigation

Nuri Kim, Obin Kwon, Hwiyeon Yoo, Yunho Choi, Jeongho Park, and Songhwai Oh
Department of Electrical and Computer Engineering, ASRI, Seoul National University
{firstname.secondname}@rllab.snu.ac.kr, songhwai@snu.ac.kr

1 Training Details and Experimental Settings

Training process. The training process is as follows: 1. The agent builds a graph map, which can be incrementally updated based on the observation. 2. Extract latent contextual representation from memory generated from a graph map using the attention method. 3. Extract the action by putting the attended memory, current observation, and prior action into the action policy function. 4. The action is first optimized with imitation learning with oracle action, and then the agent learns more suited for the environment through reinforcement learning.

Implementation details. Baselines and the proposed method, excluding the methods that do not use training [1], are trained using imitation learning with 14,400 episodes, 200 episodes per scene for 72 training scenes. Then, reinforcement learning is applied for 10M frames to fix erroneous behaviors by interacting with simulation environments. The cross-update is carried out twice for both graphs since we wanted to update nodes using updated nodes while utilizing the least computational costs.

Real robot experiment settings. We implement TSGM on a Clearpath Jackal platform for a real-world setting. Jackal is equipped with a single Ricoh Theta V camera, a 360° RGB sensor. Since this camera collects images with equirectangular projections, the equirectangular image is converted into a panoramic image of the 12 pinhole cameras having 30° FoV which is used for training. Since it was found that getting the exact panoramic depth is challenging, depth was extracted from the pretrained depth estimator, using the Omni-Depth [2] algorithm for the real-world experiments. Omni-Depth [2] estimates depth from panoramic RGB images gathered from indoor datasets such as Matterport dataset [3]. We utilized the same test simulator environment for training and testing the depth estimator since we assumed a very accurate depth input was given. To control the Jackal, we use a PD controller.

Figure 1: Clearpath Jackal

Episode settings. Gibson dataset is divided into 72 train scenes and 14 test scenes for image goal navigation. The test episodes in the Gibson dataset are derived from 14 test scenes that do not overlap the training scene. The test episodes for Table 1 of the paper are proposed in VGM [4]. The number of episodes for each difficulty is 1,007, totaling 3,021 episodes. Since episodes with similar beginning and end points might be chosen as duplicates if the scene is tiny, the number of test episodes was chosen in proportion to the scene's size.

The test episodes used in Table 2 of the paper are proposed in NRNS [1], which are divided into two types: straight and curved. The ratio of shortest path geodesic-distance to euclidean-distance between the starting and target locations in straight episodes is 1:2, and the rotational difference between the start position and destination is 45°. All other start-goal location pairs are labeled as

6th Conference on Robot Learning (CoRL 2022), Auckland, New Zealand.

curved episodes. There are 2,806 straight episodes and 3,000 curved episodes in total. Similar to the episode configuration used in Table 1, there are around 1,000 items divided according to the difficulty level.

Similarity thresholds. Two pretrained encoders are used to determine whether to add or update the graph with a new observation when building TSGM. We adopt PCL [5] to obtain image similarity and Supervised contrastive learning [6] d to train object similarity. We apply the pretrained PCL [5] model and set the image similarity threshold (τ_i) to 0.75. We trained the model [6] for object similarity utilizing provided identity values for objects on Gibson train environments as labels. After training, the object threshold is chosen to distinguish objects with the greatest margin of error when tested on the Gibson test scene. To evaluate the trained object encoder, we apply clustering metrics, normalized mutual information (NMI) and purity. NMI is calculated by taking the mutual information between the true object identity distribution and the estimated distribution and then normalizing the value by the self-information of the two distributions. Furthermore, purity is defined as the number of majority positives divided by the total number of samples. With 0.839 NMI and 0.878 purity, the trained object encoder performed well in predicting object identifications on the clustering measures. As a result, the object threshold (τ_o) is set as 0.8 for all experiments.

2 Graph Memory Construction

When an agent is placed in an unknown environment, the agent explores the environment to acquire a graph memory. The graph memory is constructed using two pretrained encoders: an image encoder enc_{im} and object encoder enc_{obj}. To improve the graph with common objects, we add a rule that prevents adding a new image node if the ratio of the number of common objects of the two image nodes is larger than 10%,

$$\text{CO}(x_m, x_t) = \frac{\#(O(x_m) \cap O(x_t))}{\#(O(x_m) \cup O(x_t))} > 10\%, \tag{1}$$

where $O(x_m)$ represents the objects connected to the image node x_m. The detailed algorithm for building a topological semantic graph is described in Algorithm 1.

Object node. Object features are retrieved using ROI-align [7] based on the position of the object bounding box. The object features are extracted from the output of the conv4 layer of ResNet18. The object category from the detector is encoded with two-layered neural networks and then concatenated to the object features to make an object node state $z_k = \langle \mathcal{F}^o_k, c_k, r_k \rangle$ for kth object. $\mathcal{F}^o_k$ is the extracted object feature, c_k represents the category of the object, and r_k is the detection score of the object. The number of categories is 80 since we employ a detector that is pretrained on the COCO dataset.

Object node updating rules. The graph builder compares the object's detection scores when the same object node is found. If a new input object's feature is similar to an object in memory and the category is the same, the graph builder determines that the two objects are identical. The graph builder compares the detection scores to increase the detected features' quality. Only when the newly discovered object has a higher score, the graph builder updates the node feature.

Graph connections. While building the graph, the affinity matrices can be calculated based on the edge information,

$$\begin{aligned} A_{im}[x_v, x_w] &= 1 \; \forall (x_v, x_w) \in \mathcal{E}_{im}, \\ A_c[x_v, z_k] &= 1 \; \forall (x_v, z_k) \in \mathcal{E}_c. \end{aligned} \tag{2}$$

To make objects in proximity connected in the graph memory, we calculate the affinity matrix for object nodes to be connected to objects in the neighbors of the current image node using the image affinity A_{im} and image-object affinity. The object graph affinity matrix can be calculated using the above matrices,

$$A_{ob} = A_c^T (A_{im} + \mathrm{I}) A_c, \tag{3}$$

where I is an identity matrix, which connects the object nodes that share the same image node.

