Renderable Neural Radiance Map for Visual Navigation: Supplementary Material

Obin Kwon Jeongho Park Songhwai Oh
Department of Electrical and Computer Engineering, ASRI, Seoul National University
obin.kwon@rllab.snu.ac.kr, jeongho.park@rllab.snu.ac.kr, songhwai@snu.ac.kr

We provide additional analyses and examples of the proposed method. The followings are included in this supplementary material:

A. Examples of reconstructed images

We provide examples of the reconstructed images from the decoder F_{dec} in Figure 2. The images are sampled from unseen environments, and not used during training. We embedded eight image observations for each scenario into RNR-Map and reconstructed the images. Also, we sample two novel views, which are not embedded in RNR-Map. We can see that F_{dec} can render images well in both seen and novel views. More importantly, RNR-Map is able to embed and reconstruct images from arbitrary scenes. RNR-Map can only render the observed region from various viewpoints, and the unobserved regions are rendered as empty space. Examples of the rendered unobserved region are shown in the last three cases in Figure 2.

Although the quality of each image is not state-of-the-art, RNR-Map can reconstruct the overall structure of the image and render large objects. Small objects and the details of the texture are often ignored in rendered images. The experiment results from the image-goal navigation task show that it is enough to accurately infer the camera pose based on the image renderings. Better training techniques and more weight parameters of the encoder and decoder will improve the image quality and navigation performance.

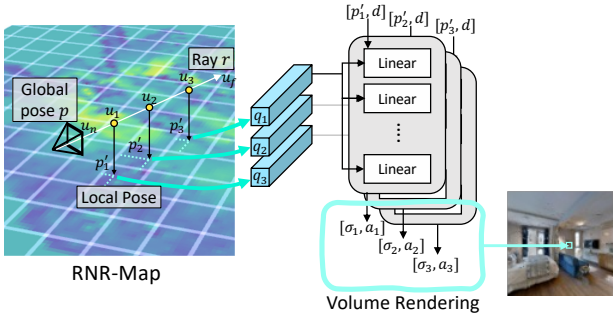


Figure 1. **Illustration of decoding process.**

B. Network Architecture and Training Details

B.1. Encoder and Decoder

We use images with a size of $128 \times 128 \times 4$. The encoder used in F_{enc} is a simple convolutional network consisting of four convolutional layers. The encoder embeds an image I into a same-sized feature C with 32 channels. We normalize each pixel feature $c_{h,w}$ along the feature dimension. These pixel features are averaged according to their (inverse) projected 3D positions and embedded into RNR-Map m.

The network structure of the decoder is adopted from GSN [2]. The illustration of the decoding process is shown in Figure 1. Given a query pose $\mathbf{p}$ and known camera intrinsic, we can calculate which 3D position will be rendered in a specific pixel (h, w). More specifically, assuming a ray which passes the pixel (h, w), we sample 3D points along the ray. Let u be the variable of how far the point is from the camera center along the ray. For each sampled 3D point, we can calculate its map position, and select the corresponding feature q from the m. As the calculated map position will have continuous values, we sample the feature using bilinear interpolation between the grids. GSN proposed a local coordinate system, which represents a 3D position of a point with a relative pose in a grid. The decoder network takes this local pose $\mathbf{p}'$, view direction $\mathbf{d}$, and the sampled latent code. This can be understood as the decoder rendering how a specific region would look like, from the local pose and view direction. The latent code becomes the modulation linear

