Renderable Neural Radiance Map for Visual Navigation

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

Abstract

We propose a novel type of map for visual navigation, a renderable neural radiance map (RNR-Map), which is designed to contain the overall visual information of a 3D environment. The RNR-Map has a grid form and consists of latent codes at each pixel. These latent codes are embedded from image observations, and can be converted to the neural radiance field which enables image rendering given a camera pose. The recorded latent codes implicitly contain visual information about the environment, which makes the RNR-Map visually descriptive. This visual information in RNR-Map can be a useful guideline for visual localization and navigation. We develop localization and navigation frameworks that can effectively utilize the RNR-Map. We evaluate the proposed frameworks on camera tracking, visual localization, and image-goal navigation. Experimental results show that the RNR-Map-based localization framework can find the target location based on a single query image with fast speed and competitive accuracy compared to other baselines. Also, this localization framework is robust to environmental changes, and even finds the most visually similar places when a query image from a different environment is given. The proposed navigation framework outperforms the existing image-goal navigation methods in difficult scenarios, under odometry and actuation noises. The navigation framework shows 65.7% success rate in curved scenarios of the NRNS [23] dataset, which is an improvement of 18.6% over the current state-of-the-art. Project page: https://rllab-snu.github.io/projects/RNR-Map/

1. Introduction

In this paper, we address how to explicitly embed the visual information from a 3D environment into a grid form and how to use it for visual navigation. We present *renderable neural radiance map (RNR-Map)*, a novel type of a grid map for navigation.We point out three main properties of RNR-Map which make RNR-Map navigation-friendly. First, it is *visually descriptive*. Commonly used grid-based maps such as occupancy maps [10, 11, 17] and semantic maps [9, 21, 48], record obstacle information or object information into grids. In contrast, RNR-Map converts image observations to latent codes which are then embedded in grid cells. Each latent code in a grid cell can be converted to a neural radiance field, which can render the corresponding region. We can utilize the implicit visual information of these latent codes to understand and reason about the observed environment. For example, we can locate places based on an image or determine which region is the most related to a given image. RNR-Map enables image-based localization only with a simple forward pass in a neural network, by directly utilizing the latent codes without rendering images. We build a navigation framework with RNR-Map, to navigate to the most plausible place given a query image. Through extensive experiments, we validate that the latent codes can serve as important visual clues for both image-based localization and image-goal navigation. More importantly, a user has an option to utilize the renderable property of RNR-Map for more fine-level of localization such as camera tracking.

RNR-Map is *generalizable*. There have been a number of studies that leverage neural radiance fields (NeRF) for various applications other than novel view synthesis. The robotic applications of NeRF are also now beginning to emerge [1, 15, 31, 44, 55]. However, many of the approaches require pretrained neural radiance fields about a specific scene and are not generalizable to various scenes. This can be a serious problem when it comes to visual navigation tasks, which typically assume that an agent performs the given task in an unseen environment [16]. In contrast, RNR-Map is applicable in arbitrary scenes without additional optimization. Even with the unseen environment, the RNR-Map can still embed the useful information from images to the map and render images. The neural radiance fields of RNR-Map are conditioned on the latent codes. A pair of encoder and decoder is trained to make these latent codes from images of arbitrary scenes and reconstruct images using neural radiance fields. These pretrained encoder and decoder enable the generalization to unseen environments.

Third, RNR-Map is *real-time capable*. The majority of the present NeRF-based navigation methods require a significant time for inference because of required computation-heavy image rendering and rendering-based optimization

*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). *(Corresponding author: Songhwai Oh)*

steps. The RNR-Map is designed to operate fast enough not to hinder the navigation system. By directly utilizing the latent codes, we can eliminate the rendering step in mapping and localization. The mapping and image-based localization frameworks operate at 91.9Hz and 56.8Hz, respectively. The only function which needs rendering-based optimization is camera tracking, which can localize under odometry noises, and it operates at 5Hz.

To the best of our knowledge, the RNR-Map is the first method having all three of the aforementioned characteristics as a navigation map. The RNR-Map and its localization and navigation frameworks are evaluated in various visual navigation tasks, including camera tracking, image-based localization, and image-goal navigation. Experimental results show that the proposed RNR-Map serves as an informative map for visual navigation. Our localization framework exhibits competitive localization accuracy and inference speed when compared to existing approaches. On the image-goal navigation task, the navigation framework displays 65.7% success rate in curved scenarios of the NRNS [23] dataset, where the current state-of-the-art method [49] shows a success rate of 55.4%.

As RNR-Map finds a place based on the visual information of the map and the query image, we also consider a variant version of image-based localization. In real-world scenarios, there can be partial changes in the target place (changes in furniture placement, lighting conditions, ...). Also, the user might only have images from similar but different environments. We test the proposed localization framework in both cases. We find that the RNR-Map is robust to environmental changes and is able to find the most visually similar places even when a novel query image from a different environment is provided.

The contributions of this paper can be summarized as follows:

- We present RNR-Map, a novel type of renderable grid map for navigation, utilizing neural radiance fields for embedding the visual appearance of the environment.
- We demonstrate efficient and effective methods for utilizing the visual information in RNR-Map for searching an image goal by developing RNR-Map-based localization and navigation framework.
- Extensive experiments show that the proposed method shows the state-of-the-art performance in both localization and image-goal navigation.

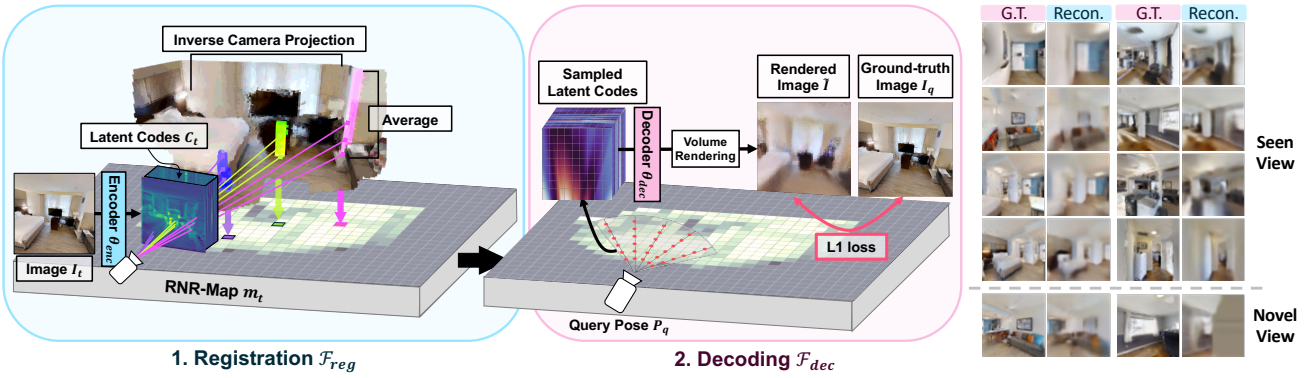
2. Related Work

Embodied AI with spatial memories. One of the central issues in recent embodied AI research is how to construct a useful memory for the embodied agent [16]. A memory that contains the navigation history, as well as information about the observed environment, is required for successful task execution. There is a large body of works using occupancy maps for visual navigation [10, 11, 17, 41]. An occupancy map expresses the environment in a grid form, and each grid cell has obstacle information about the corresponding region. An occupancy map represents the overall structure of the environment and can guide a robot to navigate the environment safely. There have been various kinds of research about building a spatial memory which contains additional information more than obstacles. The additional information can be object classes of the observed objects [7, 9, 21, 48], or implicitly learned useful information for a specific task [22, 24, 36, 38]. MapNet [24], SMNet [7] and ISS [38] have a similar mapping architecture with our method. Like our approach, they learn how to convert RGBD observations into useful latent information, and record it in the spatial memories using 3D inverse camera projection. Using the recorded latent information, MapNet [24] learns to localize the agent pose, and SMNet learns to generate a semantic map. ISS [38] is more related to ours since this method addresses scene generation and novel view image synthesis from a grid map. Our research is focused on how we can utilize such latent information for visual navigation. We develop localization and navigation frameworks which actively utilize the embedded visual information in RNR-Map.