Algorithm 1: Build Topological Semantic Graph Memory

Input: Sequence of the input images, $\mathcal{B}$
Output: Topological semantic graph, $\mathcal{G}$

Initialize empty vertex set, $\mathcal{V}_{im} \leftarrow \varnothing$, $\mathcal{V}_{ob} \leftarrow \varnothing$
Detect objects from the input image, $\mathbf{z}_1 \leftarrow$ Detector(I_1)
Initialize the last localized image, $I_l \leftarrow I_1$
Initialize the neighboring objects $O_l \leftarrow \mathbf{z}_1$
Let $\mathcal{F}_l^i \leftarrow enc_{im}(I_l)$; $x_l \leftarrow \langle \mathcal{F}_l^i \rangle$
for $I_t \in \mathcal{B}$, each observation in the batch, **do**
    Encode image, $\mathcal{F}_t^i \leftarrow enc_{im}(I_t)$; $x_t \leftarrow \langle \mathcal{F}_t^i \rangle$
    if current place is where an agent already explored, i.e., $\exists x_m \in \mathcal{V}_{im}$ s.t. $cos\_dist(\mathcal{F}_m^i, \mathcal{F}_t^i) \geq \tau_i$ and CO(x_m, x_t) $> 10\%$, **then**
        Update the most similar image node in the memory using new feature:
        $k \leftarrow \text{argmax}_m cos\_dist(\mathcal{F}_m^i, \mathcal{F}_t^i)$
        $\mathcal{F}_k^i \leftarrow \mathcal{F}_t^i$
        Connect the kth node to the last localized node:
        $\mathcal{E}_{im} \leftarrow \mathcal{E}_{im} \cup (x_k, x_l)$
    end
    else if I_t is new, i.e., $cos\_dist(\mathcal{F}_l^i, \mathcal{F}_t^i) \leq \tau_i$ or t=0, **then**
        Add x_t, which is $\langle \mathcal{F}_t^i \rangle$ to the graph $\mathcal{G}$:
        $\mathcal{V}_{im} \leftarrow \mathcal{V}_{im} \cup \{x_t\}$
        $\mathcal{E}_{im} \leftarrow \mathcal{E}_{im} \cup (x_t, x_l)$
        Update the last localized image node:
        $I_l \leftarrow I_t$
        Detect objects from the input image, $\mathbf{z}_t \leftarrow$ Detector(I_t)
        Sort objects by detection scores.
        for $\forall b_k \in \mathbf{z}_t$, each object in the current observation, **do**
            Encode objects, $\mathcal{F}_k^o \leftarrow enc_{obj}(b_k)$
            if b_k exists in the neighboring objects O_l, i.e., $\exists z_m \in O_l$ s.t. $cos\_dist(\mathcal{F}_m^o, \mathcal{F}_k^o) \geq \tau_o$ and $c_k = c_m$ and $r_k > r_m$ **then**
                Update the most similar object node in the O_l using new feature:
                $k \leftarrow \text{argmax}_m cos\_dist(\mathcal{F}_m^o, \mathcal{F}_t^o)$
                $\mathcal{F}_k^o \leftarrow \mathcal{F}_t^o$
            end
            else if Input object is new according to the neighboring objects, i.e., $cos\_dist(\mathcal{F}_m^o, \mathcal{F}_k^o) \leq \tau_o, \forall b_m \in \hat{\mathbf{b}}_n$ or $c_k \neq c_m$, **then**
                Add the object state z_k, which is $\langle \mathcal{F}_k^o, c_k, r_k \rangle$ to the graph $\mathcal{G}$:
                $\mathcal{V}_{ob} \leftarrow \mathcal{V}_{ob} \cup \{z_k\}$
                $\mathcal{E}_c \leftarrow \mathcal{E}_c \cup (x_l, z_k)$
            end
        end
        $O_l \leftarrow \langle \mathbf{z}_t, \text{Neighbor}(\mathbf{z}_t) \rangle$
    end
end
$\mathcal{G} \leftarrow \{\mathcal{V}_{im}, \mathcal{V}_{ob}, \mathcal{E}_{im}, \mathcal{E}_c\}$
return $\mathcal{G}$

3 Cross Graph Mixer

The cross graph mixer module runs for L steps and is defined in terms of a message function M and vertex update function U. The message function can be seen as a composition of two functions, $M = C \circ S$, where S is the self-update function, and C is the cross-update function. At first, the

image memory state x and object memory state z are the inputs of the network,

$$\begin{aligned} hi_v^1 &= x_v, \forall v \in \mathcal{V}_{im} \\ ho_k^1 &= z_k, \forall k \in \mathcal{V}_{ob}. \end{aligned} \tag{4}$$

For lth step, where $l \in \{1, ..., L\}$, image and object node states are self-updated to get contextual representations between nearby locations or objects,

$$\begin{aligned} \hat{mi}_v^l &= \sum_{w \in \mathcal{N}_i(v)} S_i^l(hi_w^l, A_{im}, g), \\ \hat{mo}_v^l &= \sum_{k \in \mathcal{N}_o(v)} S_o^l(ho_k^l, A_{ob}, g), \end{aligned} \tag{5}$$

where $\mathcal{N}_i(v)$ and $\mathcal{N}_o(v)$ denote the image/object neighbors of the vth node, respectively. The self-update message function S makes a d dimensional vector by aggregating connected nodes, i.e.,

$$\begin{aligned} S_i^l(hi_w, A_{im}, g) &= \mathrm{A}_{im}[v, w]\mathrm{B}_i^l(e_{vw}) f_i(hi_w^l || g) \\ S_o^l(ho_k, A_{ob}, g) &= \mathrm{A}_{ob}[v, k]\mathrm{B}_o^l(e_{vk}) f_o(ho_k^l || g), \end{aligned} \tag{6}$$

where $||$ is a concatenation operation and f_i and f_o are two-layered neural networks. Here, $\mathrm{B}(e_{vw})$ is a learned connection relationship between vth and wth nodes, i.e., the edge vectors between the nodes, which maps the edge vector e_{vw} to a $d_i \times d_i$ matrix,

$$\begin{aligned} \mathrm{B}_i^l(e_{vw}) &= \mathbf{v}_i^T \mathrm{concat}(\mathbf{W}_i hi_v^l, \mathbf{W}_i hi_v^l) \\ \mathrm{B}_o^l(e_{vk}) &= \mathbf{v}_o^T \mathrm{concat}(\mathbf{W}_i ho_v^l, \mathbf{W}_o ho_k^l), \end{aligned} \tag{7}$$

where $\mathbf{W}_i \in \mathbb{R}^{C \times d_i}$, $\mathbf{v}_i \in \mathbb{R}^{2d_i \times d_i}$, $\mathbf{W}_o \in \mathbb{R}^{C \times d_o}$, $\mathbf{v}_o \in \mathbb{R}^{2d_o \times d_o \times d_o}$ are the matrix parameters, which are applied for all latent nodes.

Then, the nodes aggregate the different types of nodes during the message passing phase to make a complete message mi_v^l and mo_v^l. For this, nodes cross update is done for image nodes to aggregate messages from object nodes, while object nodes use image nodes to create messages,

$$\begin{aligned} mi_v^l &= \sum_{k \in \mathcal{N}_o(v)} C_i^l(\hat{mi}_v^l, \hat{mo}_k^l, A_c), \\ mo_v^l &= \sum_{w \in \mathcal{N}_i(v)} C_o^l(\hat{mo}_v^l, \hat{mi}_w^l, A_c), \end{aligned} \tag{8}$$

where

$$\begin{aligned} C_i^l(\hat{mi}_v^l, \hat{mo}_k^l, A_c) &= \hat{mi}_v^l + f_i'(\mathrm{A}_c[v, k]\mathrm{B}_i^l(e_{vk}) f_i''(\hat{mo}_k^l || g)), \\ C_o^l(\hat{mo}_v^l, \hat{mi}_w^l, A_c) &= \hat{mo}_v^l + f_o'(\mathrm{A}_c[w, v]\mathrm{B}_o^l(e_{wv}) f_o''(\hat{mi}_w^l || g)), \end{aligned} \tag{9}$$

where f_o', f_i', f_o'' and f_i'' are two-layered neural networks.