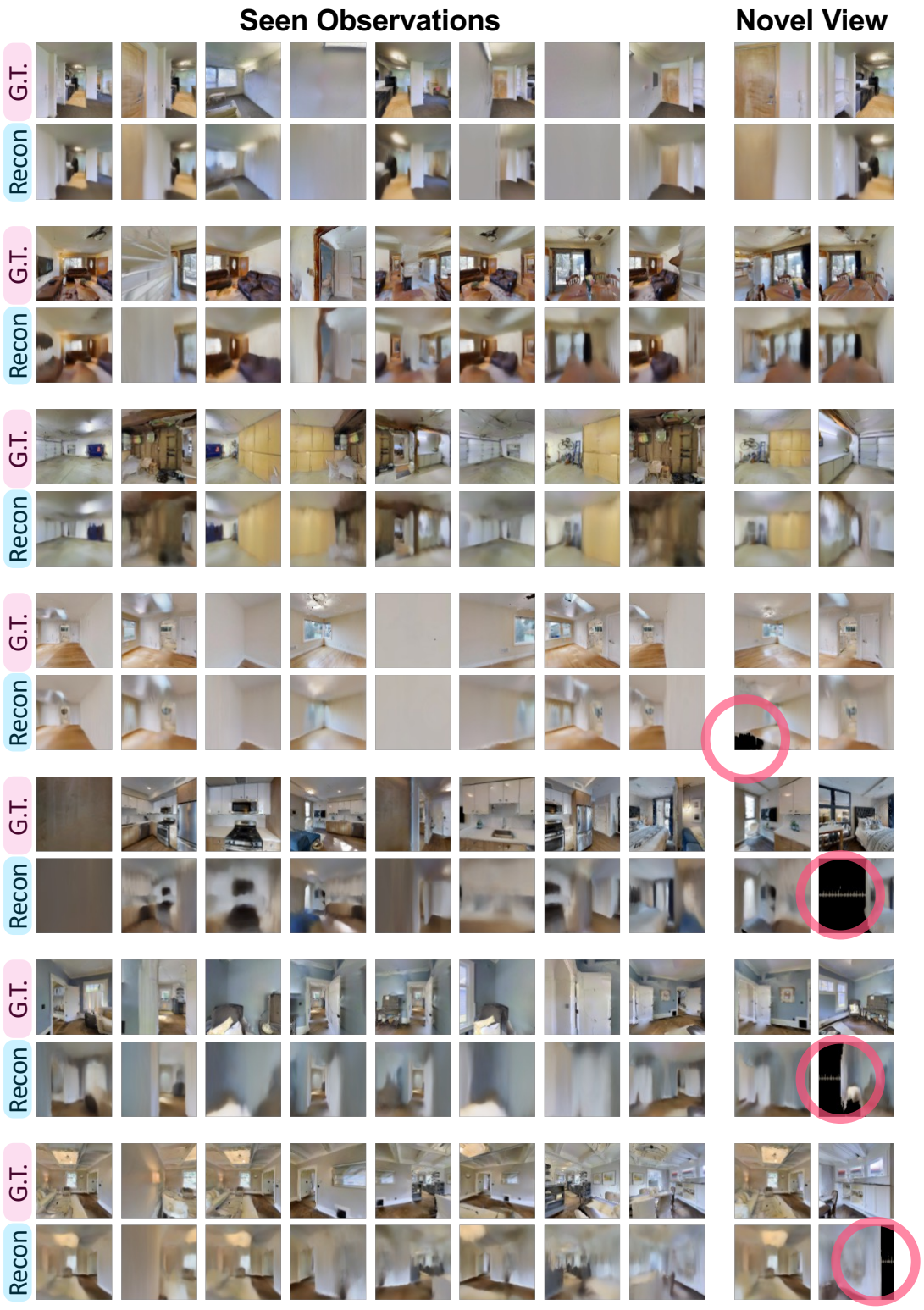


Figure 2. **Examples of the reconstructed images.** The odd rows are the ground-truth images, and the even rows are the reconstructed images from F_{dec}. In each row, eight images are embedded in RNR-Map. The last two columns are reconstructed from a novel view. Note that unobserved regions are rendered as an empty space (marked with red circle ❍). Each image is rendered with size 128×128.

layer, and outputs the occupancy σ, and appearance $\mathbf{a} \in \mathcal{R}^3$ The pairs of occupancy and appearance are calculated for all the sampled latent codes along the ray. Finally, the rendered color of a ray $\mathbf{r}$ is calculated with implicit volumetric rendering [11]:

$$c(\mathbf{r}, m) = \int_{u_n}^{u_f} Tr(u)\sigma(\mathbf{r}(u), q)\mathbf{a}(r(u), \mathbf{d}, q)du$$
$$Tr(u) = \exp(-\int_{u_n}^{u} \sigma(\mathbf{r}(u), q)du), \quad (1)$$

which is the same as the rendering function in GSN [2].

We only have changed the last step of the rendering in GSN. For computational efficiency, GSN renders a small-size of feature map and upsamples it as an image using a convolutional network. Since the objective of GSN is to generate a realistic scene, a whole image with good quality is needed to evaluate the generated scene. However, RNR-Map needs to individually render a small subset of pixels for the efficient calculation in the camera tracking function F_{track}. Hence, we did not use the last convolutional network in GSN, and replaced it with a simple linear layer. With this linear layer, we can get the color of the selected pixels, not requiring whole image rendering.

The size of the RNR-map represents $32m \times 32m$ area with $128 \times 128 \times 32$ size of a tensor, where 32 refers to the feature dimension. Each grid cell in RNR-map represents $25cm \times 25cm$ of a region.

B.2. Localization Network F_{loc}

The image-based localization is based on two RNR-Map, m and m_{trg}. m is constructed using partial observations from the environment, and m_{trg} is constructed with the given query image. Note that the query image is embedded in m_{trg} at the origin pose p_0, ($m_{trg} = F_{\text{reg}}(I_{\text{trg}}, p_0; \theta_{\text{enc}})$.) The localization proccess F_{loc} is done by convolving the given RNR-Map $m \in \mathcal{R}^{128\times128\times32}$ and the target RNR-Map $m_{trg}^{32\times32\times32}$. As m_{trg} has only the information of the target image at the origin, most of the grid cells are empty. Hence, we crop and only use the center of m_{trg}, to reduce unnecessary computations.

The localization network F_{loc} consists of three convolutional neural networks, F_k, F_q and F_E. The last neural network F_E has two heads F_{E_1}, F_{E_2}, whose outputs are the heatmap $\hat{E}$ and the predicted orientation of the query pose $\hat{a}_{\text{trg}}$, respectively. F_k, F_q, and F_{E_1} have the same U-Net architecture and output the same-size feature as the input. F_{E_2} is based on ResNet-18, which processes the output of the cross-correlation into the 18-angle bins.

B.3. Stopper F_{stop}

The stopper F_{stop} determines whether the agent is near the target location or not, based on m and m_{trg}. We employed the attention mechanism for F_{stop}, the overall procedure is illustrated in Figure 3. The local area has the majority of the information required to determine the relative distance to the target. Hence we crop the neighborhood of the agent position in m, and provide it as an input to F_{stop}. The cropped patch $m' \in \mathcal{R}^{32\times32\times D}$ and the target m_{trg} are flattened and considered as a sequence of the latent codes. We conduct self-attention for each sequence, and then conduct cross-attention between them as shown in Figure 3. The outputs of the cross-attention are unflattened as the original size and forwarded to a convolutional network. This convolutional network consists of four convolutional layers and a linear layer. The last linear layer outputs the prediction of the closeness to the target.

