SAD-SLAM: Sign-Agnostic Dynamic SLAM

Group 11

R09522848 王琮文， R10945068 詹易玹， R11946012 王奕方

1. Introduction

Dense visual simultaneous localization and mapping (SLAM) is a crucial aspect of 3D computer vision with numerous applications, such as autonomous driving and indoor robotics, as well as mixed reality. With the advancement of deep learning, there has been a surge in research focusing on the development of SLAM systems using this new technology. However, deep-learning based SLAM still falls behind traditional SLAM with many problems remaining to overcome, such as real-time mapping and tracking and the ability to scale up to large scenes.

After surveying related literature, we discovered a paper published in CVPR 2022 proposes NICE-SLAM [1] to address the above-mentioned two issues. As illustrated in Fig. 1, NICE-SLAM takes an RGB-D image stream as input and outputs both the camera pose and a learned scene representation in the form of a hierarchical feature grid. It optimizes the scene representation and camera pose in an alternating fashion by backpropagating the image and depth reconstruction loss through a differentiable renderer.

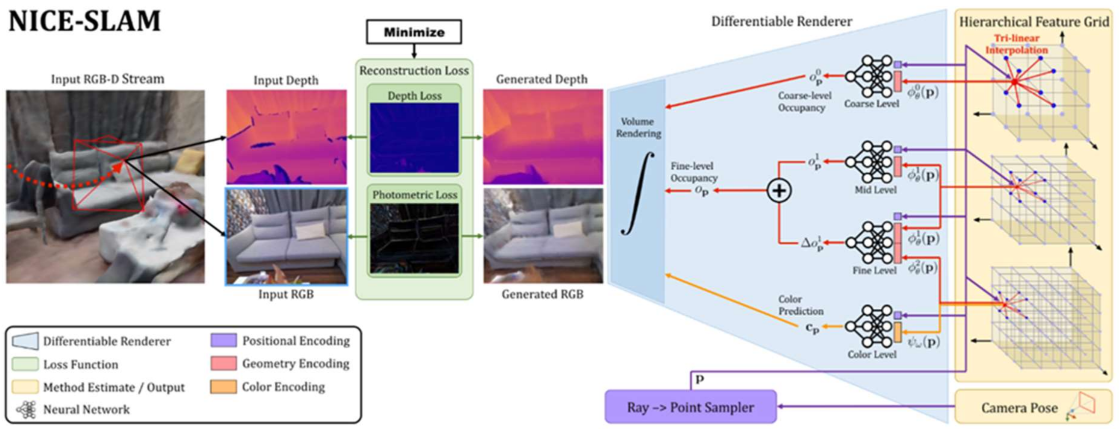


Fig. 1. The architecture of NICE-SLAM [1].

We identify two areas of improvement in the system: reducing the error in mapping and tracking and handling dynamic objects. The pretrained multilayer perceptron (MLP) decoder used in NICE-SLAM as a differentiable renderer still has room for optimization in terms of mesh reconstruction. Moreover, NICE-SLAM is not robust in dynamic scenes because it only filters pixels with large depth/color re-rendering loss during tracking, which are assumed to be caused by dynamic objects. Therefore, we propose a new model called "SAD-SLAM," which aims to improve upon NICE-SLAM by using sign-agnostic optimization to reduce reconstruction error and by removing dynamic objects from the scene captured.

2. Methodology

2.1. Mapping and Tracking Optimization

2.1.1 Dataset

We evaluated on two datasets, ScanNet [2] and Replica [3]. ScanNet contains multiple large real-world indoor scenes with rich annotated objects, while Replica includes highly photo-realistic 3D indoor scene mesh in a simulated environment at room and building scale. Some examples of ScanNet and Replica are shown in Fig. 2.

Fig. 2. ScanNet (left) and Replica (right) dataset [1].

2.1.2 Method

NICE-SLAM uses three different fixed MLPs to decode the grid features into occupancy values. The decoder is pertained from ConvONet [4], which consists of a convolutional neural network (CNN) encoder and an MLP decoder. After training using binary cross entropy loss, only the decoder is used in NICE-SLAM for reconstruction.