Finally, the update function U transfers messages from object nodes to image nodes,

$$\begin{aligned} hi_v^{l+1} &= U_i^l(hi_v^l, mo_v^l), \\ ho_v^{l+1} &= U_o^l(ho_v^l, mi_v^l), \end{aligned} \tag{10}$$

where,

$$\begin{aligned} U_i^l(hi_v^l, mo_v^{l+1}) &= hi_v^l + g_i(A_c[\cdot, v] mo_v^{l+1}), \\ U_o^l(ho_v^l, mi_v^{l+1}) &= ho_v^l + g_o(A_c[v, \cdot] mi_v^{l+1}), \end{aligned} \tag{11}$$

where g_i is a neural network that maps hidden states of an object to hidden states of the connected images and g_o is a neural network that maps hidden states of an image to hidden states of the connected objects.

After L iterations, the output of the network become a contextual memory. The output of the image stage is,

$$\begin{aligned} \mathbf{mi}_v &= hi_v^L \ \forall v \in \{1, ..., N\}, \\ \mathbf{mo}_k &= ho_k^L \ \forall k \in \{1, ..., M\}. \end{aligned} \tag{12}$$

Note that $\mathbf{mi} = \{\mathbf{mi}_1, ..., \mathbf{mi}_N\}$ is the contextual image memory and $\mathbf{mo} = \{\mathbf{mo}_1, ..., \mathbf{mo}_M\}$ is the contextual object memory.

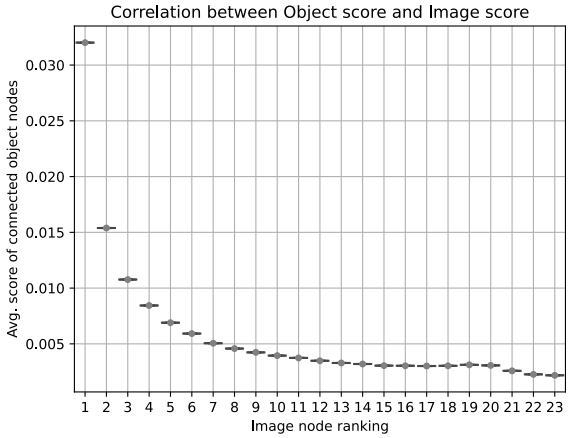

Figure 2: Correlation between the object score and image score. It indicates that the object nodes help image nodes to localize to the current node.

4 Object-Based Methods

We compared TSGM to navigational methods [8, 9] that employ object information. In VTNet [8], an image and objects observed by an agent are combined to produce a fused representation of the place. VTNet [8] does not have any explicit memory and only has an implicit RNN memory. TSGM, on the other hand, can explicitly employ previous knowledge gathered while navigating the environment. VTNet [8] connects object information when they are detected in the same image. Since it combines objects detected in the same image, VTNet [8] can be seen as a method that only connects objects detected in the same image. On the other hand, TSGM can connect neighboring object nodes even if the objects are not detected in the same image, resulting in contextually solid representations.

In Object memory transformer [9], image and object representation are saved in the explicit memory every time step. Since TSGM only puts a new node into a graph memory based on the similarity between memory and current observations for both image and object graphs, it has less redundancy than [9]. [9] utilizes only the preceding T chunks of data are utilized. TSGM, on the other hand, makes use of all graph memory information derived from past exploration of the environment. This is achievable due to TSGM's low redundancy.

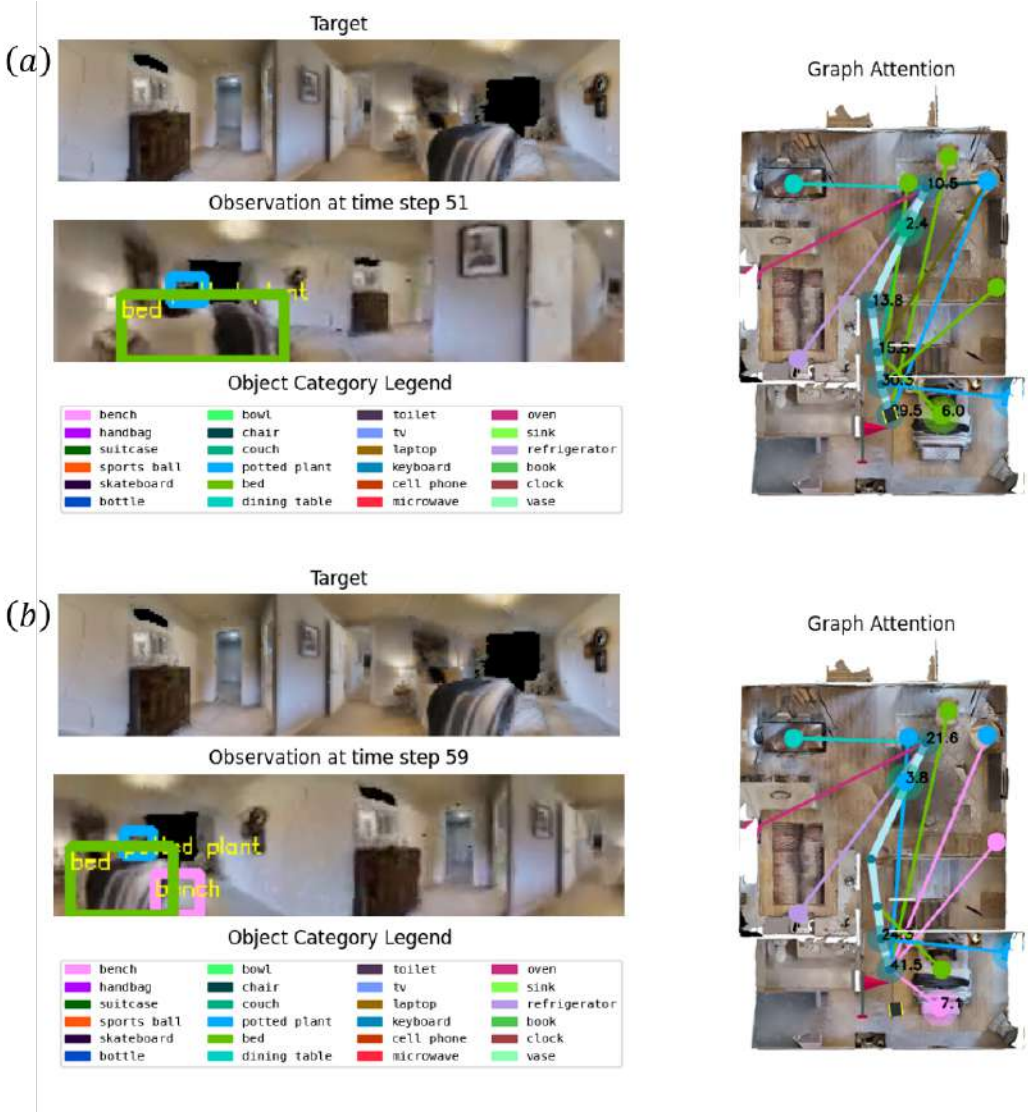

Figure 3: Attention scores of the image and object nodes.