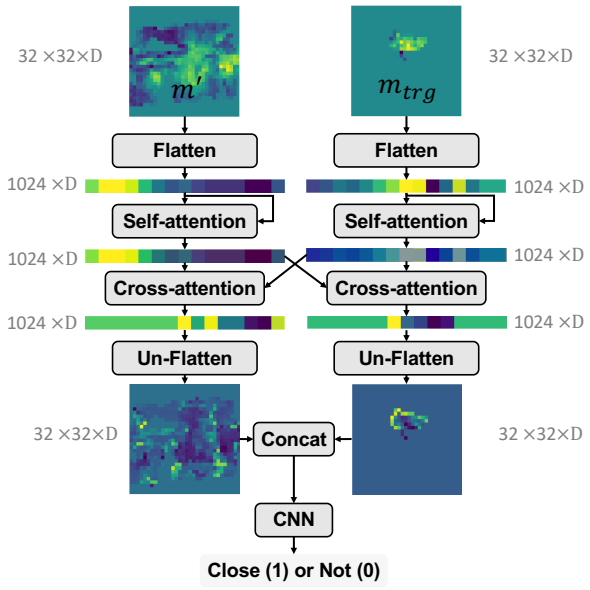


Figure 3. **Diagram of $\mathbf{F_{stop}}$.**

B.4. Training

We have collected 200 random navigation trajectories on 86 (training 72 + validation 14) scenes from the Gibson [23] dataset. We first trained the pair of encoder and decoder, and then trained the F_{loc} and F_{stop} with the frozen parameters of the encoder.

The neural networks used in RNR-Map are all trained with the same dataset. The data sample for each neural network is also created in a similar manner. We sample a trajectory from the dataset and select a subset of frames from the trajectory. The pose information of the subset is normalized with the first frame of the sampled subset, considering the first frame as the origin. The images are embedded in RNR-Map according to the normalized pose. Then we select a query frame from the trajectory. For the encoder and decoder, the query image is selected in the subset. They

are trained to reconstruct the query image from RNR-Map. When F_{loc} is trained for image-based localization, the query image is also selected from the subset which is embedded in RNR-Map. In contrast, when F_{loc} is trained for navigation, the query image is often selected outside of the subset, which is not provided in RNR-Map. Using this data sample, F_{loc} is trained to estimate which grid cell would be the closest to the target location. The F_{stop} is trained to determine the query position is in the neighborhood of the origin (the first frame of the sampled subset).

The training of the encoder and decoder is done with four GPUs (24GB NVIDIA GeForce RTX 3090), with a batch size of 16. The F_{loc} and F_{stop} are trained with one GPU (24GB NVIDIA GeForce RTX 3090), with a batch size of 32. All neural networks are trained until the validation loss converges.

C. Localization Experiments

C.1. Camera Tracking

We compare the proposed method on the camera tracking task against existing methods that build an environmental map in the latent space. **MapNet** [7] proposed a method to build an allocentric spatial memory from egocentric observation. This method builds a spatial memory for camera pose localization, which is done by comparing the latent features between the current image and the memory. The localization function F_{loc} in our method resembles this operation. **NICE-SLAM** [27] builds a renderable latent map of the environment using a differentiable rendering function (NeRF). Localization and mapping of this method are based on the photometric error between the rendered image and the observations. We used the official repository of each method[1]. The objective of this task is to estimate the camera trajectory, following a stream of image observations and noisy odometry sensor readings. We evaluate RNR-Map and MapNet with 1,000 trajectories from the validation scene, and a 10% subset for NICE-SLAM. We were only able to test a small subset for NICE-SLAM due to its large computational time. Each trajectory contains 1,000 frames of observations. We use root mean squared error of average trajectory error (ATE RMSE) [18] as a metric for the accuracy of camera tracking. The inference time is also measured, and it is the average mapping and tracking time for a single frame. The inference times of every method are measured on a desktop PC with a Intel i7-9700KF CPU @ 3.60GHz, and an NVIDIA GeForce RTX 2080 Ti GPU.

The experiment results are shown in Table 1. Raw noise in the first row shows the average trajectory error when the noises are not adjusted by any camera-tracking method. Our method is slightly slower than MapNet. The reason is that the localization function of MapNet is based on cross-correlation between the latent maps, while our camera tracking function F_{track} is based on rendering-based optimization. However, our method shows higher accuracy in inferring camera poses. MapNet discretizes the environment into grids and selects the most relevant grid cell for localization. In contrast, by rendering the image observations, we can adjust the camera pose at a finer level, smaller than the grid size.

Since our method directly embeds the image feature to the grids, RNR-Map shows a much faster inference speed because NICE-SLAM needs a rendering-based optimization process for mapping. The NICE-SLAM camera tracking results are significantly worse than the performances described in the original paper [27]. This is because our trajectory dataset is for navigation. There is much less overlap between each frame than in the dataset used for SLAM tasks, which results in performance deterioration. We also tested a different hyperparameter set for NICE-SLAM which conducts more optimization steps on both mapping and tracking (NICE-SLAM*). NICE-SLAM* outputs much more accurate results for the camera tracking. There is a significant trade-off between accuracy and inference time in NICE-SLAM, as more optimization steps lead to high accuracy on camera tracking but take considerable time.

C.2. Image-Based Localization

The objective of the image-based localization task is to find the pose of the query image, which is observed in the distant past. This task is different from camera tracking, which asks about the current pose of the agent, requiring relatively recent information. We tested each baseline on the image-based localization, and the results are shown in Table 1. We sampled 10 query images from each trajectory used in camera tracking (a total of 10,000 test samples). Also, in here, we were only able to test 10% of the samples for NICE-SLAM (1,000 test samples).

The RNR-Map is aggregated along the trajectory, preserving past information. As a result, it can be used to locate a target that has been observed in the distant past. MapNet [7] is focused on finding the camera pose at the moment, so it uses a recurrent neural network (RNN) for a better understanding of sequential observations. This RNN makes the method less effective in the image-based localization task because old information can be easily lost.