We introduce sign-agnostic (SA) implicit surface optimization on decoder from SA-ConvONet [5]. SA-ConvONet, whose architecture is shown in Fig. 3, deals with surface reconstruction from point clouds. It uses the pretrained model, ConvONet, same as NICE-SLAM. Moreover, in order to increase the generality of unseen inputs during inference, it optimizes the occupancy network through test-time training by considering the surface normal in each local field. It gets sign (surface normal) information of each unsigned point and maximizes the consistency between iso-surface and unoriented point clouds with an unsigned cross-entropy loss (Eq. 1).

$$L_{UCE} = \sum_{q \in s \cap s^c} BCE\left(O(q), O^{\dagger}(q)\right) \tag{1}$$

During inference, SA-ConvONet randomly samples from ground truth surface point set and non-surface random point set by adding a noise on the surface point. The surface is defined with a 0.5 level set, as the signed occupancy of the surface point set (S) is targeted as 0.5 and the signed occupancy of the non-surface point set (S^c) is targeted as 1.0. The occupancy of the prediction and target are given by (Eq. 2) and (Eq. 3).

$$O(q) = sigmoid\left(\left|f\left(p, \phi(p)\right)\right|\right) \in [0.5,1) \tag{2}$$

$$O^{\dagger}(q) = \begin{cases} 0.5\ , & if\ q \in S \\ 1.0\ , & if\ q \in S^c \end{cases} \tag{3}$$

Borrowing from the idea used in SA-ConvONet, we implemented the sign-agnostic optimization in NICE-SLAM during mapping. During every mapping optimization, we sampled surface points from each selected keyframe in the global keyframe list. Then we calculated the unsigned cross-entropy loss, and optimized the middle-level decoder and the fine-level decoder to obtain sign information of each point. As for hyperparameters, we ran 5 iterations during every mapping, and used an Adam optimizer with a learning rate 0.00003 that decays 0.3 in every 4 iterations. Each time, we loaded the pretrained decoder model of NICE-SLAM first to achieve the generality of input points.

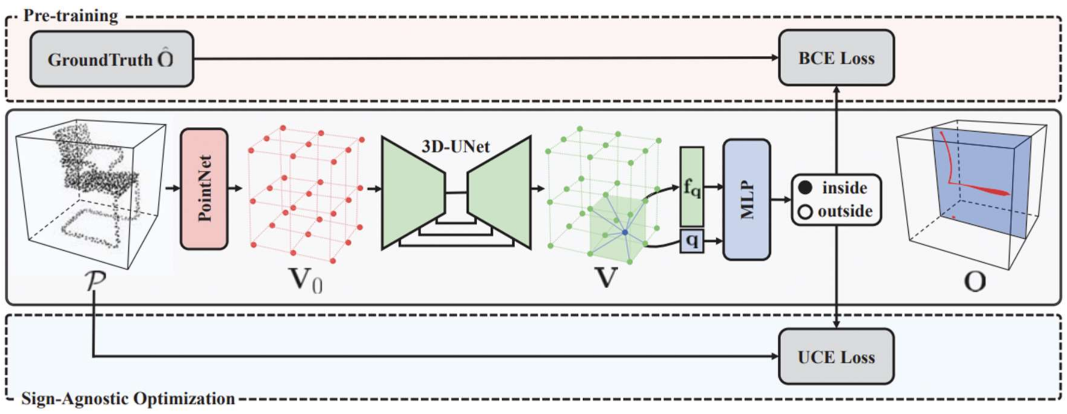


Fig. 3. The architecture of SA-ConvONet [4].

2.2 Dynamic Objects Removal

2.2.1 Dataset

We used the Bonn RGB-D Dynamic Dataset [6], on which NICE-SLAM was not tested. This dataset contains image sequences with dynamic objects and the ground truth camera poses. We selected the image sequence named “rgbd_bonn_crowd” in the dataset, which captures a scene sometimes blocked by people walking through, for model evaluation.

2.2.2 Method

Because NICE-SLAM updates the model by calculating the loss between original RGB-D images and generated RGB-D images, we expected removing dynamic objects from input images may improve the ability of the model to handle dynamic scenes. Inspired by the approach used in DynaSLAM [7], we synthesized a realistic image without moving content if dynamic objects were detected in the frame and used it as model input. The procedure involves instance segmentation, tracking and background inpainting.