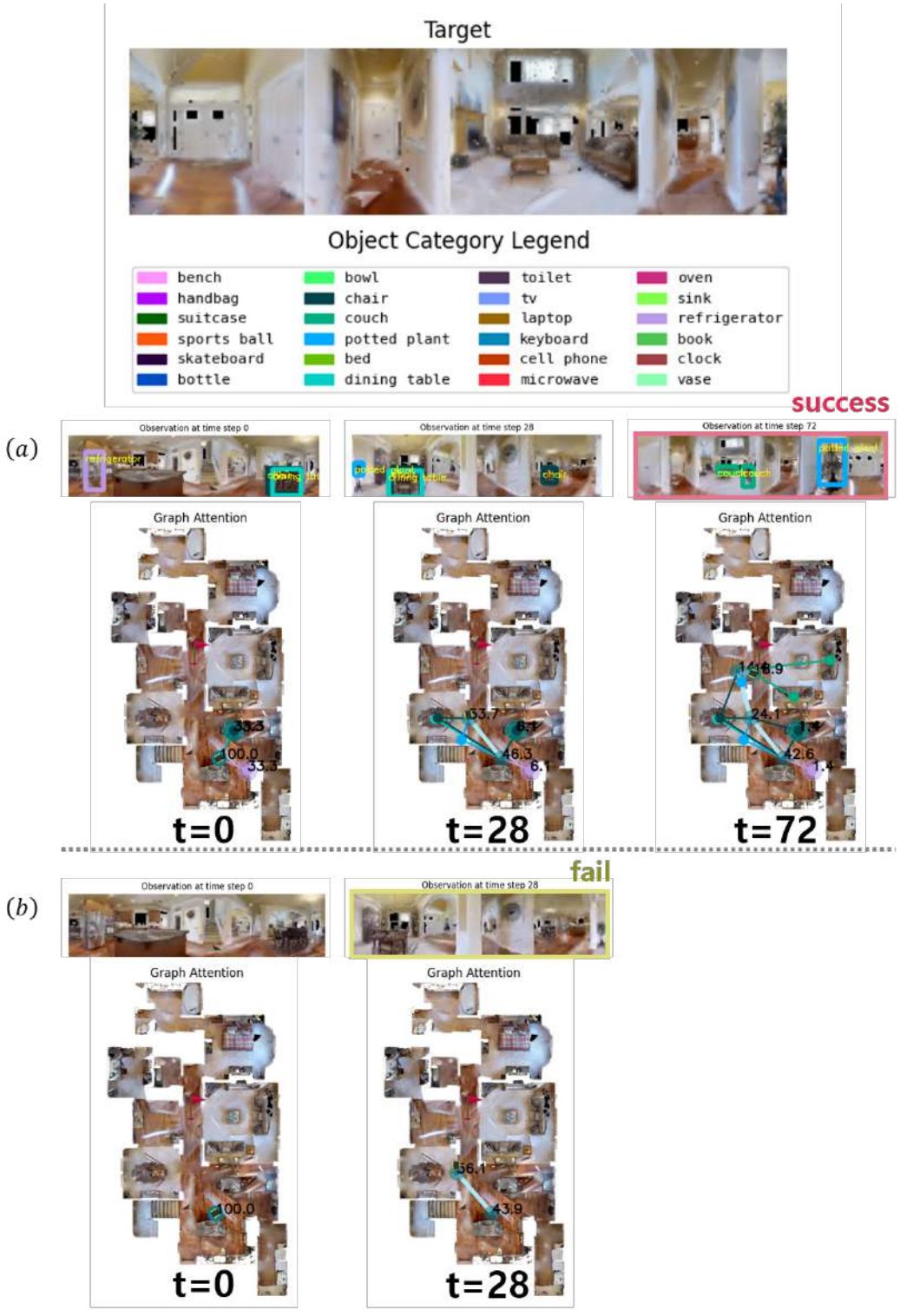

Figure 4: Comparison with the method without objects. The stop button can be pressed more precisely through the object context.

Figure 5: Comparison with the method without objects. The agent can utilize object configurations to get the semantic knowledge how to go from the bathroom to living room.

5 Impact of Landmarks

Object helps localization. To demonstrate the influence of object information on performance improvement, we conducted additional experiments using attention values from the memory attention module (Section 3.4). The agent draws the best image memory in the memory attention module given the current observation. We assume that localization is successful if the selected image node, which has the highest attention score, is the closest to the agent location. The success rate of localization is computed using the 1,007 hard episodes, which are used in Table 1 of the manuscript. As a result, localization performance is 38.5% without an object node whereas it is 68.2% with an object node, an improvement of 29.7%.

Object and image are correlated. To demonstrate that incorporating objects assists in localization, we investigate the correlation between image and object nodes. We averaged the obtained attention values across the objects in an image since it contains many objects. The vertical axis in Figure 2 reflects the average attention score of the object nodes connected with the corresponding ranking's image node. Since the attention score tends to diminish as the number of image nodes rises, the image node ranking was employed as the horizontal axis by sorting in descending order using the attention value. As a consequence, the image node selected as the current node demonstrated a substantial association with a higher object node score (Figure 2). The likelihood of being in the adjacent image node increases as the object node score increases, suggesting that the inclusion of object nodes benefited with agent localization. As a consequence, the image node selected as the current node demonstrated a substantial association with a higher object node score. In summary, it is claimed that the utilization of object nodes enables the agent to search the path more efficiently while avoiding getting lost by properly localizing the agent.

Visualization of the attention score. We showed the attention scores of image and object nodes to demonstrate the significance of object features. To use a ground truth detector, we finetuned TSGM using PPO for 2M frames in the Gibson tiny dataset for this experiment. For the comparison experiments, a model without objects was also finetuned in the same setting, using PPO for 2M frames. The categories in the Gibson tiny dataset are the same as those in the COCO dataset; we displayed 24 of them. Besides the illustration, we show the legend that indicates the color of each object category. The image node is blue, whereas the object node is colored differently for each category. The attention values are multiplied by 100 and scaled to a score ranging from 0 to 100 for the best view. In Figure 3, the attention scores are depicted in Figure 3(a) at step 51 and Figure 3(b) at step 59. When the *bench* node is not visible, the nearest image node has a score of 29.5. After the *bench* node is recognized, the current node's attention score climbs to 41.5. The score of the currently located *bench* and *potted plant* is high, indicating that the current image node's localization accuracy has improved. A model in Figure 4(a) is trained using an object, whereas a model in Figure 4(b) was trained without an object. The initial paths taken by Figure 4(a) and Figure 4(b) are similar, but Figure 4(b) failed since it pressed the stop button at the incorrect place. The model in Figure 4(b) was unsuccessful since it could not localize at the goal. On the other hand, Figure 4(a) successfully hit the stop button precisely at the target location after realizing that the place at time step 28 was not the target position by using the object configuration information. Figure 5 shows the result of the model trained without an object (Figure 5(b)) and the version trained with an object (Figure 5(a)). The goal is successfully reached by Figure 5(a) in 75 steps, but Figure 5(b) demonstrates that it does not exit the bathroom. This issue appears since the agent failed to acquire the semantic relevant knowledge to go from the bathroom or room to the living room.