For NICE-SLAM, we render each pose candidate and select the best pose with the lowest photometric difference. As the rendering process takes a lot of time, we made NICE-SLAM more privileged, providing the possible pose candidates. The pose candidates are the recorded poses from the mapping process. NICE-SLAM shows a much longer inference time because the predicted location always has to be compared in the image domain, requiring a rendering process. In contrast, RNR-Map can find the location at high

[1]MapNet: https://github.com/jotaf98/mapnet, NICE-SLAM: https://github.com/cvg/NICE-SLAM

	(a) Camera Tracking + Mapping		(b) Image-Based Localization			
	ATE RMSE (m)	**Inference Time (s)**	**25cm Recall (%)**	**50cm Recall (%)**	**1m Recall (%)**	**Inference Time (s)**
Raw Noise	0.877	-	-	-	-	-
MapNet [7]	0.332	**0.104**	8.6	15.4	21.2	0.027
Nice-SLAM [27]	0.941	6.743	85.9	93.9	94.3	24.520
Nice-SLAM* [27]	**0.071**	25.977	**91.1**	93.7	94.4	24.520
RNR-Map (Ours)	0.108	0.271	76.6	**99.2**	**99.5**	**0.018**

Table 1. **Localization Results. (a) Camera Tracking.** ATE: Average Trajectory Error. The inference time is the average time of mapping and tracking time for a single frame. **(b) Image-Based Localization.** N-cm Recall refers to the ratio of the cases which the localization error is below N-cm. The inference time is the amount of calculation time for a single query, assuming a map is given.

Difficulty	**OC 1**	**OC 2**	**OC 3**	**OC 4**
Avg Img. diff.	7%	14.1%	23.8%	33.9%
Dist. Err.(m)	0.083	0.091	0.111	0.140
25cm Recall (%)	76.7	72.8	71.5	69.6
50cm Recall (%)	99.8	98.8	98.6	97.4
1m Recal (%)	99.8	99.0	99.2	98.5

Table 2. **Localization Results on Object Change scenarios.** Avg Img. Diff.: Average image differences between the query image and the originally observed image. Dist. Err.: Distance error between the localized position and the query position.

speed with high accuracy, by directly comparing the latent codes rather than rendering every position. F_{loc} of RNR-Map locates the target with less than 50cm with an accuracy of 99%, with a fast speed of 0.018 seconds. F_{loc} selects the most probable grid which corresponds to $25cm \times 25cm$ region. This reduces the performance of the 25cm Recall, which requires precise localization finer than the grid size. However, compared to rendering each image and comparing them to the query, the RNR-Map still exhibits 5.87% higher localization performance in 50cm Recall with an incomparably faster time.

We provide some examples of success cases and failure cases from experiment results in Figure 4. The F_{loc} generally finds the query observation with a small error. However, when there are multiple visually-similar regions in the given environment, F_{loc} often selects the wrong places. We can see that the found location has a similar visual appearance to the query image.

C.3. Similar-image-goal localization

We can leverage this RNR-Map for searching the most similar place to the query image even if the exact place is not in the current environment. Two scenarios can be considered: (1) (Object Change) There have been large changes in the environment so the object configuration of the environment is different from the information in the RNR-Map. (2) (Novel Environment) The user only has a query image from a different environment but wants to find the most similar place in the current environment.

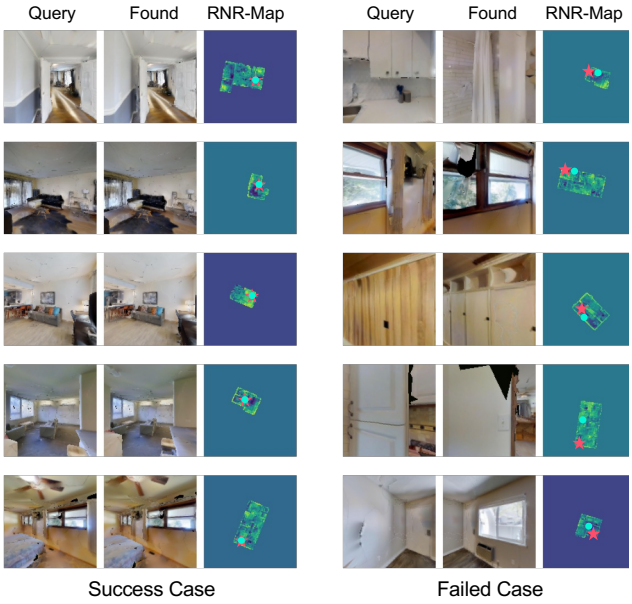


Figure 4. **Image-based Localization Examples.** The first column shows some examples of successful localization, and the second column shows some examples of failed localization. The query location is marked as red dot, and the predicted localization is marked as a blue dot.

Object change. We divided the difficulty of the localization scenario with the changed environments into four levels: OC1, OC2, OC3, and OC4. We first recorded RNR-Map map for each validation scene without any additional objects. Then we randomly place the random objects (bags, boxes, sofa, chair, toys, *etc.*) in the scenes, and take pictures of the changed environment. We gave this picture with objects as a query image. The difficulty is determined by how much the image has been changed because of the random objects. The image difference is calculated as the L1 loss between the newly captured image with the random objects and the original image without objects. This value is normalized by the size of an image so that it represents how much the image has been changed from the original. Each difficulty contains 300 samples of the key scene and query image pair.

We show the quantitative results in Table 2. We observed that the RNR-Map robustly localizes query images even if some object configuration changes. The RNR-Map finds the query location with less than 50cm error in 97.4% of the cases, even in the most difficult scenario. The qualitative results are shown in Figure 6a.