(1) Instance segmentation: Using the instance segmentation model to remove potentially dynamic content, e.g., people.
(2) Tracking: Determining the transform matrix by calculating the relative camera pose between the current frame and previous keyframe.
(3) Background inpainting: Given the relative position, projecting the RGB and depth channels from the previous keyframes into the dynamic segments of the current frame. The new RGB values were calculated by estimating the homography between the two frames and copying the values from the corresponding segment in the previous keyframes. The depth values were computed by transforming the vector from the scene point to the point on the image plane of the previous keyframe in the world coordinate system and calculating the length of the transformed vector.

Palazzolo et al. argued that Mask R-CNN is easy to generalize to other tasks, e.g., allowing us to estimate human poses in the same framework [8]. We used the pretrained Mask R-CNN model with a ResNet-50-FPN backbone for instance segmentation. Mask R-CNN is a two-stage model consisting of a backbone network, a region proposal network (RPN), and two parallel branches for classification and mask prediction (Fig. 4). The backbone network is typically a CNN that is used to extract features from the input image. The RPN is used to generate a set of candidate object regions, or bounding boxes, in the image. The classification branch is used to predict the class of each object, and the mask branch is used to predict a binary mask for each object.

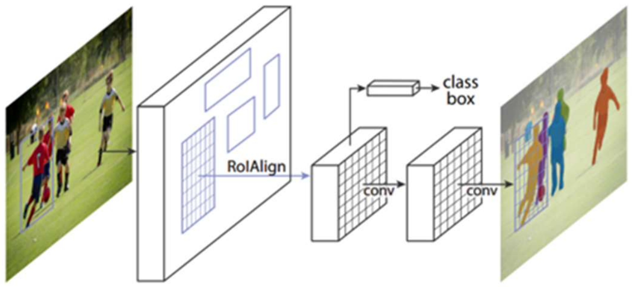


Fig. 4. Mask R-CNN.

3. Results

3.1 Mapping and Tracking Optimization

We followed the quantitative metrics evaluated on NICE-SLAM. For tracking, we evaluated the optimization using average trajectory errors (ATE) with root mean square error (RMSE) on both ScanNet and Replica. For mapping, we calculated the mesh reconstruction error on Replica by the ground-truth mesh, where the unseen region is culled. We evaluated on 3 metrics:

(1) Accuracy (unit: cm) is the average distance between sampled points from the reconstructed mesh and the nearest ground-truth point

(2) Completion (unit: cm) is the average distance between sampled points from the ground-truth mesh and the nearest reconstructed

(3) Completion ratio (unit: %) is the percentage of points in the reconstructed mesh with Completion under 5 cm.

3.1.1 ScanNet

As shown in Fig. 5, the ATE RMSE for NICE-SLAM (N) and NICE-SLAM + sign-agnostic optimization (N+SA) is 0.037 and 0.029 respectively. As we can see from the trajectory, N+SA obviously optimizes the camera pose to fit the ground truth better. It means that the optimization of the middle-level and fine-level decoder during mapping enhances the performance of trajectory simultaneously.

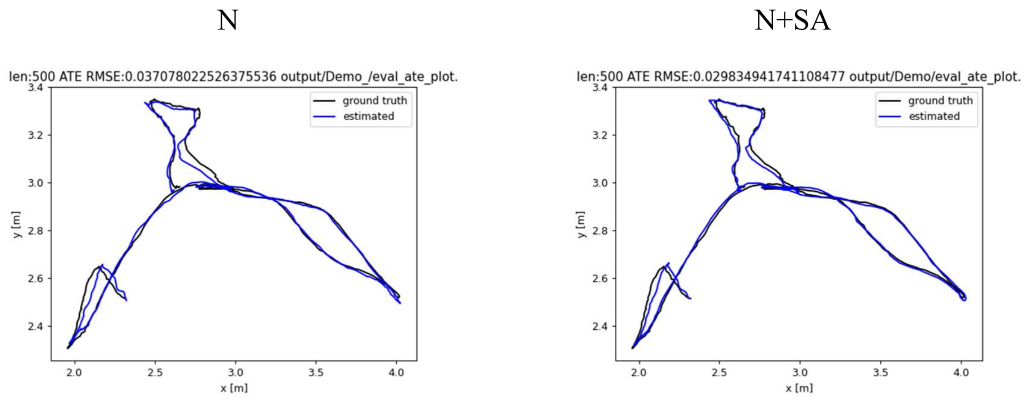


Fig. 5. The average trajectory errors (ATE) of NICE-SLAM (N), NICE-SLAM + sign-agnostic optimization (N+SA) on ScanNet.