6 Path and Graph Visualizations

The experimental results on real world are visualized in Figure 6,7,8,9. All episode goals are sampled from $6m$ to $8m$. Since the locations of nodes are not accurate in real world, we collected the node locations for the best view.

Robot paths on the simulator are illustrated in Figure 10,11,12,13,14,15,16,17,18,19,20. In the Figures, the color gradation represents the flow of time. The blue arrow symbolizes the starting point, and the red flag indicates the destination. An image node is represented by a blue circle, and an object node is represented by a pink triangle. The paths are from test episodes proposed in VGM [4], also used in Table 1. The paths are drawn in two columns in pairs. The path generated by the pro-

posed algorithm is in the left column, and the path obtained by the comparison algorithm is in the right column. The routes provided are more effective and efficient than the comparison algorithm [4] and reach towards the target.

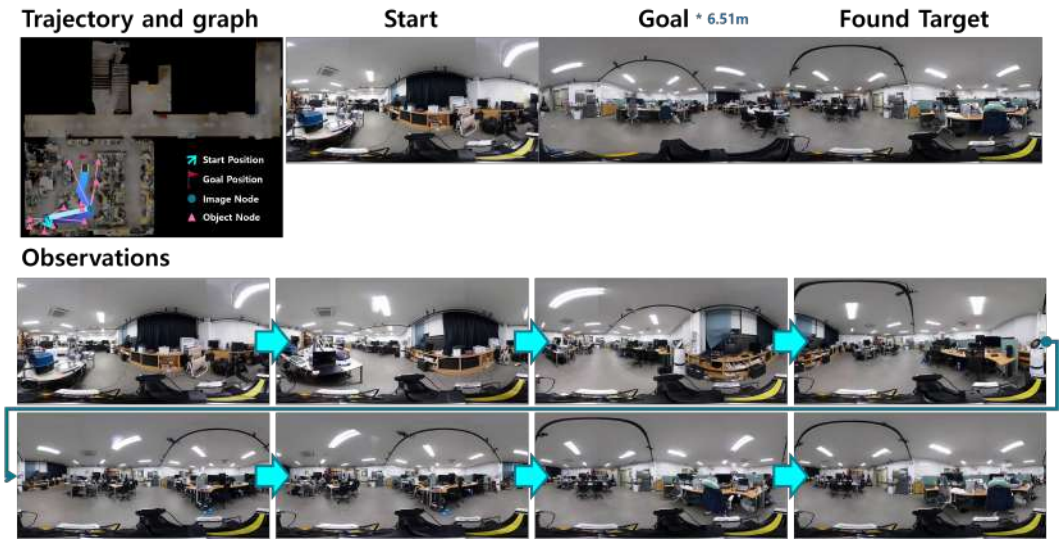

Figure 6: Real robot experimental results from the laboratory environment.

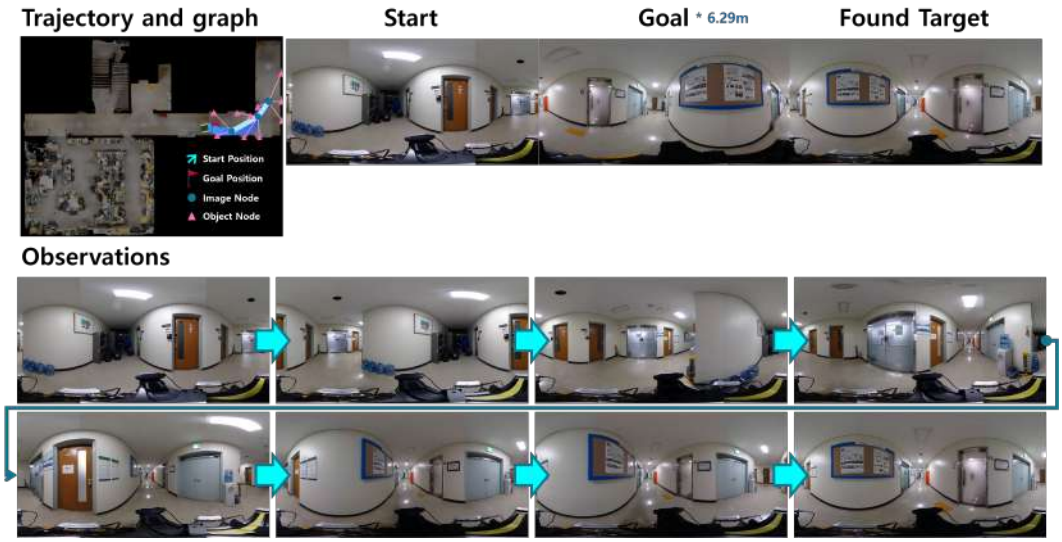

Figure 7: Real robot experimental results from the laboratory environment.

Figure 8: Real robot experimental results from the laboratory environment.

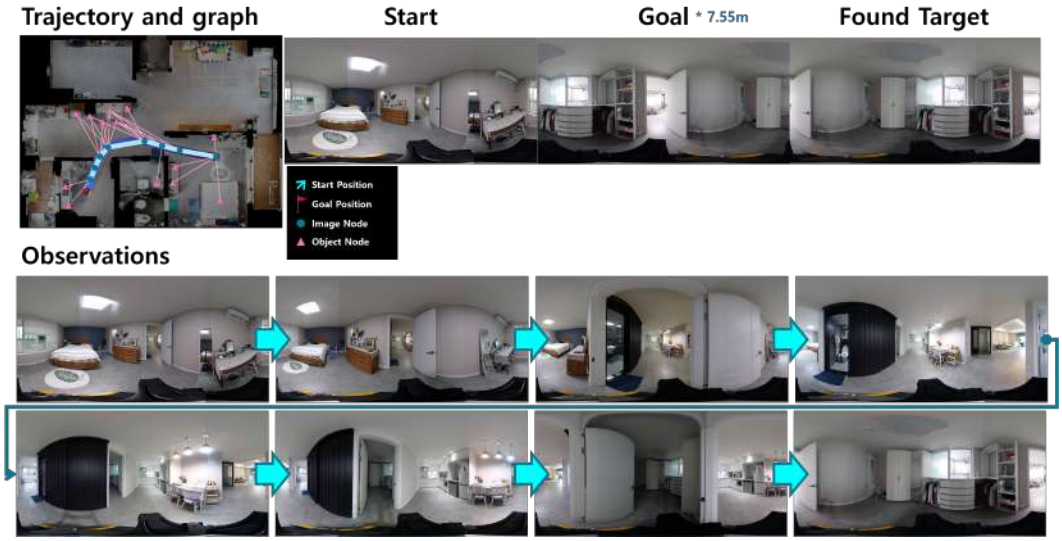

Figure 9: Real robot experimental results from the home environment.

Figure 10: Examples from Gibson's Cantwell environment.

Figure 11: Examples from Gibson's Cantwell environment.