Novel environments. We also tested the novel environment (NE), where the query image is from a different scene. The objective of this task is to find the most visually similar location to the query image. However, the visual similarity is hard to quantify and can vary depending on the metric. To evaluate the localized observation, we use various types of metrics which can measure the similarity between images. We leverage two image encoders which are trained using contrastive learning (CLIP, CL). In contrastive learning, the image encoder converts images into feature vectors and is trained to maximize the cosine similarity between the features from similar images. Using the two contrastive learning models, we calculate the cosine similarity of the feature vectors from the query image and the localized observation. We also utilize the structural similarity index (SSIM) and L1 loss to measure the visual similarity. The following metrics are employed:

- **CLIP** [15]: CLIP [15] is a weakly supervised vision-language model with a web-scale dataset, which is trained to embed images and text into a joint latent space. CLIP encoding contains a semantic understanding of an image, and this model is widely used in various kinds of downstream tasks [3, 5, 13, 14, 16, 19]. We use CLIP for measuring visual similarities between images. The weight parameters from the publicly released model [2] are used. This model only takes RGB images.
- **CL (SupContrast) [8]**: We use a contrastive learning model [8] specially trained on the images from the Gibson [23] and MP3D [1] datasets. During the training, similar images are defined by the physical distance between the positions where the image was taken. This model is trained to encode the images from the same region (maximum 2m apart) into similar feature vectors, which show high cosine similarity. This model takes RGBD images.
- **SSIM [21] (Structural similarity index)**: We measured SSIM between the depth images of the query image and the localized observation. This measure can represent the similarity between the 3D geometric structures in images.
- **(Inverse) L1**: We measure the direct pixel differences between the images. The sum of L1 losses from both RGB and depth images are used. As L1 represents the distance between the images, we inverse the value to use it as a similarity measure.

[2] https://github.com/openai/CLIP

Search Method \ Similarity (%)	CLIP [15]	CL	SSIM	L1
Random	88.23	30.91	78.66	88.44
Max CLIP [15]	100.0	61.36	84.58	90.48
Max CL	93.26	100.0	**91.29**	91.60
Max SSIM	93.75	66.98	100.0	**93.40**
Max Inv. L1	90.77	49.68	90.91	100.0
RNR-Map (Ours)	**94.52**	**82.9**	90.93	91.41

Table 3. **Novel Environment Localization Results.** The rows show each search method that finds the most visually similar place based on own metric or method (random, ours). Each column reports the normalized visual similarity from each metric, measuring the image found by each search method in rows. Naturally, the measured similarity of a metric found by the same metric would be 100%.

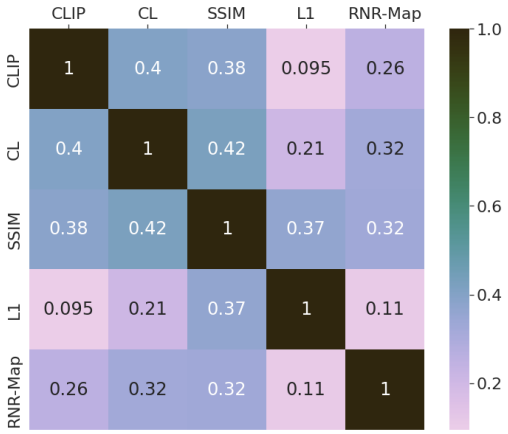


Figure 5. **Correlation matrix of the similarity metrics and the probability value from RNR-Map.**

We tested 100 novel environment scenarios. As the query image is not from the same scene, the maximum visual similarity may vary depending on the given scene. We normalize each similarity metric by the maximum possible value from the given scene. The normalized value represents how much the found location is visually similar compared to the most visually-similar location. The experiment results are shown in Table 3. The random in the first row shows how much the metrics would appear when we select a random position.

We observed that showing high similarity on one metric did not always imply high similarity on other metrics. We report the experimental results when a query image is searched based on a specific metric (second to the fifth row of Table 3). We sampled all possible locations from the given scene and evaluated each image with the metrics. For example, the image which shows the highest CLIP visual similarity becomes the localized image by using CLIP as a search method (Max CLIP in Table 3). The CL column of the Max CLIP row

(a) **Examples of Object Change scenarios.** Examples from OC2, OC3 and OC4 levels are shown in the figure. More examples are provided in the supplementary video.

(b) **Examples of Novel Environment scenarios.** Query is the given localization query from different environments, and Found is the localized observation found by RNR-Map. The image pairs are sorted based on SSIM similarity and CL similarity.

Figure 6. **Examples of similar-image-goal localization task.**

	Gibson-Curved				Easy		Medium		Hard		Overall	
	Exloration Score	**Latent Score**	**GT Point Navi**	**GT Stopper**	**SR**	**SPL**	**SR**	**SPL**	**SR**	**SPL**	**SR**	SPL
Ground-truth Information	✔	✔	✔	✔	98.0	85.0	95.0	70.8	86.4	53.9	93.1	69.9
	✔	✔	✘	✔	94.6	81.8	90.1	66.9	72.2	45.0	85.6	64.6
	✔	✔	✔	✘	72.4	61.2	65.2	47.1	55.2	33.2	64.3	47.1
Exploration Strategy	✔	✘	✘	✘	59.0	39.5	57.0	35.5	37.9	21.5	51.3	32.2
	✘	✔	✘	✘	71.1	**60.5**	65.9	45.4	40.4	24.2	59.2	43.4
Full Model	✔	✔	✘	✘	**71.7**	59.9	**67.2**	**47.5**	**45.2**	**29.2**	**61.4**	**45.6**

Table 4. **Ablation studies of the navigation module on Gibson-curved scenarios. Exploration score** and **Latent score** are used in the exploration module for planning where to visit. **GT Point Navi** and **GT Stopper** are ablations of the point navigation policy and the stopper module F_{stop}, respectively. Each module is replaced with a simple function that has access to ground-truth information from the simulator.

shows that the image which has the highest CLIP similarity shows 61.36% of CL similarity compared to the maximum CL similarity. We can see that the maximum of each metric does not always mean the maximum in the other similarity metrics.