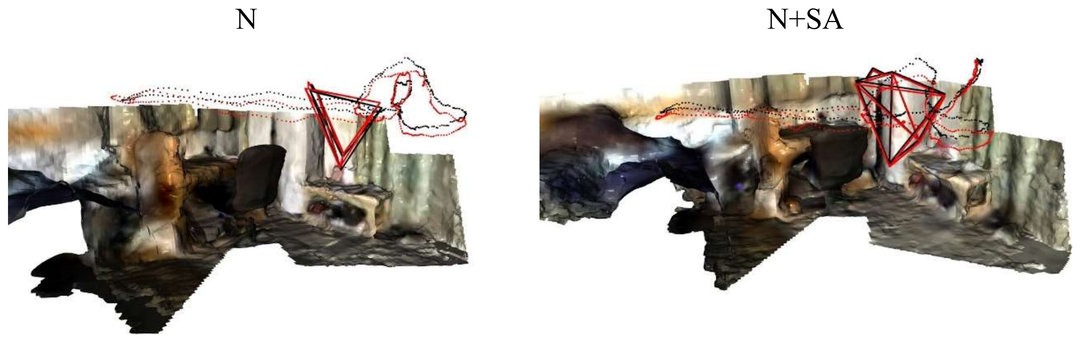


Fig. 6. Scene mesh reconstruction using NICE-SLAM (N), NICE-SLAM + sign-agnostic optimization (N+SA) on ScanNet.

Besides, N+S can reconstruct a scene with sharper and more detailed geometry compared to the N method, as it can be observed from the shape of the chair in Fig. 6.

3.1.2 Replica

As shown in Fig. 7, the ATE RMSE for NICE-SLAM (N) and NICE-SLAM + sign-agnostic optimization (N+SA) is 0.014 and 0.020 respectively. We can find that the N method already performs well as ATE RMSE is small. Because the highly photo-realistic 3D indoor scene mesh is generated in Replica dataset, this flawless data input enables the accurate prediction of the trajectory. On the other hand, N+SA also performs well and predicts the estimated trajectory close to the ground truth. The ATE RMSE of N+SA is slightly larger than that of NICE-SLAM due to the deviated trajectory in the circle region in Fig. 7.

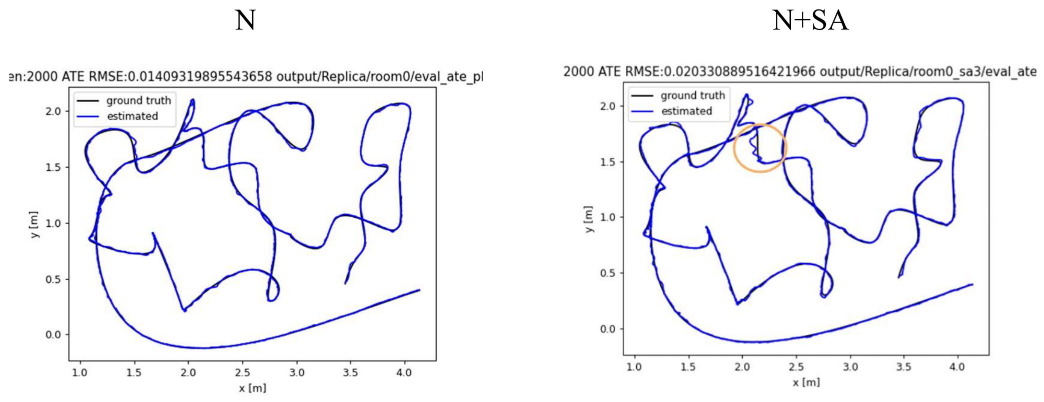


Fig. 7. The average trajectory errors (ATE) of NICE-SLAM (N), NICE-SLAM + sign-agnostic optimization (N+SA) on Replica.

Table 1 shows that N+SA method has lower accuracy and completion compared to the N method. Additionally, the completion ratio of N+SA is higher than the N method. We conclude that sign-agnostic optimization can improve not only the scene mesh reconstruction but also the performance of tracking and mapping in SLAM.