Robotics with neural radiance fields. The neural radiance field (NeRF) [33] has gained significant popularity in various AI tasks. Not only in computer vision or graphics tasks, but NeRF is also adopted for robot applications in recent years. NeRF predicts the RGB color and density of a point in a scene so that an image from an arbitrary viewpoint can be rendered. This property enables pose estimation [1, 30, 31, 44] based on the photometric loss between the observed image and the rendered image, or manipulation of tricky objects [5, 12, 14, 25, 29]. A pretrained NeRF can also work as a virtual simulator, in which a robot can plan its trajectory [1] or can be used to train an action policy for the real-world [6]. Among the existing approaches, NICE-SLAM [55] is relevant to our work because it performs mapping and localization in arbitrary environments. NICE-SLAM builds a 3D implicit representation of the environment from image observations. The camera pose is inferred from optimizing the photometric loss between the observation image and the rendered image. Our method, on the other hand, places less emphasis on mapping quality, and it is designed for successfully completing navigation tasks. We focus on how the RNR-Map can be efficiently used for visual navigation, in terms of both speed and performance. The mapping and the target-searching function of RNR-Map are designed to operate fast enough to not hinder the other parts of the system. Also, the proposed RNR-Map method is generalizable in various environments without additional fine-tuning.

3. RNR-Map

A robot agent has an RGB-D camera, and also knows its odometry information. Here, the odometry means how



(a) **Reconstruction Framework** (b) **Examples of the rendered images**

Figure 1. (a) **Illustration of the reconstruction framework.** Two neural networks, encoder θ_{enc} and decoder θ_{dec} are used in this reconstruction framework. (b) **Examples of the rendered images.** Odd columns are the given observations, and even columns are reconstructed results. The proposed method can reconstruct the images from the novel view (the last row).

much the agent has moved from its previous position and we consider 3-DoF pose in this paper. At time t, the robot observes an RGBD image I_t and its relative pose $\Delta p_t = (\Delta x_t, \Delta y_t, \Delta a_t)$ from the previous pose (xy position and heading angle). By cumulating pose differences, the agent can determine its relative pose p_t from the start pose $p_0 = (0, 0, 0)$.

A pair of the pretrained encoder and decoder is used when building a RNR-Map. The training process of these encoder and decoder resembles the autoencoder method. However, unlike 2D images, autoencoding a 3D environment is not a trivial problem. We build a reconstruction framework as shown in Figure 1a. The encoder encodes an image and embeds the pixel features into the RNR-Map. We denote each embedded feature in a grid of RNR-Map as a latent code. A query pose is then provided, and the decoder samples latent codes along each camera ray corresponding to each pixel and renders the corresponding images. We present details of each part in the following section.

Registration $\mathbf{F}_{\text{reg}}, \mathbf{F}_{\text{map}}$. When an RGBD image $I_t \in \mathbb{R}^{H\times W\times 4}$ comes in, the encoder encodes the image into a same-sized feature $C_t \in \mathbb{R}^{H\times W\times D}$, where H and W refers to height and width of the image, respectively, and D refers to the channel size. First, each pixel $c_{h,w} \in \mathbb{R}^D$ in C_t is mapped to its corresponding 3D world position $[q_x, q_y, q_z]^T$ using the known camera intrinsic $\mathbf{K}$, the extrinsic matrix $[\mathbf{R}|\mathbf{t}]$ based on the agent pose, and the depth information $d_{h,w}$ of the pixel. The world position of each pixel is calculated using inverse camera projection, as follows:

$$\begin{bmatrix} q_x(h,w) \\ q_y(h,w) \\ q_z(h,w) \end{bmatrix} = d_{h,w}\mathbf{R}^{-1}\mathbf{K}^{-1}\begin{bmatrix} h \\ w \\ 1 \end{bmatrix} - \mathbf{t}\ . \tag{1}$$

Then we digitize each position by normalizing with the map resolution s, and get map position (u, v) as shown in (2). We aggregate the pixel features that belong to the same 2D map position, and average the aggregated features into a single feature vector. The pixel features are registered in the corresponding grid in the RNR-Map $m \in \mathbb{R}^{U\times V\times D}$, where U and V refer to the height and width of the RNR-Map, respectively. The number of averaged features at each 2D map position is also recorded in mask $n \in \mathbb{R}^{U\times V}$. We denote the element of m at the map position (u, v) by $m(u, v)$ and the mask at (u, v) by $n(u, v)$. The registered feature $m(u, v) \in \mathbb{R}^D$ is the latent code which contains visual information of the region corresponding to the map position (u, v). This process can be written as follows:

$$X_{(u,v)} = \left\{ c_{h,w} \in C_t \middle| u = \left\lfloor \frac{q_x(h,w)}{s} \right\rceil, v = \left\lfloor \frac{q_y(h,w)}{s} \right\rceil \right\}$$
$$m(u,v) = \frac{1}{n(u,v)} \sum_{c_i \in X_{(u,v)}} c_i, \quad n(u,v) = |X_{(u,v)}|. \tag{2}$$

The registration process F_{reg} includes the encoding of C_t, inverse projection, and feature registration. The F_{reg} is represented as:

$$m_t^l, n_t^l = F_{\text{reg}}(I_t, p_t; \theta_{\text{enc}}), \tag{3}$$

where θ_{enc} refers to the network parameters of the encoder. The registration process F_{reg} outputs a local map m_t^l and a local mask n_t^l. The local map m_t^l only contains the information from the image I_t, and this will be integrated with other local maps to form the global map m^g.[1] When multiple observations are given, we can use n to compute the average value from the original and new latent codes. We name this integration process F_{map}, which operates F_{reg} over multiple observations. F_{map} at time t with previous m_{t-1}^g is formulated as follows:

$$(m_t^g, n_t^g) = F_{\text{map}}(I_t, p_t, m_{t-1}^g, n_{t-1}^g; \theta_{\text{enc}})$$
$$m_t^g(u,v) = \frac{m_t^l(u,v)\cdot n_t^l(u,v) + m_{t-1}^g(u,v)\cdot n_{t-1}^g(u,v)}{n_t^l(u,v) + n_{t-1}^g(u,v)}$$
$$n_t^g(u,v) = n_t^l(u,v) + n_{t-1}^g(u,v)\ . \tag{4}$$

[1] For simplicity, m without any superscript refers to the global map (m^g) in the rest of the paper.

Decoding $\mathbf{F}_{\text{dec}}$. To make these latent codes contain visual information, we reconstruct the image observation from the latent codes. We use a decoder which has a structure similar to the generative scene network (GSN) [13] for rendering an RGBD image from the 2D latent map. Originally, GSN generates a random indoor floorplan from random variables. Then GSN proposed how to render images from the generated floorplan, based on locally conditioned radiance fields. Our approach involves designing the encoder θ_{enc} and the registration process F_{reg} to transform image observations into latent codes, which can be converted to the locally conditioned radiance fields. We utilize the locally conditioned radiance fields from GSN to render an image from m. Given the camera parameters, we can sample latent codes on points along the camera ray, corresponding to the pixel location which will be rendered in the camera. The sampled latent codes are converted into modulation linear layer-based locally conditioned radiance fields [4, 13]. The decoder is trained to render an RGBD image from the latent codes to be close to the image observations I_t. The reader can refer to [13] for a more detailed explanation about the rendering procedure.