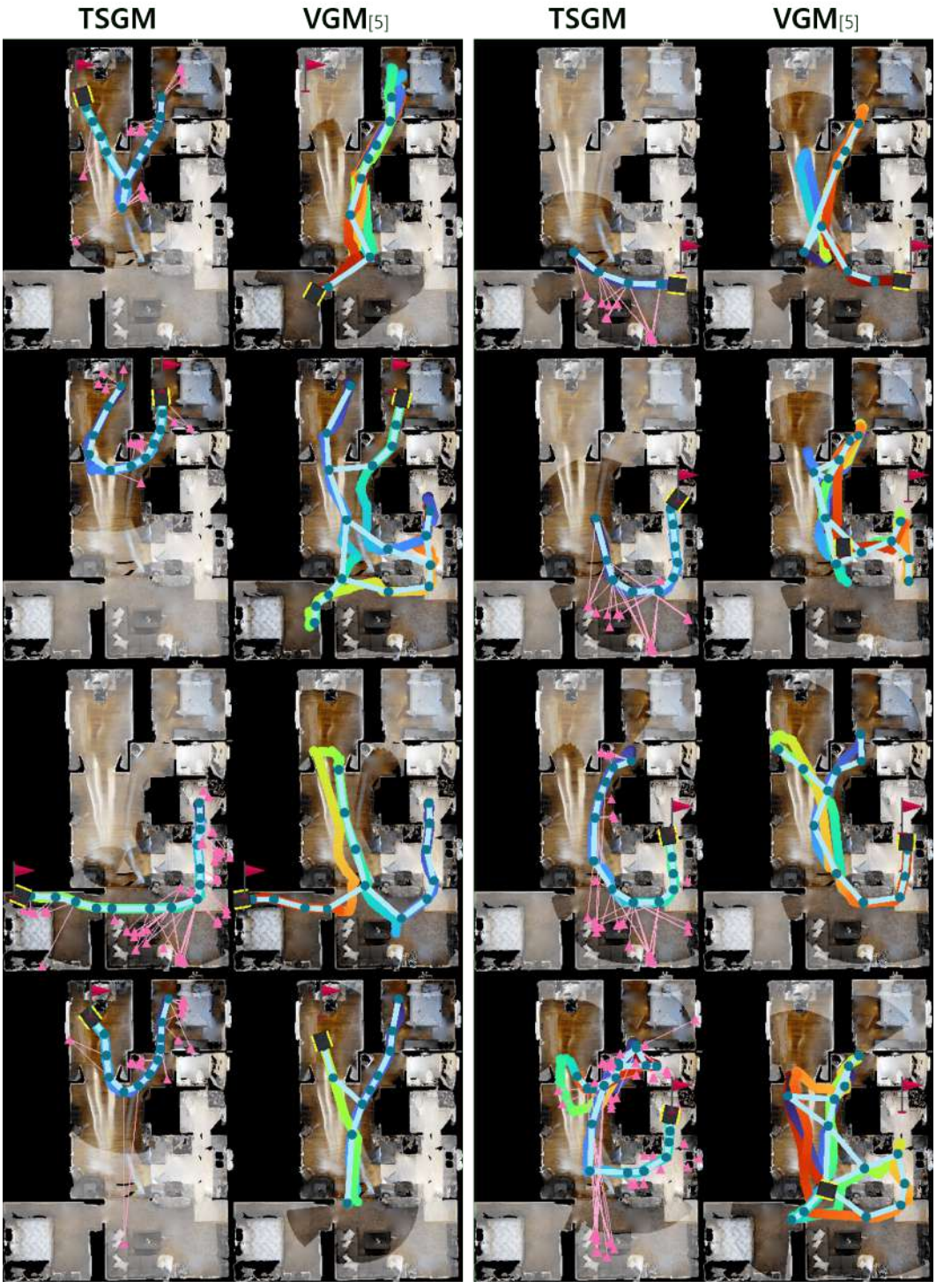

Figure 12: Examples from Gibson's Eastville environment.

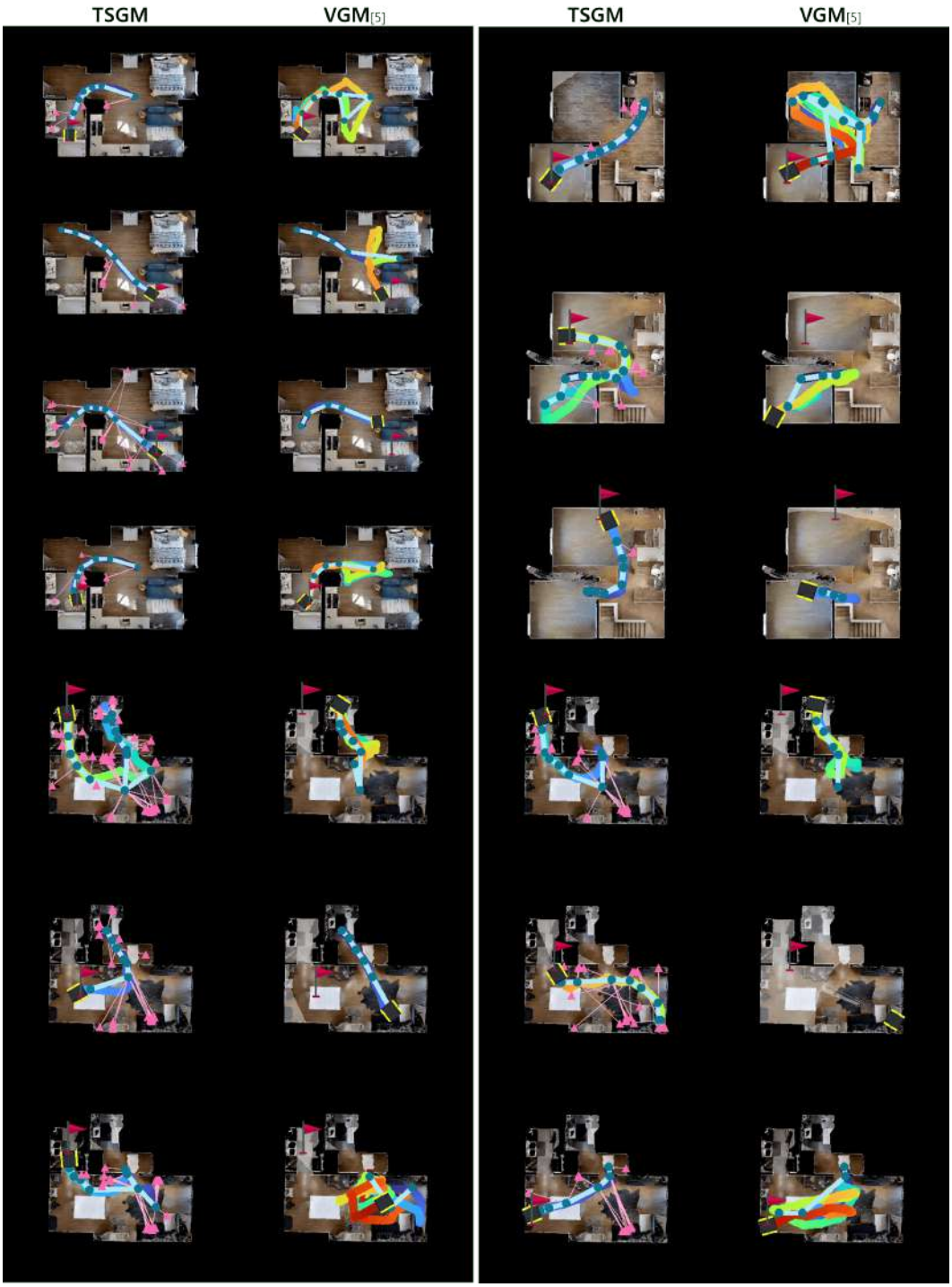

Figure 13: Examples from Gibson's Denmark (bottom), Greigville (top left), Ribera (top right) environment.