Table 1. Scene mesh reconstruction metrics using NICE-SLAM (N), NICE-SLAM + sign-agnostic optimization (N+SA) on Replica.

	N	N+SA
Accuracy (cm)	2.82	**2.52**
Completion (cm)	2.75	**2.74**
Completion Ratio (<5cm %)	91.50	**91.52**

3.2 Dynamic Objects Removal

First, to verify the correctness of our implementation, we manually selected two frames that capture a similar region of the scene from the image sequence. As shown in Fig. 8, in the current frame, after segmentation and background inpainting, two people no longer exist and the desk in the background is correctly inpainted in both RGB and depth channels by projecting the values from the corresponding segment in the previous frame.

Next, we implemented dynamic object removal in NICE-SLAM. Unfortunately, we discovered that the computational complexity of the modified NICE-SLAM system significantly increased as the RGB-D values were recalculated pixel by pixel. Our modified system with dynamic object removal, especially for the stage of inpainting, slows down the running speed. Therefore, we conducted experiments using only the Mask R-CNN for dynamic object segmentation and then filtering out the detected dynamic segment in the frame when calculating the depth/color re-rendering loss.

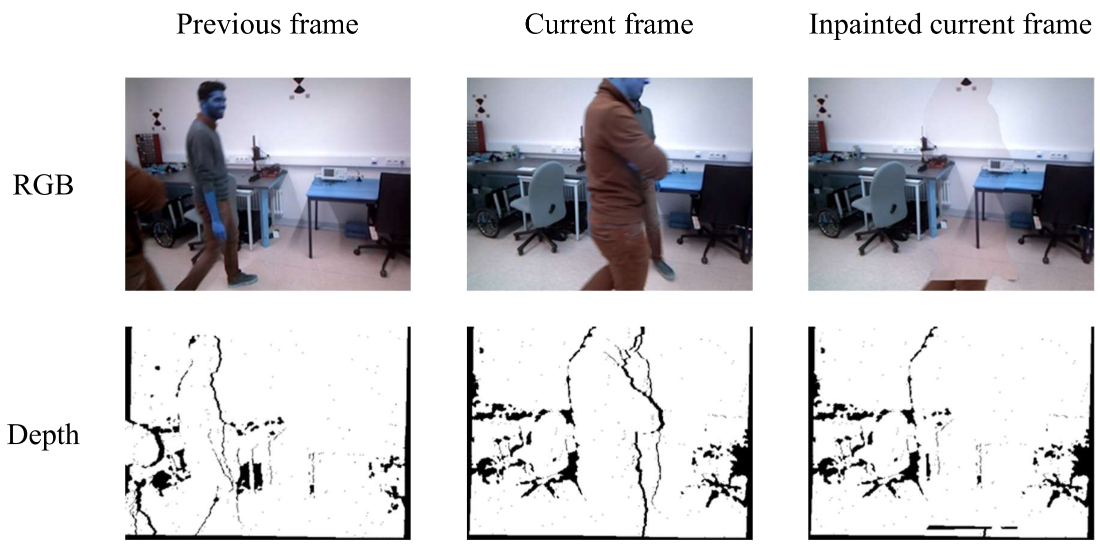


Fig. 8. Instance segmentation and background inpainting.

The following result comes from the first ten percent of the testing image sequence. We evaluated the trajectory error of the three methods. The results shows that the ATE RMSE is 0.69, 0.131, 0.149 for NICE-SLAM (N), NICE-SLAM + Mask R-CNN (N+M) and NICE-SLAM + Mask R-CNN + background inpainting (N+M+BI), respectively (Fig. 9). After incorporating Mask R-CNN into NICE-SLAM, the error in camera pose estimation is significantly reduced. We compared the generated RGB-D images produced by the N and N+M method during the mapping process. An example of these images is shown in Fig. 10. It is clear that when dynamic objects (in this case, people) are present in the frame, the N+M method generates a more accurate scene image without people than the N method. This implies NICE-SLAM with Mask R-CNN can learn the scene representation more accurately and robustly.