By training the rendered RGBD images $\tilde{I}_t$ to be close to the original observation I_t, the encoder is trained to extract the visual information from the image. This mechanism is similar to training an autoencoder, to extract and summarize useful information from an image into a latent vector. We sample N images in a random scene and embed them into the RNR-Map. Then, we render each image from the final RNR-Map and compare them with the original images. The encoder and the decoder are trained using the following loss:

$$m_i^g, n_i^g = F_{\text{map}}(I_i, p_i, m_{i-1}^g, n_{i-1}^g; \theta_{\text{enc}}), \, i{=}1{:}N$$
$$\text{Loss}(\theta_{\text{enc}}, \theta_{\text{dec}}) = \frac{1}{N}\sum_{i=1}^{N} ||I_i - F_{\text{dec}}(m_N^g, p_i; \theta_{\text{dec}})||_1, \tag{5}$$

where θ_{enc} and θ_{dec} are weight parameters of the encoder and decoder, respectively.

Since the rendering process is conditioned on the latent codes from the image observation, our proposed reconstruction framework is capable of embedding and rendering arbitrary images from unseen environments. This leads to the generalizable property of RNR-Map. Also, the decoder can synthesize a novel view, different from the observed direction, based on m. Examples of reconstructed observations are shown in Figure 1b. The decoder can render images from novel viewpoints in the observed region, based on the latent codes. Note that the rendering boundary is limited to the observed 3D points. We can see that the decoder generates grey color for the unobserved regions, in the last row of Figure 1b. More examples of reconstructed images are provided in the supplementary material A, as well as the network architectures of the encoder and decoder B.1.

Mapping. After training, we can use F_{map} for logging visual information from the environment. Note that the rendering function F_{dec} is not needed for mapping. The RNR-Map is built incrementally during navigation, based on the odometry information and the known camera intrinsics. At time t, the agent obtains m_t and n_t using F_{map}, as formulated in (4). If the same 3D point in the environment is observed multiple times, F_{map} averages the latent codes based on the number of observation instances.

4. Localization

One of the crucial components of the navigation task is localization. In the following sections, we describe how RNR-Map is used for two localization tasks: *image-based localization* and *camera tracking*. Here, we consider 3-DoF pose (x, y, a) for localization.

4.1. Image-Based Localization

The implicit visual information of the latent codes in RNR-Map can provide useful clues for finding out which grid cell is the closest to the given target image I_{trg}. Inspired by the fast localization method in [24], we directly compare latent codes from the target image I_{trg} and the RNR-Map m for localization. We denote this image-based localization function as F_{loc}. Suppose that the target image is taken at the pose $p_{\text{trg}} = (x_{\text{trg}}, y_{\text{trg}}, a_{\text{trg}})$, and the corresponding map position is $(u_{\text{trg}}, v_{\text{trg}})$. F_{loc} outputs a heatmap $E \in \mathbb{R}^{U \times V}$ and the predicted orientation of the target a_{trg}. E highlights which grid cell corresponds to the target location $(u_{\text{trg}}, v_{\text{trg}})$, among the observed area in m.

The localization process F_{loc} is shown in Figure 2. The RNR-Map m is constructed from a partial observation about the environment. The query image is transformed into m_{trg} using $F_{\text{reg}}(I_{\text{trg}}, p_0; \theta_{\text{enc}})$. Here, we use origin $p_0 = (0, 0, 0)$ as an input to F_{reg} since the agent does not know the position of the target location. F_{loc} includes three convolutional neural networks, F_k, F_q, and F_E. F_k and F_q are for proccessing m and m_{trg} into m'_k and m'_q, respectively. We found it helpful to introduce neural networks (F_k, F_q) for better localization results. Then, we search query m'_q by convolving (cross-correlation, denoted as Conv) with m'_k. The query map m'_q is rotated into R different angles $\{0°, ..., 360° \times \frac{R-1}{R}\}$. $(m'_q)_r$ denotes $Rot_r(m'_q)$, where Rot_r represents the r-th from the R possible rotations. Each rotated query $(m'_q)_r$ works as a filter, and the output from the Conv is forwarded to F_E. F_E outputs $E \in \mathbb{R}^{U \times V}$ which highlights the most query-related region in m. Each pixel $e_{u,v} \in E$ in the heatmap represents the probability of the query image being taken at (u, v). Also, F_E has an additional head which predict the orientation of the target observation.

The overall localization process can be put together as:

$$F_{\text{loc}}(m, m_{\text{trg}}; \theta_{loc}) = F_E(\text{Conv}(F_k(m), \{F_q(m_{\text{trg}})_r\}_1^R)), \tag{6}$$

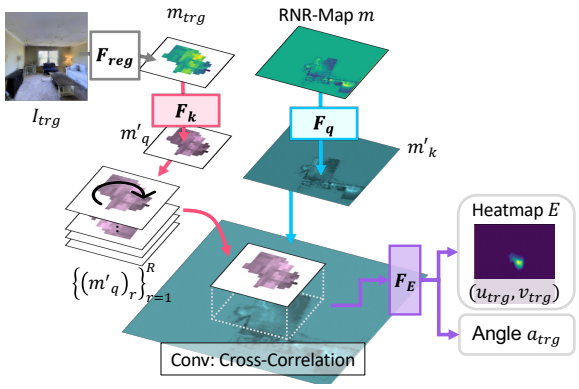


Figure 2. **Localization using $\mathbf{F}_{\mathrm{loc}}$.** A target observation can be localized by cross-correlation (Conv) between m and m_{trg}. Before the cross-correlation, each RNR-Map is forwarded to the CNN F_k and F_q. After Conv, F_E takes the output of the Conv and outputs a heatmap E which highlights the most plausible target area.

where θ_{loc} denotes the weight parameters of F_k, F_q, and F_E. Detailed explanations about the network architecture are provided in the supplementary material B.2.

We make a ground-truth Gaussian heatmap E_{gt} which highlights the ground-truth map location $(u_{\mathrm{trg}}, v_{\mathrm{trg}})$ of the query image. The representation of an orientation a_{trg} is discretized into 18 bins, and F_{E} is trained to select the appropriate bin using cross-entropy loss. The framework is trained to generate a distribution similar to the ground-truth heatmap E_{gt} and predicts the a_{trg}. The following loss term is used to train F_{loc}:

$$\begin{aligned}(\hat{E}, \hat{a}_{\mathrm{trg}}) &= F_{\mathrm{loc}}(m, m_{\mathrm{trg}}; \theta_{loc}) \\ \mathrm{Loss}(\theta_{\mathrm{loc}}) &= D_{KL}(E_{\mathrm{gt}}, \hat{E}) + CE(a_{\mathrm{trg}}, \hat{a}_{\mathrm{trg}}),\end{aligned} \tag{7}$$

where D_{KL} refers to KL divergence, and CE refers to cross-entropy loss. We provide quantitative and qualitative experimental results of F_{loc} in the supplementary material C.2.

4.2. Camera Tracking

During the navigation, the agent needs to be aware of its own pose to accurately record the observed visual observation. By cumulating odometry readings, the agent can calculate its relative pose to the start pose. However, in the real world, it is difficult to determine the accurate pose of a robot due to noises in odometry readings. The differential rendering function of F_{dec} can be used for adjusting the rough estimation of the agent pose p_t. The pose optimization is based on the photometric loss between the rendered image and the current observation. As the rendering process F_{dec} is differential, the pose of the agent p_t can be optimized with the gradient descent method. We name this camera tracking function as F_{track}. At time t, the agent has the previously estimated pose $\hat{p}_{t-1}$, and takes the odometry data Δp_t. The rough estimation of the current pose p_t can be calculated by simply adding Δp_t to the previous pose $\hat{p}_{t-1}$. $\bar{p}_t$ denotes such roughly estimated pose: $\bar{p}_t = \hat{p}_{t-1} + \Delta p_t$. Using

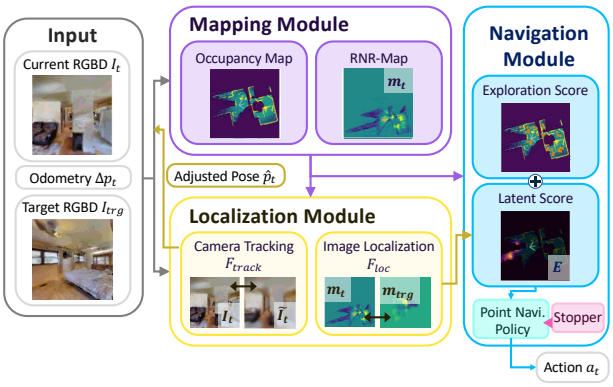


Figure 3. **Navigation System Overview.**

F_{track}, $\bar{p}_t$ is optimized to $\hat{p}_t$. The output of pose optimization can be derived by the following equation:

$$\hat{p}_t = F_{\mathrm{track}}(m_{t-1}, \bar{p}_t) = \arg\min_{\delta p_t} |F_{\mathrm{dec}}(m_{t-1}, \bar{p}_t + \delta p_t) - I_t|, \tag{8}$$

which minimizes the error between the current observation and the rendered image from the latent map. $\bar{p}_t$ is the initial value of the pose in the optimization process. By sampling a small subset of pixels from the image, we can make this optimization process fast enough to use it in navigation. We provide the accuracy and inference speed of the proposed camera tracking method in the supplementary material C.1.