Figure 14: Examples from Gibson's Mosquito environment.

Figure 15: Examples from Gibson's Mosquito environment.

Figure 16: Examples from Gibson's Mosquito environment.

Figure 17: Examples from Gibson's Pablo (top) and Mosquito (bottom) environment.

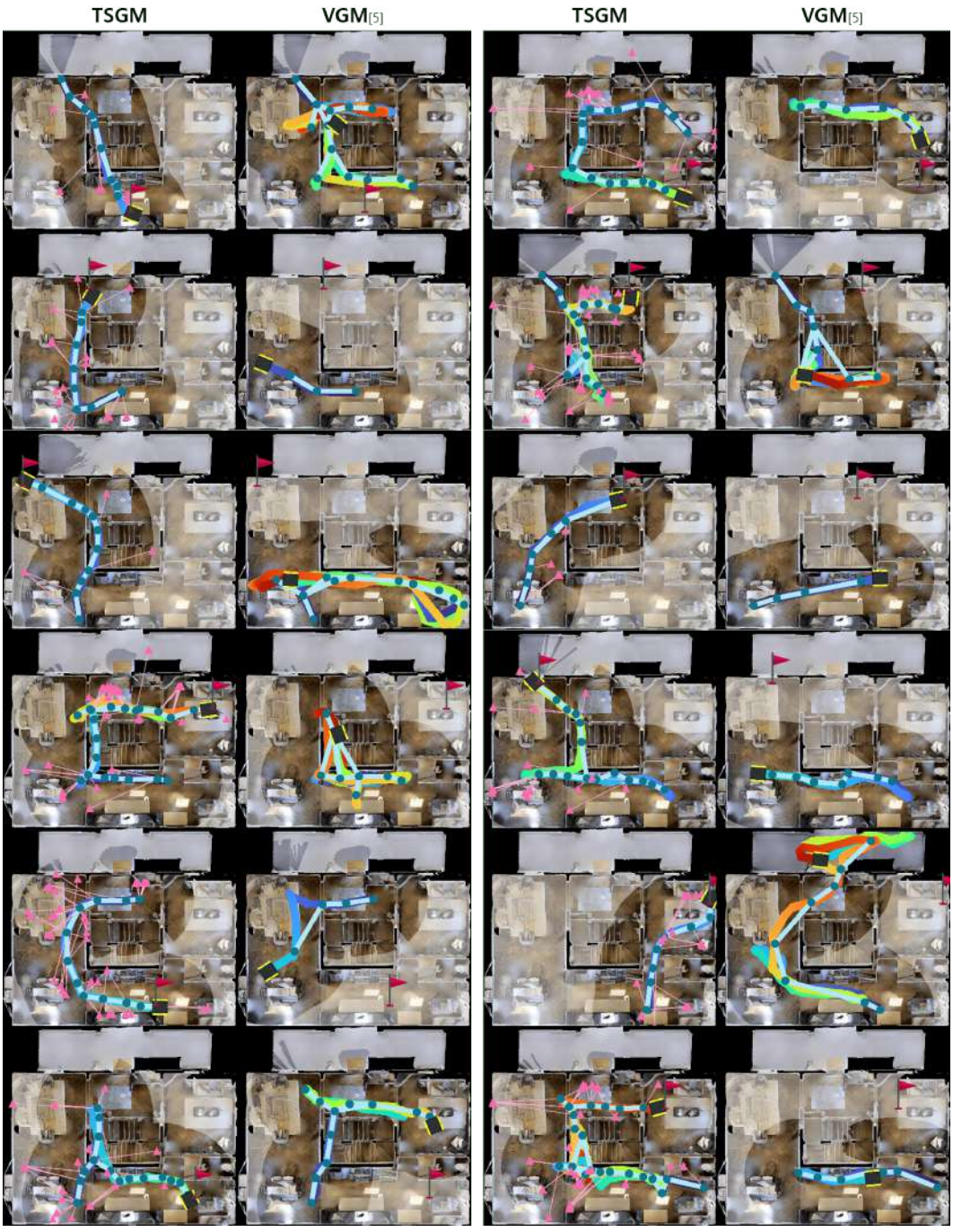

Figure 18: Examples from Gibson's Scioto environment.