Meanwhile, the RNR-Map localizes a query image with consistently high value in all metrics (the last row). The RNR-Map shows general visual similarity in various metrics, in contrast to using a specific metric as a search method. We can infer that the RNR-Map finds a visually similar location even with a query image from a different environment.

Furthermore, we want to stress that the localization process F_{loc} of RNR-Map is done with a forward pass to the neural network with 56.8Hz of speed. Even with the fast speed, RNR-Map still achieves competitive similarity compared to the cases when all the possible images are compared with the query image one by one. We also plot the correlation matrix between the metrics and the probability value from RNR-Map in Figure 5. The RNR-Map shows positive correlations with the similarity metrics. The examples of the query image and the localized observation pairs are shown in Figure 6b. In Figure 6b, the image pairs are sorted by CL similarity and SSIM similarity. We observed that SSIM similarity is high when the overall 3D structures of the images are similar, while CL similarity is high when the overall colors of the images are similar.

D. Navigation ablation studies

In this section, we report the ablation studies of the navigation module. The results are shown in Table 4, and each method is evaluated on Gibson-curved scenario from NRNS dataset [6] without noise setting.

Ground-truth information. We provided the ground-truth information to the point navigation policy and the stopper module. With ground-truth information, the point-navigation policy can reach the target using the shortest path without collision. The stopper module with the ground-truth information has access to the geodesic distance to the target, so that it can detect the target location and take a stop action accurately. We can observe that navigation performance increases when given ground-truth information. The information about the distance to the target location is critical to the performance. Both the success rate and SPL dramatically increase because the ground-truth information enables efficient navigation without wandering or collision. Also, the agent has no risk of taking the false-stop action (terminating the episode even when the target is far) or overlooking the target location. We observed that the ground-truth point navigation policy often leads to the failure of an episode. Hence, in the medium scenarios, the model without the ground-truth point navigation policy (full model in the last row) shows better performance than the one with the ground-truth point navigation policy. This is because of the relatively inefficient path from our point-navigation module, which leads to a random chance to explore the environment more and find the target location. We anticipate that an improved point navigation policy and better F_{stop} will largely improve the navigation performances.

Exploration strategy. The exploration module in the navigation module selects where to explore in order to find the goal location. The exploration module draws a Voronoi graph on the occupancy map and selects a node as an exploration candidate from the created graph. Two criteria are used for selecting the exploration candidates, latent score and exploration score. The latent score is based on the localization heatmap value from F_{loc}. The occupancy map has three types of values, free (1), occupied (2), and unseen (0). The exploration score is based on the number of unseen pixels in the neighborhood of a pixel. The more unseen pixels are in the neighborhood, the more likely a location has been underexplored and has a high probability of discovering new areas. We ablated each score, and the results are shown in Table 4. Without the latent score, only the exploration score is used for exploring the environment. The exploration score helps the agent visit all the possible places in the scene. However, this score does not consider the information from the target, so the agent may not be able

Training Dataset	Method	Easy		Medium		Hard		Overall	
		SR	SPL	SR	SPL	SR	SPL	SR	SPL
MP3D	**NRNS [6]**	23.7	12.7	16.2	8.3	9.1	5.1	16.3	8.7
MP3D	**SLING+NRNS [22]**	43.2	19.7	32.5	15.1	22.1	9.9	32.6	14.9
Gibson	**OVRL [24]**	16.9	8.3	25.8	13.5	14.3	7.0	10.6	4.6
MP3D	**OVRL [24]**	52.4	35.2	42.6	26.3	29.7	16.9	41.6	26.1
Gibson	**SLING+OVRL [22]**	47.5	30.2	28.4	17.0	19.3	9.3	31.7	18.8
MP3D	**SLING+OVRL [22]**	**62.6**	41.1	48.4	31.5	29.2	17.7	46.7	30.1
Gibson	**RNR-Map (ours)**	58.42	**49.01**	**50.35**	**35.57**	**34.38**	**22**	**47.7**	**35.5**

Table 5. **Image-goal navigation results on MP3D [1]-curved scenarios in NRNS [6] dataset.** The first column shows the training dataset each method has trained on.

to find the target within the time limitation due to inefficient exploration. We can see that the agent finds the target more successfully when only the latent score is used. In the full model, we use the sum of the exploration score and the latent score. Using the combination of two scores helps the agent to find the target location efficiently. The latent score can guide the agent to the target location based on the RNR-Map. However, when there is not much information or clues about the target location, the exploration score can encourage the agent to explore the environment.

E. Navigation on MP3D Dataset

We provide the image-goal navigation experiment results on MP3D dataset in Table 5. The experiments are done without noise setting, and we report the digits from [6], [22] for the baselines. The proposed RNR-Map-based navigation framework outperforms the baselines, except for the success rate in easy scenarios. The neural networks in RNR-Map are trained using Gibson [23] dataset. Based on the results, we can observe that the RNR-Map-based framework is generalizable to a different dataset, without any fine-tuning.