5. Navigation

Now we describe how the RNR-Map and RNR-Map-based localization functions can be utilized in a navigation system. We consider the visual navigation task, especially image-goal navigation. The objective of image-goal navigation is to find the target location given the image taken from the target location. We build a visual navigation framework which includes F_{map} for mapping, F_{track} for localization, and F_{loc} for target searching. Figure 3 shows an overview of the proposed navigation system. The system consists of three modules, *mapping*, *localization*, and *navigation*. In the following section, we describe how each module works during navigation.

5.1. Mapping Module

The mapping module builds RNR-Map using the pre-trained encoder and F_{map}. At the start of the episode ($t = 0$), the mapping module transforms the target image I_{trg} into an RNR-Map m_{trg}. While maintaining m_{trg}, the module updates the current RNR-Map m_t using each image observation I_t with F_{map}. Also, the mapping module builds an occupancy map using depth information. This occupancy map is used for collision avoidance in a point navigation policy. The navigation module also uses this occupancy map to add more exploration property to the navigation system.

5.2. Localization Module

The localization framework described in Section 4 works as a localization module in the proposed navigation system. This localization module has two objectives during navigation. First, the localization module finds the most probable area which is similar to the target location. Second, considering a noisy odometry sensor, the localization module is needed to figure out the precise current pose of the agent. The image-based localization function F_{loc} and camera tracking function F_{track} are actively used in this module. With high probability, the target location may not be in the current RNR-Map m_t in the navigation scenario. Hence, F_{loc} for the navigation task is trained to find the region in m_t which is closer to the target location.

5.3. Navigation Module

The navigation module consists of three submodules: exploration, point navigation, and stopper.

Exploration. The objective of the exploration module is to select the most plausible region to explore. The module decides where to visit in order to search the target location, based on the probabilty heatmap E from F_{loc} in the localization module. We have adopted the concept from robot exploration [27, 34, 53, 56], which builds a generalized Voronoi graph on the occupancy map. We draw a Voronoi graph on the occupancy map and calculate visitation priority scores for each node of the created graph. Based on the scores, the exploration module selects the nodes to explore. Two types of scores are used for selecting exploration candidates, the latent score and the exploration score. The latent score is based on the heatmap E and represents how probable the node is near the target location. The exploration score of each node is simply calculated based on the values in the occupancy map. The occupancy map has three types of value: occupied, free and unseen. The exploration score of a node is proportional to the number of unseen pixels in the neighborhood of a node. The visitation priority of a node is determined based on the sum of the latent score and the exploration score.

Point navigation policy and Stopper. The point navigation policy is a simple occupancy-map-based path-following algorithm. When the exploration target node is selected, the point navigation module draws the shortest path to the target position. Following the path, the point navigation policy heuristically avoids obstacles using the occupancy map. The stopper module determines the arrival at the target location and calculates the relative pose from the target location. We employ a neural network F_{stop} which decides whether the agent is near the target location. This neural network is trained to output a binary value (1 if the target location is reached and 0, otherwise) based on the m_{trg} and m_t. For efficient target reaching, we adopted the recent last-mile navigation method [49] in stopper module. Based on keypoint matching, the relative pose between the target location and the current location is calculated using Perspective-n-Point [28] and RANSAC [19]. After F_{stop} detects the target location, the point navigation policy navigates to the target using the estimated relative pose. We provide detailed explanations about graph generation and exploration planning in the supplementary material G.

5.4. Implementation Details

The modules that require training are the encoder and decoder, and the neural networks used in F_{loc} and F_{stop}. We trained them using the same dataset. We have collected 200 random navigation trajectories from each scene in 72 Gibson [51] environments with the Habitat simulator [32]. The pair of encoder and decoder is first trained, then F_{loc} and F_{stop} are trained based on the pretrained encoder. Further implementation details (network architectures and training details) are provided in the supplementary material B.

6. Experiments

We have evaluated RNR-Map in both localization and navigation tasks. However, as the main objective of the proposed RNR-Map is visual navigation, we focus on analyzing the experiment results of the navigation task in the main manuscript. For localization tasks, we summarize the experimental results here and provide detailed information and analyses in the supplementary material C.

6.1. Localization

We have tested F_{track} and F_{loc} with other related baselines [24, 55]. In **camera tracking task**, the RNR-Map F_{track} shows high speed (5Hz) and accuracy (0.108m error) which are adequate for a real-time navigation system. More than localizing the current pose of the agent, the RNR-Map F_{loc} is able to locate the previously seen images (**image-based localization task**) with a high recall rate (inliers less than 50cm with 99%) in the recorded map. 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 that 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. We have tested the RNR-Map in both scenarios and observed that F_{loc} robustly localizes query images even if some object configuration changes. When 33.9% of the observed images have changed, F_{loc} localizes the query image with less than 50cm error in 97.4% of cases, and 20° error in 97.5% of cases. Also, when given a query image from a different scene, the localized location achieves 94.5% of visual similarity compared to the best possible visual similarity in the given scene. The examples of the image-based localization tasks are shown in Figure 4. We can see that F_{loc} finds visually similar places based on

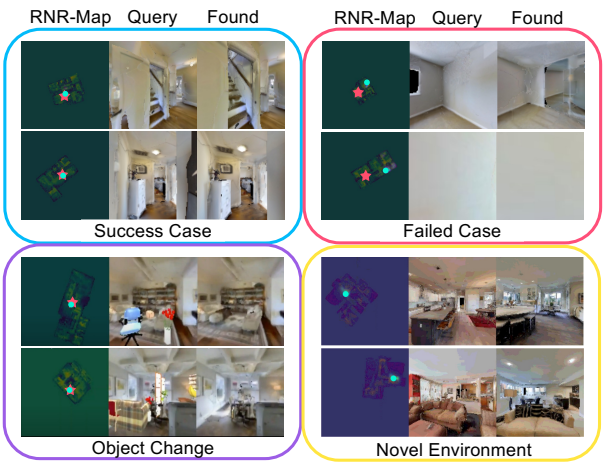


Figure 4. **Examples of image-based localization**. ★ location of the query image on RNR-Map, and ● is the location found by F_{loc}. More examples are provided in the supplementary material C.2 and C.3.

the RNR-Map, even when the environment has changed or given in a novel environment. Additionally, we found that the suggested method F_{loc} mislocates when there are multiple, similar locations in the environment or when the query has poor visual information. We provide more detailed experiments with baselines (MapNet [24], NICE-SLAM [55]) and examples in the supplementary material C.