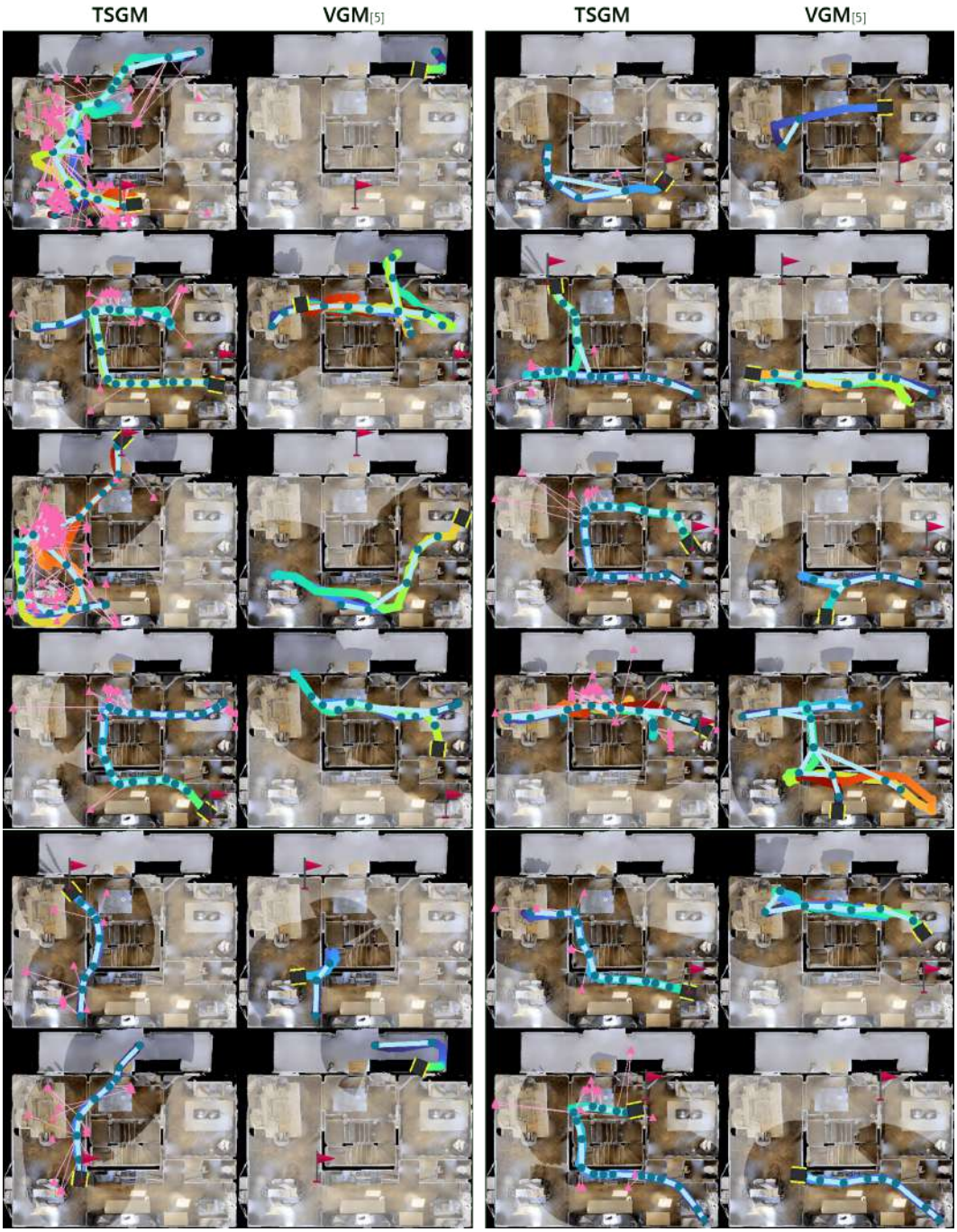

Figure 19: Examples from Gibson's Scioto environment.

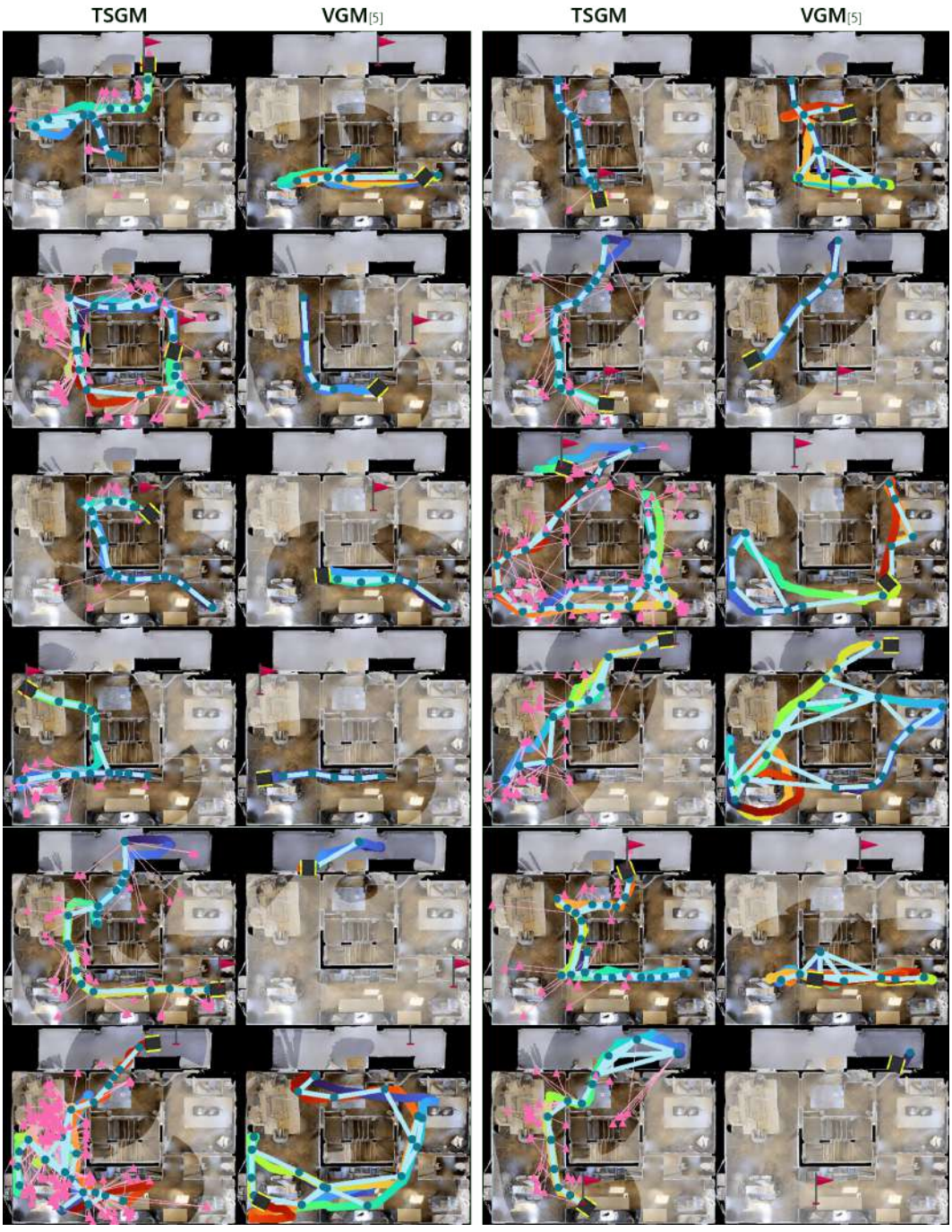

Figure 20: Examples from Gibson's Scioto environment.

References

[1] M. Hahn, D. S. Chaplot, S. Tulsiani, M. Mukadam, J. M. Rehg, and A. Gupta. No RL, No Simulation: Learning to Navigate without Navigating. In *Neural Information Processing Systems (NeurIPS)*, 2021.

[2] N. Zioulis, A. Karakottas, D. Zarpalas, and P. Daras. OmniDepth: Dense Depth Estimation for Indoors Spherical Panoramas. In *European Conference on Computer Vision (ECCV)*, 2018.

[3] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang. Matterport3D: Learning from RGB-D data in indoor environments. *International Conference on 3D Vision (3DV)*, 2017.

[4] O. Kwon, N. Kim, Y. Choi, H. Yoo, J. Park, and S. Oh. Visual Graph Memory with Unsupervised Representation for Visual Navigation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2021.

[5] J. Li, P. Zhou, C. Xiong, and S. C. Hoi. Prototypical Contrastive Learning of Unsupervised Representations. *International Conference on Learning Representations (ICLR)*, 2021.

[6] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan. Supervised contrastive learning. In *Neural Information Processing Systems (NeurIPS)*, 2020.

[7] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[8] H. Du, X. Yu, and L. Zheng. VTNet: Visual Transformer Network for Object Goal Navigation. In *International Conference on Learning Representations (ICLR)*, 2021.

[9] R. Fukushima, K. Ota, A. Kanezaki, Y. Sasaki, and Y. Yoshiyasu. Object Memory Transformer for Object Goal Navigation. In *IEEE international conference on robotics and automation (ICRA)*, 2022.