F. Navigation Examples

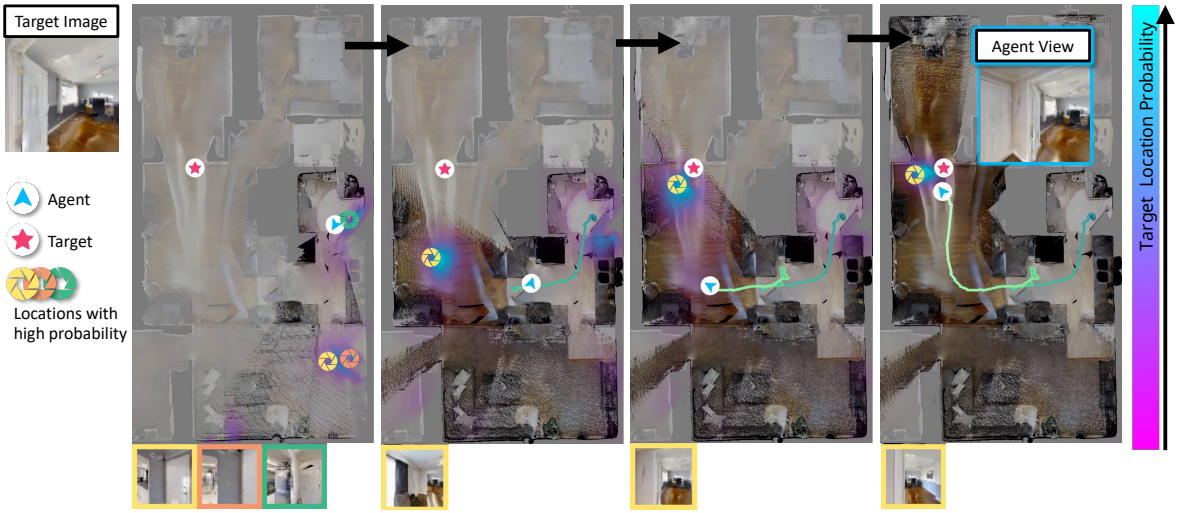
We provide examples from the image-goal navigation episodes of RNR-Map in Figure 7. The latent scores usually highlight unexplored areas at the beginning of episodes. This is because the F_{loc} for navigation is trained to predict the closer area to the target given the partial information of the environment. We can see that the way to other rooms is highlighted in the first column of Figure 7a and 7b. The agent follows the latent score and expands its RNR-Map according to the image observations. During the exploration, the agent observed the target-related region and successfully reached the target location. More examples are provided in the attached video.

G. Implementation Details for the navigation submodules

Graph Generation For efficient exploration planning, we discretize the region in the observed environment into a graph. Figure 8 shows the process of graph generation. This method is inspired by robot exploration literature [9, 12, 25, 28], which draws an (approximated) Voronoi graph on the occupancy map. Originally, the Voronoi graph consists of nodes that are equally distanced from the neighbor obstacles. We simplify this graph construction with the image skeletonizing method in the image processing library (`skimage.morphology.skeletonize`) [20]. This function is based on the image thinning algorithm proposed in [26]. As the skeleton only consists of lines, we need to determine which pixels will be the nodes. We select the pixels that have many neighbors as a node. The neighbor of more than two pixels indicates that the pixel is not on a line, but instead in the intersection of several lines. Then we split the skeleton based on the selected nodes, and determine the relationship between the nodes. The created graph is used for planning the exploration of the agent. The exploration module selects the node to explore, rather than sampling a pixel among the free spaces in the occupancy map.

Exploration Score The exploration module evaluates each node in the graph with two criteria. The first one is the latent score, which is from the heatmap from F_{loc} in the localization module. This heatmap highlights the place related to the target image. The second one is the exploration score, which is based on the number of unseen pixels in the neighborhood of the node. A location is more likely to have been underexplored and have a high likelihood of discovering new areas if there are more unseen pixels nearby. The examples of calculating the exploration score are presented in Figure 8. Drawing a set of rays centered at the node, we count the number of unseen pixels in the rays. We also evaluate whether a ray is blocked by an obstacle, and limit the length of the ray to less than the distance to the blocked obstacle. The number of unseen pixels is normalized into 0.0 to 1.0, and this

(a) **Example 1.**



(b) **Example 2.**

Figure 7. **Examples from image-goal navigation episodes.** The heatmap values (the latent score) from F_{loc} are presented on the map according to the color bar on the right. We also marked the locations of high-probability values with the images from the location.

value becomes the exploration score. Furthermore, we also calculate the distance from the agent to each node and add the inverse value to the exploration score. This encourages greedy exploration, which explores the near neighborhood first.

Point Navigation Module The point navigation module takes the map position of the agent and the selected node position to explore. This module calculates the collision-free shortest path to the node based on the fast-marching method. We used the open-source library for the fast-marching method [3]. After a navigation path is obtained, the point navigation module outputs an appropriate action based on its relative pose to the path points.

Stopper Module The stopper module has two components, F_{stop} and the last-mile approaching function which is adopted from [22]. First, F_{stop} determines whether the agent is near the target location. Second, if F_{stop} found that the target is near the agent, we conduct keypoint-matching [17] between the current observation and the target image. If a sufficient number of keypoints are matched, we can infer

[3] https://github.com/scikit-fmm/scikit-fmm

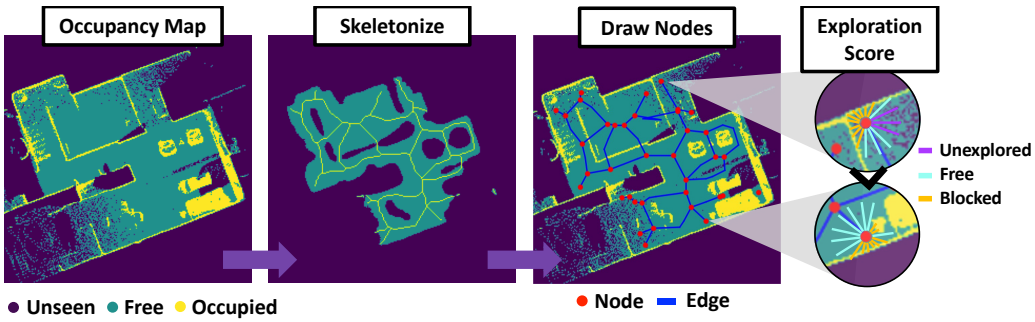


Figure 8. **Overview of the graph generation.** We take the free area of the occupancy map and skeletonized the image. Then, we draw a graph on the skeleton. Each node has an exploration score, which is calculated based on the number of unseen pixels in the neighborhood.