6.2. Image-Goal Navigation

6.2.1 Baselines

We compare RNR-Map with learning-based agents using behavior cloning (**BC+RNN**) and reinforcement learning (**DDPPO** [50]). Also, to figure out the advantages of the RNR-Map over an occupancy map, we include the occupancy-map-based coverage method [10] as a baseline. The objective of this method is to visit all possible area in the given scene. We modified this method with the target distance prediction from [23], to make the agent reach the target when it is detected while exploring the environment (**ANS [10]+Pose Pred**). We also compare our method with the recent state-of-the-art image-goal navigation methods. **ZSEL** [2], and **OVRL** [52] are reinforcement learning-based methods which learn image-goal navigation task with specially designed rewards and the pretrained visual representation. **NRNS** builds a topological map and select the nodes to explore by predicting the distances between the target image and the node images with a distance prediction network. **SLING** is a last-mile navigation method which predict the relative pose of the target based on keypoint matching, after the target is detected. This method needs to be integrated with the exploratory algorithm such as NRNS or OVRL. Note that our method adopted this method, for efficient target reaching. The digits of the baseline DDPPO, OVRL, (NRNS,OVRL)+SLING are from [49] and their open-sourced code[2], and NRNS, ZSEL are from the original papers [23], [2], respectively.

6.2.2 Task setup

We have tested each method in the public image-goal navigation datasets from NRNS [23] and Gibson [51] with Habitat simulator [32]. Gibson dataset consists of 72 houses for training split, and 14 houses for the validation split. NRNS dataset consists of three difficulty levels (easy, medium, hard) with two path types (straight and curved). Each difficulty level has 1000 episodes for each path type, except for the hard-straight set (806). The objective of the image-goal navigation is to find the target place given the image of the target location. The agent only has access to the current RGBD observations and an odometry reading. We consider a noisy setting [10] where the odometry sensor and the robot actuation include noises as in the real world. The RGBD image observation comes from a directional camera with $90°$ of HFOV. A discrete action space is used in this paper with four types of actions: move forward $0.25m$. turn right $10°$, turn left $10°$, and stop. The maximum time step of each episode is set to 500. An episode is considered a success when the agent takes a stop action within $1m$ from the target location. Two evaluation metrics are used: success rate (SR) and success weighted by (normalized inverse) path length (SPL) [3], which represents the efficiency of a navigation path.

6.2.3 Image-goal navigation Results

RNR-Map helps efficient navigation. Table 1 shows the average SR and SPL of each method. We can see that the proposed navigation framework with the RNR-Map shows competitive or higher performance compared to the baselines, on image-goal navigation tasks. Many of the baselines (DDPPO, ZSEL, OVRL, OVRL+SLING) include reinforcement learning which is sample inefficient and computationally heavy, while having relatively simple representation about the environment. In contrast, the RNR-Map shows higher performances while only requiring an offline trajectory dataset for training neural networks. Based on this result, we argue that having a good internal representation of the environment and finding how to extract exploratory signals from such representation are important. An agent with an informative environmental representation can navigate well without the numerous inefficient interactions often required in reinforcement learning. Compared to baselines which have their own internal representation of the environment (NRNS, NRNS+SLING, ANS), our method shows a much higher performances in curved scenarios. From this result, we can infer that the RNR-Map indeed provides useful information for searching targets, more than coverage signals, and better than the existing methods. The ablation study shown in Table 2 also displays a similar trend. We ablated the main functions of the navigation framework F_{loc} and F_{track}, as well as noises. Without the latent score from

[2] https://github.com/Jbwasse2/SLING

	Straight								Curved							
	Easy		Medium		Hard		Overall		Easy		Medium		Hard		Overall	
	SR	SPL	SR	SPL	SR	SPL	SR	SPL	SR	SPL	SR	SPL	SR	SPL	SR	SPL
BC + RNN	39.4	27.9	25.7	15.9	13.3	9.0	26.1	17.6	26.4	12.6	20.3	10.5	8.4	4.8	18.4	9.3
DDPPO [50]	22.2	16.5	20.7	18.5	4.2	3.7	15.7	12.9	22.2	16.5	20.7	18.5	4.2	3.7	15.7	12.9
ANS + Target Pred [10]	68.8	55.1	54.0	30.3	42.4	22.9	55.1	36.1	48.0	21.0	46.0	20.5	31.3	14.6	41.8	18.7
NRNS [23]	64.1	55.4	47.9	39.5	25.2	18.1	45.7	37.7	27.3	10.6	23.1	10.4	10.5	5.6	20.3	8.8
ZSEL [2]	-	-	-	-	-	-	-	-	41.0	28.2	27.3	18.6	9.3	6.0	25.9	17.6
OVRL [52]	53.6	34.7	48.6	33.3	32.5	21.9	44.9	30.0	53.6	31.8	47.6	30.2	35.6	22.0	45.6	28.0
NRNS + SLING [49]	**85.3**	**74.4**	66.8	**49.3**	41.1	28.8	64.4	**50.8**	58.6	16.1	47.6	16.8	24.9	10.1	43.7	14.3
OVRL + SLING [49]	71.2	54.1	60.3	44.4	43.0	29.1	58.2	42.5	68.4	47.0	57.7	39.8	40.2	25.5	55.4	37.4
RNR-Map (ours)	76.4	55.3	**73.6**	46.1	**54.6**	**30.2**	**68.2**	43.9	**75.3**	**52.5**	**70.9**	**42.3**	**51.0**	**27.4**	**65.7**	**40.8**

Table 1. **Image-goal Navigation Result**. SR: Success Rate. SPL: Success weighted by Path Length.

			Easy		Medium		Hard		Overall	
Noise	F_{loc}	F_{track}	SR	SPL	SR	SPL	SR	SPL	SR	SPL
✘	✘	-	60.3	41.0	57.7	36.8	39.9	23.4	52.6	33.7
✘	✔	-	72.8	61.1	67.7	48.9	48.8	31.1	63.1	47.0
✔	✔	✘	74.6	53.6	64.0	40.9	40.8	23.6	59.8	39.4
✔	✔	✔	75.3	53.9	72.2	44.2	52.8	28.9	66.9	42.3

Table 2. **Ablation study**. The values are the average of the straight and curved scenarios.

Mapping F_{reg}	Tracking F_{track}	Localization F_{loc}	Rendering F_{dec}
91.9 Hz (10.9 ms)	5.0 Hz (200 ms)	56.8 Hz (17.6 ms)	14.7 Hz (68.0 ms)

Table 3. **Runtime analysis of RNR-Map.**

F_{loc}, the success rate and SPL dramatically drop. We also have tested the proposed method in MP3D [8] dataset, and observed similar results with Gibson dataset. We provide the results in the supplementary material E, and additional ablation study about the submodules of the navigation module is also provided in D.

F_{track} makes RNR-Map robust to noise. Comparing the third row and the last row of Table 2, we can infer that the pose adjusting function F_{track} helps the agent to find the image goal more effectively, even with the noisy odometry. The proposed method shows higher success rates in noisy settings, while SPL values are low. We believe this is from the randomness of noises, which helps the local navigation policy to escape from being stuck in a single location.

RNR-Map is real-time capable. We analyzed the runtime for each feature of RNR-Map and report them in Table 3.

The runtimes are measured on a desktop PC with a Intel i7-9700KF CPU @ 3.60GHz, and an NVIDIA GeForce RTX 2080 Ti GPU. We can see that each function of RNR-Map operates fast enough for real-time navigation, even when including NeRF-based rendering.

Navigation example. Figure 5 provides an example of an image-goal navigation episode. F_{loc} highlights two types of places, a place that looks similar to the target image and an explorable place like the end of the aisle, or doorway. We can see that the highlighted area at the start of the episode has a similar observation to the target image. First, the agent navigates to a similar-looking place but decided not to stop at the location. While navigating to another place that shows a high latent score, the agent has found the target location, and the latent score also changed to highlight the target. Additional qualitative results of navigation episodes are provided in the supplementary material F and video.

Figure 5. **Example of image-goal navigation episode.** The heatmap value of E from F_{loc} is highlighted on the map according to the colorbar on the left.

7. Conclusion

In this paper, we have proposed RNR-Map for visual navigation, which captures visual information about the environment. The RNR-Map is helpful for visual navigation in two ways: (1) The latent codes in the RNR-Map can provide rich signals to find the target location, given a query image. (2) The rendering property of the RNR-Map helps the agent to accurately estimate its pose based on a photometric error, in an unseen environment. The proposed method has outperformed other methods in image-based localization. Also, we have found that the image-based localization of RNR-Map is robust to environmental changes. In image-goal navigation tasks, the proposed method outperforms the current state-of-the-art image-goal navigation methods. Furthermore, the fast inference time of the RNR-Map shows its potential for real-world applications. However, the proposed method still has limitations. Once the observation images are embedded in grids, RNR-Map is hard to correct the past odometry error. We can consider applying loop closure using the proposed localization framework. Inspired by existing graph-based SLAM methods, a pose graph with local RNR-Maps can be leveraged to optimize poses, leading to consensus in the pixel renderings from the global RNR-Map.

References

[1] Michal Adamkiewicz, Timothy Chen, Adam Caccavale, Rachel Gardner, Preston Culbertson, Jeannette Bohg, and Mac Schwager. Vision-Only Robot Navigation in a Neural Radiance World. *IEEE Robotics and Automation Letters*, 7(2):4606–4613, 2022. 1, 2

[2] Ziad Al-Halah, Santhosh K. Ramakrishnan, and Kristen Grauman. Zero Experience Required: Plug & Play Modular Transfer Learning for Semantic Visual Navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 7, 8

[3] Peter Anderson, Angel X. Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, and Amir Roshan Zamir. On Evaluation of Embodied Navigation Agents. *arXiv preprint arXiv:1807.06757*, 2018. 7

[4] Ivan Anokhin, Kirill Demochkin, Taras Khakhulin, Gleb Sterkin, Victor Lempitsky, and Denis Korzhenkov. Image Generators with Conditionally-Independent Pixel Synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 4

[5] Valts Blukis, Taeyeop Lee, Jonathan Tremblay, Bowen Wen, In So Kweon, Kuk-Jin Yoon, Dieter Fox, and Stan Birchfield. Neural Fields for Robotic Object Manipulation from a Single Image. *arXiv preprint arXiv:2210.12126*, 2022. 2

[6] Arunkumar Byravan, Jan Humplik, Leonard Hasenclever, Arthur Brussee, Francesco Nori, Tuomas Haarnoja, Ben Moran, Steven Bohez, Fereshteh Sadeghi, Bojan Vujatovic, et al. NeRF2Real: Sim2real Transfer of Vision-guided Bipedal Motion Skills using Neural Radiance Fields. *arXiv preprint arXiv:2210.04932*, 2022. 2

[7] Vincent Cartillier, Zhile Ren, Neha Jain, Stefan Lee, Irfan Essa, and Dhruv Batra. Semantic mapnet: Building allocentric semanticmaps and representations from egocentric views. *arXiv preprint arXiv:2010.01191*, 2020. 2

[8] 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. 8, 12, 17, 19

[9] Devendra Singh Chaplot, Dhiraj Gandhi, Abhinav Gupta, and Ruslan Salakhutdinov. Object Goal Navigation using Goal-Oriented Semantic Exploration. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 1, 2

[10] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning To Explore Using Active Neural SLAM. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020. 1, 2, 7, 8

[11] Tao Chen, Saurabh Gupta, and Abhinav Gupta. Learning Exploration Policies for Navigation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019. 1, 2

[12] Qiyu Dai, Yan Zhu, Yiran Geng, Ciyu Ruan, Jiazhao Zhang, and He Wang. GraspNeRF: Multiview-based 6-DoF Grasp Detection for Transparent and Specular Objects Using Generalizable NeRF. *arXiv preprint arXiv:2210.06575*, 2022. 2

[13] 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. 4, 12

[14] Danny Driess, Zhiao Huang, Yunzhu Li, Russ Tedrake, and Marc Toussaint. Learning Multi-Object Dynamics with Compositional Neural Radiance Fields. In *Proceedings of the Conference on Robot Learning (CoRL)*, 2022. 2

[15] Danny Driess, Ingmar Schubert, Pete Florence, Yunzhu Li, and Marc Toussaint. Reinforcement learning with neural radiance fields. *arXiv preprint arXiv:2206.01634*, 2022. 1

[16] Jiafei Duan, Samson Yu, Hui Li Tan, Hongyuan Zhu, and Cheston Tan. A Survey of Embodied AI: From Simulators to Research Tasks. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2022. 1, 2

[17] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989. 1, 2

[18] Han Fang, Pengfei Xiong, Luhui Xu, and Yu Chen. CLIP2Video: Mastering Video-Text Retrieval via Image CLIP. *arXiv preprint arXiv:2106.11097*, 2021. 17

[19] 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. 6, 20

[20] Kevin Frans, Lisa B Soros, and Olaf Witkowski. Clipdraw: Exploring text-to-drawing synthesis through language-image encoders. *arXiv preprint arXiv:2106.14843*, 2021. 17

[21] Georgios Georgakis, Bernadette Bucher, Karl Schmeckpeper, Siddharth Singh, and Kostas Daniilidis. Learning to Map for Active Semantic Goal Navigation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022. 1, 2

[22] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive Mapping and Planning for Visual Navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[23] 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. 1, 2, 7, 8, 19, 20

[24] 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. 2, 4, 6, 7, 14, 15

[25] Jeffrey Ichnowski*, Yahav Avigal*, Justin Kerr, and Ken Goldberg. Dex-NeRF: Using a neural radiance field to grasp transparent objects. In *Conference on Robot Learning (CoRL)*, 2020. 2

[26] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised Contrastive Learning. 2020. 17

[27] Jonghoek Kim, Fumin Zhang, and Magnus Egerstedt. A provably complete exploration strategy by constructing voronoi diagrams. *Autonomous Robots*, 29(3):367–380, 2010. 6, 20

[28] 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. 6, 20

[29] Yunzhu Li, Shuang Li, Vincent Sitzmann, Pulkit Agrawal, and Antonio Torralba. 3D Neural Scene Representations for Visuomotor Control. In *Conference on Robot Learning*, 2021. 2

[30] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. BARF: Bundle-Adjusting Neural Radiance Fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021. 2

[31] Dominic Maggio, Marcus Abate, Jingnan Shi, Courtney Mario, and Luca Carlone. Loc-NeRF: Monte Carlo Localization using Neural Radiance Fields. *arXiv preprint arXiv:2209.09050*, 2022. 1, 2

[32] 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 *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019. 6, 7

[33] 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. 2, 12

[34] 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. 6, 20

[35] 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. 17

[36] Emilio Parisotto and Ruslan Salakhutdinov. Neural Map: Structured Memory for Deep Reinforcement Learning. In *International Conference on Learning Representations*, 2018. 2

[37] 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. 17

[38] Benjamin Planche, Xuejian Rong, Ziyan Wu, Srikrishna Karanam, Harald Kosch, YingLi Tian, Jan Ernst, and Hutter Andreas. Incremental Scene Synthesis. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2019. 2

[39] 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. 16, 17

[40] 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. 17

[41] Ziad Al-Halah Santhosh Kumar Ramakrishnan and Kristen Grauman. Occupancy Anticipation for Efficient Exploration and Navigation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020. 2

[42] 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. 20

[43] 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. 15

[44] Edgar Sucar, Shikun Liu, Joseph Ortiz, and Andrew Davison. iMAP: Implicit Mapping and Positioning in Real-Time. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021. 1, 2

[45] 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. 17

[46] 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. 20

[47] 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. 17

[48] Saim Wani, Shivansh Patel, Unnat Jain, Angel X. Chang, and Manolis Savva. MultiON: Benchmarking Semantic Map Memory using Multi-Object Navigation. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 1, 2

[49] 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. 2, 6, 7, 8, 19, 20

[50] 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. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020. 7, 8

[51] 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. 6, 7, 14, 17, 20

[52] 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. 7, 8, 19

[53] 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. 6, 20

[54] Tongjie Y Zhang and Ching Y. Suen. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):236–239, 1984. 20

[55] 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. 1, 2, 6, 7, 15

[56] 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. 6, 20

Appendix: Renderable Neural Radiance Map for Visual Navigation

We provide additional analyses and examples of the proposed method. The followings are included in this supplementary material: 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 7. 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 7.