the relative pose between the target location and the current position using the depth information. As proposed in [22], we use Perspective-n-Point [10], and RANSAC [4]. After the relative pose is determined, we set the local goal point, and the point navigation module navigates to the estimated target. If the number of matched keypoints decreases below a certain threshold (20 in our case), the last-mile approach is terminated, and the exploration module selects the next exploration target.

References

[1] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3D: Learning from RGB-D Data in Indoor Environments. *International Conference on 3D Vision (3DV)*, 2017. 1, 6, 9

[2] Terrance DeVries, Miguel Angel Bautista, Nitish Srivastava, Graham W. Taylor, and Joshua M. Susskind. Unconstrained scene generation with locally conditioned radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021. 1, 3

[3] Han Fang, Pengfei Xiong, Luhui Xu, and Yu Chen. CLIP2Video: Mastering Video-Text Retrieval via Image CLIP. *arXiv preprint arXiv:2106.11097*, 2021. 6

[4] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 11

[5] Kevin Frans, Lisa B Soros, and Olaf Witkowski. Clipdraw: Exploring text-to-drawing synthesis through language-image encoders. *arXiv preprint arXiv:2106.14843*, 2021. 6

[6] Meera Hahn, Devendra Chaplot, Shubham Tulsiani, Mustafa Mukadam, James Rehg, and Abhinav Gupta. No RL, No Simulation: Learning to Navigate without Navigating. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 8, 9

[7] João F. Henriques and Andrea Vedaldi. MapNet: An Allocentric Spatial Memory for Mapping Environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 4, 5

[8] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised Contrastive Learning. 2020. 6

[9] Jonghoek Kim, Fumin Zhang, and Magnus Egerstedt. A provably complete exploration strategy by constructing voronoi diagrams. *Autonomous Robots*, 29(3):367–380, 2010. 9

[10] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate O(n) solution to the PnP problem. *International Journal of Computer Vision*, 81(2):155–166, 2009. 11

[11] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020. 3

[12] Keiji Nagatani and Howie Choset. Toward robust sensor based exploration by constructing reduced generalized voronoi graph. In *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No. 99CH36289)*, volume 3, pages 1687–1692. IEEE, 1999. 9

[13] Daniil Pakhomov, Sanchit Hira, Narayani Wagle, Kemar E Green, and Nassir Navab. Segmentation in style: Unsupervised semantic image segmentation with stylegan and clip. *arXiv preprint arXiv:2107.12518*, 2021. 6

[14] Or Patashnik, Zongze Wu, Eli Shechtman, Daniel Cohen-Or, and Dani Lischinski. Styleclip: Text-driven manipulation of stylegan imagery. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2085–2094, October 2021. 6

[15] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2021. 6

[16] Aditya Sanghi, Hang Chu, Joseph G. Lambourne, Ye Wang, Chin-Yi Cheng, Marco Fumero, and Kamal Rahimi Malekshan. CLIP-Forge: Towards Zero-Shot Text-To-Shape Generation. In *Proceedings of the IEEE/CVF Conference on*

Computer Vision and Pattern Recognition (CVPR), pages 18603–18613, June 2022. 6

[17] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. SuperGlue: Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 10

[18] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 573–580. IEEE, 2012. 4

[19] Yingtao Tian and David Ha. Modern evolution strategies for creativity: Fitting concrete images and abstract concepts. In *International Conference on Computational Intelligence in Music, Sound, Art and Design (Part of EvoStar)*, pages 275–291. Springer, 2022. 6

[20] Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014. 9

[21] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004. 6

[22] Justin Wasserman, Karmesh Yadav, Girish Chowdhary, Abhinav Gupta, and Unnat Jain. Last-Mile Embodied Visual Navigation. In *Proceedings of the Conference on Robot Learning (CoRL)*, 2022. 9, 10, 11

[23] Fei Xia, Amir R. Zamir, Zhi-Yang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson Env: Real-World Perception for Embodied Agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 3, 6, 9

[24] Karmesh Yadav, Ram Ramrakhya, Arjun Majumdar, Vincent-Pierre Berges, Sachit Kuhar, Dhruv Batra, Alexei Baevski, and Oleksandr Maksymets. Offline Visual Representation Learning for Embodied Navigation. *arXiv preprint arXiv:2204.13226*, 2022. 9

[25] Qiwen Zhang, David Whitney, Florian Shkurti, and Ioannis Rekleitis. Ear-based exploration on hybrid metric/topological maps. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014. 9

[26] Tongjie Y Zhang and Ching Y. Suen. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):236–239, 1984. 9

[27] Zihan Zhu, Songyou Peng, Viktor Larsson, Weiwei Xu, Hujun Bao, Zhaopeng Cui, Martin R. Oswald, and Marc Pollefeys. NICE-SLAM: Neural Implicit Scalable Encoding for SLAM. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 4, 5

[28] Xinkai Zuo, Fan Yang, Yifan Liang, Zhou Gang, Fei Su, Haihong Zhu, and Lin Li. An Improved Autonomous Exploration Framework for Indoor Mobile Robotics Using Reduced Approximated Generalized Voronoi Graphs. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 1:351–359, 2020. 9