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.

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.

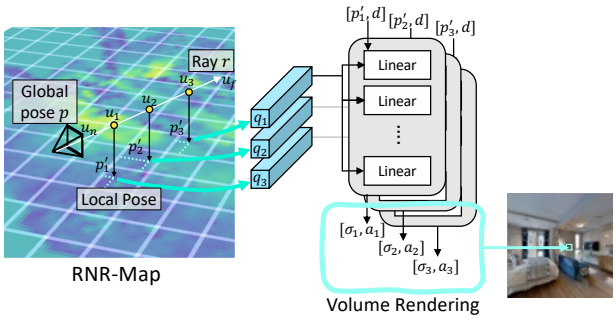


Figure 6. **Illustration of decoding process.**

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 [13]. The illustration of the decoding process is shown in Figure 6. 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 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 [33]:

$$
\begin{aligned}
c(\mathbf{r}, m) &= \int_{u_n}^{u_f} Tr(u)\sigma(\mathbf{r}(u), q)\mathbf{a}(r(u), \mathbf{d}, q)du \\
Tr(u) &= \exp(-\int_{u_n}^{u} \sigma(\mathbf{r}(u), q)du),
\end{aligned}
\tag{9}
$$

which is the same as the rendering function in GSN [13].

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

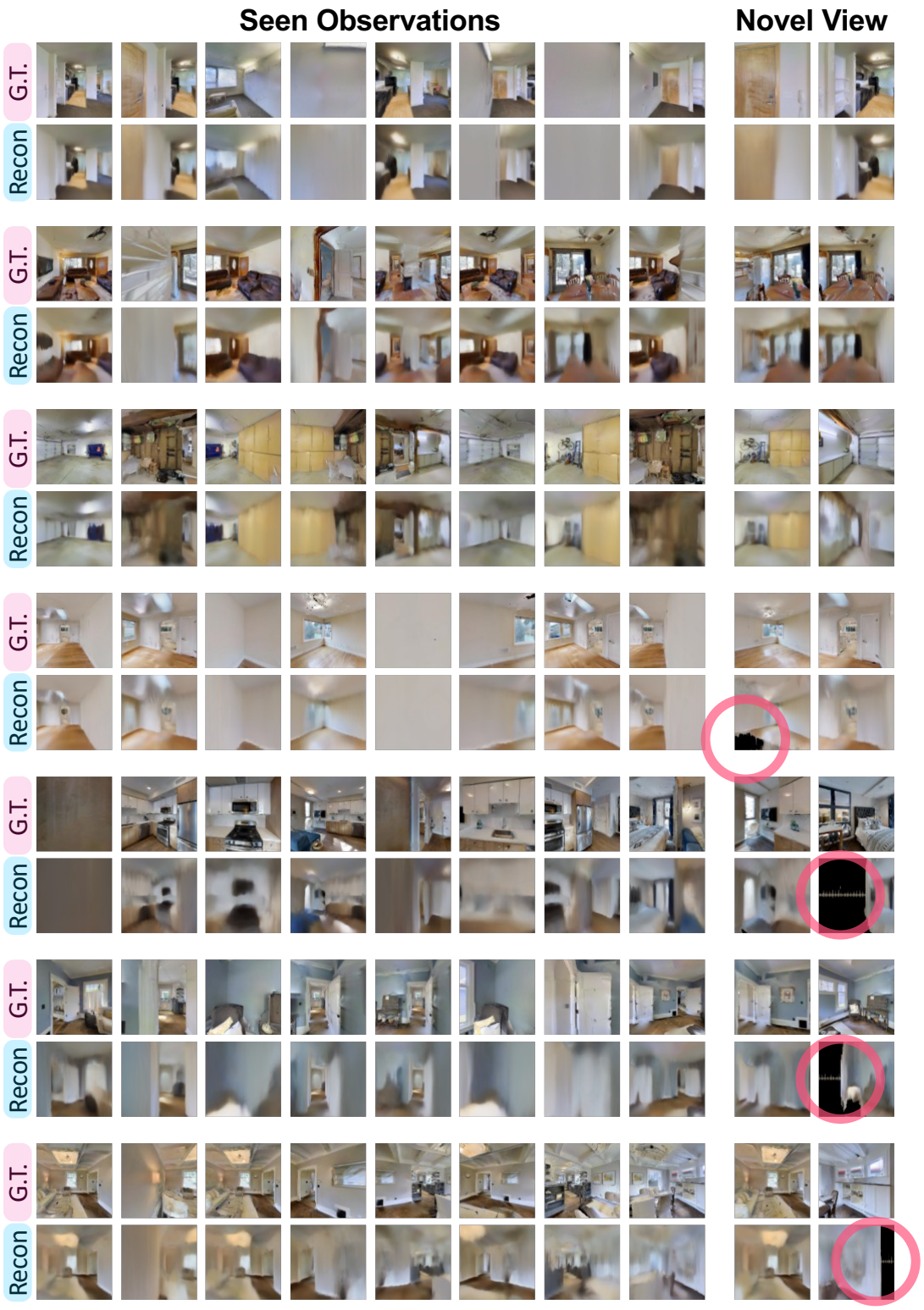


Figure 7. **Examples of the reconstructed images.** The odd rows are the ground-truth images, and the even rows are the reconstructed images from F_{dec}. In each row, eight images are embedded in RNR-Map. The last two columns are reconstructed from a novel view. Note that unobserved regions are rendered as an empty space (marked with red circle ❍). Each image is rendered with size 128×128.

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 8. 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 8. 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.

B.4. Training

We have collected 200 random navigation trajectories on 86 (training 72 + validation 14) scenes from the Gibson [51] 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).

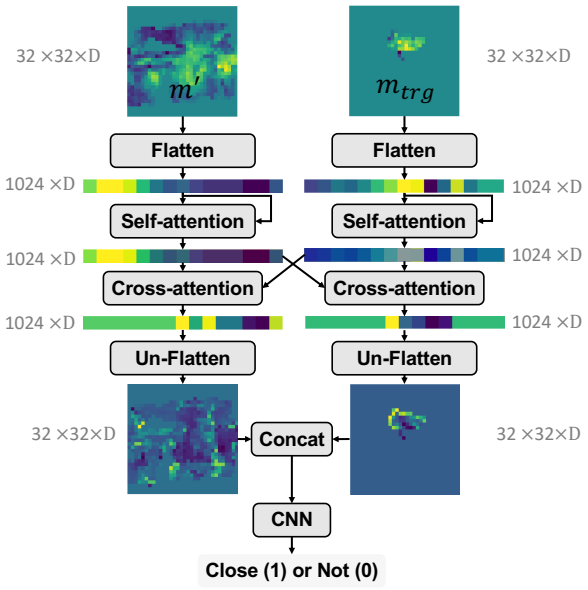


Figure 8. **Diagram of $\mathbf{F_{stop}}$.**

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** [24] proposed a method to build an allocentric spatial memory from egocentric observa-

	(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 [24]	0.332	**0.104**	8.6	15.4	21.2	0.027
Nice-SLAM [55]	0.941	6.743	85.9	93.9	94.3	24.520
Nice-SLAM* [55]	**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 4. **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.

tion. 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** [55] 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[3]. 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) [43] 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 4. 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 [55]. 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.

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

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 4. 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 [24] 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 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 $25cm \times 25cm$

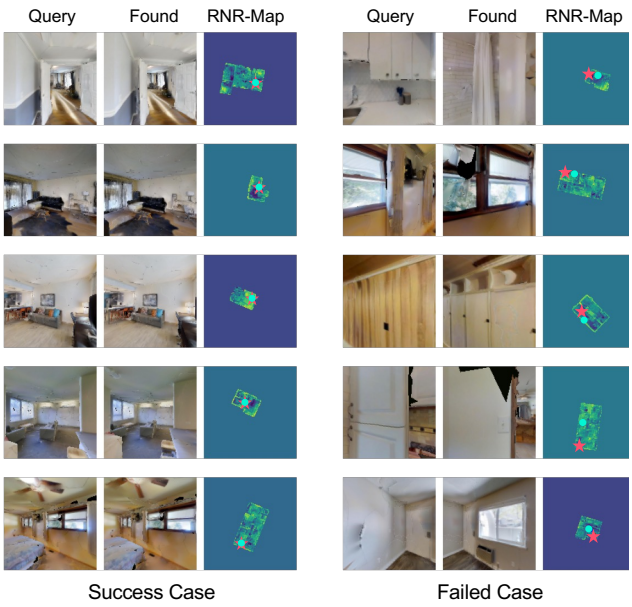


Figure 9. **Image-based Localization Examples.** The first column shows some examples of successful localization, and the second column shows some examples of failed localization. The query location is marked as red dot, and the predicted localization is marked as a blue dot.

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 5. **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.

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 9. 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.

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 5. 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 11a.

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** [39]: CLIP [39] 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 understand-

Search Method \ Similarity (%)	CLIP [39]	CL	SSIM	L1
Random	88.23	30.91	78.66	88.44
Max CLIP [39]	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 6. **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%.

ing of an image, and this model is widely used in various kinds of downstream tasks [18, 20, 35, 37, 40, 45]. We use CLIP for measuring visual similarities between images. The weight parameters from the publicly released model [4] are used. This model only takes RGB images.

- **CL (SupContrast) [26]**: We use a contrastive learning model [26] specially trained on the images from the Gibson [51] and MP3D [8] 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 [47] (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.

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 6. 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

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

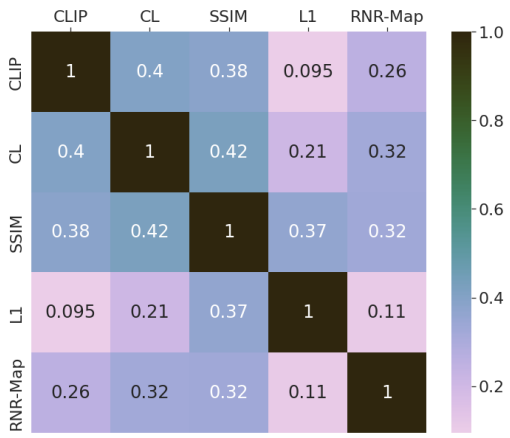


Figure 10. **Correlation matrix of the similarity metrics and the probability value from RNR-Map.**

based on a specific metric (second to the fifth row of Table 6). 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 6). The CL column of the Max CLIP row 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 10. 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 11b. In Figure 11b, 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 7, and each method is evaluated on Gibson-curved scenario from NRNS

(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 11. **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 7. **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.

Training Dataset	Method	Easy		Medium		Hard		Overall	
		SR	**SPL**	**SR**	**SPL**	**SR**	**SPL**	**SR**	**SPL**
MP3D	**NRNS [23]**	23.7	12.7	16.2	8.3	9.1	5.1	16.3	8.7
MP3D	**SLING+NRNS [49]**	43.2	19.7	32.5	15.1	22.1	9.9	32.6	14.9
Gibson	**OVRL [52]**	16.9	8.3	25.8	13.5	14.3	7.0	10.6	4.6
MP3D	**OVRL [52]**	52.4	35.2	42.6	26.3	29.7	16.9	41.6	26.1
Gibson	**SLING+OVRL [49]**	47.5	30.2	28.4	17.0	19.3	9.3	31.7	18.8
MP3D	**SLING+OVRL [49]**	**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 8. **Image-goal navigation results on MP3D [8]-curved scenarios in NRNS [23] dataset.** The first column shows the training dataset each method has trained on.

dataset [23] 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 7. 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 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 8. The experiments are done without noise setting, and we report the digits from [23], [49] 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 [51] 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 12. 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 12a and 12b. 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 13 shows the process of graph generation. This method is inspired by robot exploration literature [27, 34, 53, 56], 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`) [46]. This function is based on the image thinning algorithm proposed in [54]. 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.

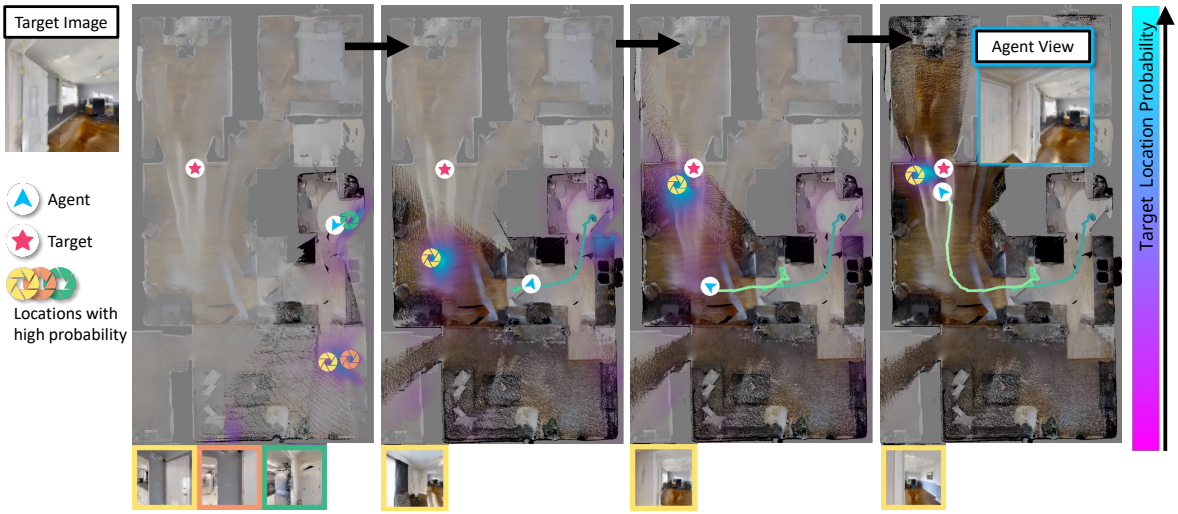
Exploration Score The exploration module evaluates each node in the graph with two criteria. The first one is the latent score, which is from the heatmap from F_{loc} in the localization module. This heatmap highlights the place related to the target image. The second one is the exploration score, which 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 13. 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 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 [5]. 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 [49]. 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 [42] between the current observation and the target image. If a sufficient number of keypoints are matched, we can infer the relative pose between the target location and the current position using the depth information. As proposed in [49], we use Perspective-n-Point [28], and RANSAC [19]. 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.

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

(a) **Example 1.**



(b) **Example 2.**

Figure 12. **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.

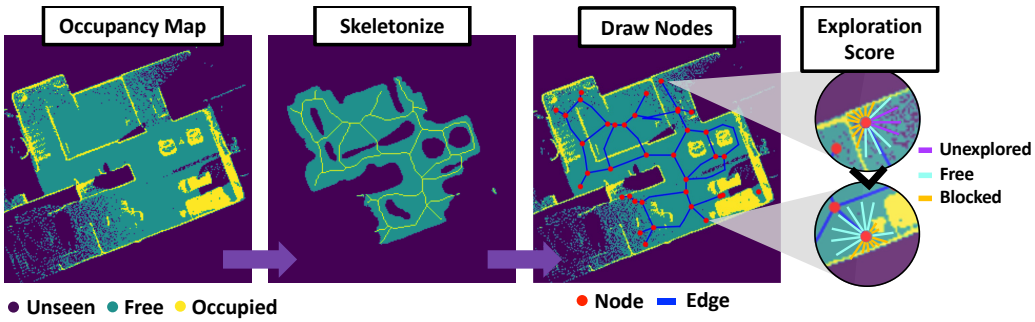


Figure 13. **Overview of the graph generation.** We take the free area of the occupancy map and skeletonized the image. Then, we draw a graph on the skeleton. Each node has an exploration score, which is calculated based on the number of unseen pixels in the neighborhood.