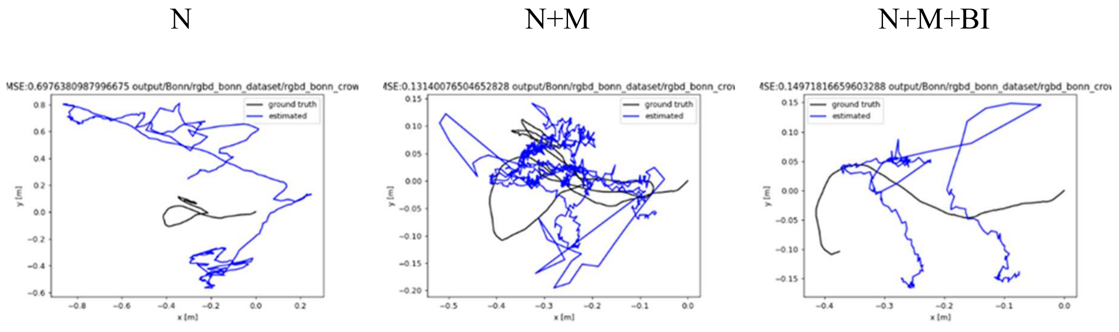


Fig. 9. The average trajectory errors (ATE) of NICE-SLAM (N), NICE-SLAM + Mask R-CNN (N+M) and NICE-SLAM + Mask R-CNN + background inpainting (N+M+BI).

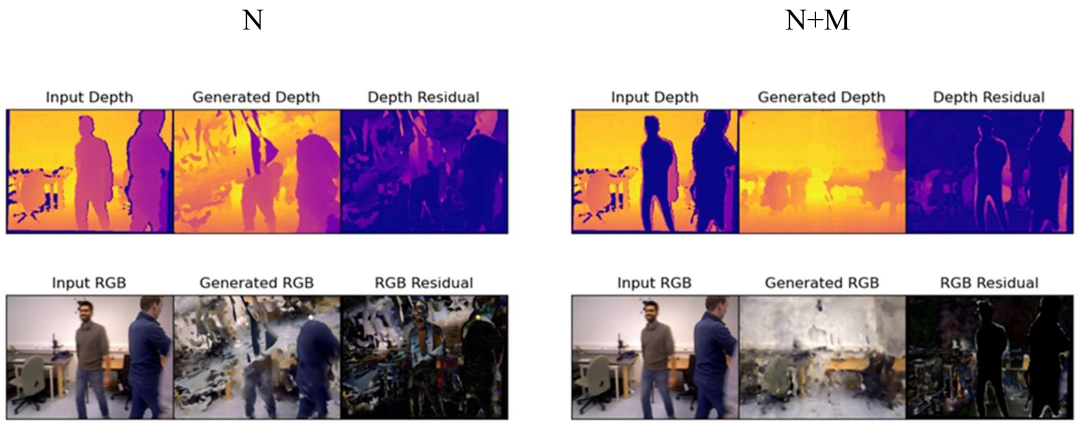


Fig. 10. The generated RGB-D images of NICE-SLAM (N) and NICE-SLAM + Mask R-CNN (N+M) during mapping.

Compared to N+M, N+M+BI has larger ATE RMSE. This result conflicts with our hypothesis that N+M+BI would have performance similar to that reported in the original NICE-SLAM paper as its input is assumed to be identical to an image sequence captured in the scene without moving objects. In our experiments, we chose only one image from the set of keyframes computed by NICE-SLAM as the source for background inpainting to reduce computational complexity. Fig. 11 shows the inpainted frames at timestamp 230 and 235 during evaluation by two keyframe selection strategies, i.e., using the earliest or the latest image in the set of keyframes. Background inpainting sometimes produces incorrect value filling and this may be one potential reason for the slightly higher ATE RMSE of N+M+BI compared to N+M. In addition, we observed keyframe selection strategy used greatly impacted the quality of inpainted images. In our tests, using the earliest keyframe as the source for background inpainting produced better results than using the latest keyframe.

As shown in Fig. 12, the original NICE-SLAM is unable to successfully reconstruct the scene mesh when there are dynamic objects in the scene. In particular, large deviations happen when the frame captures something that did not appear in previous frames (in this case, people). While N+M is unable to reconstruct the entire scene accurately, it correctly reconstructs the unblocked regions of the scene and ignore the dynamic objects. This suggests that there may be potential for improving the accuracy of the N+M method by combining it with more accurate and computationally efficient background inpainting techniques.

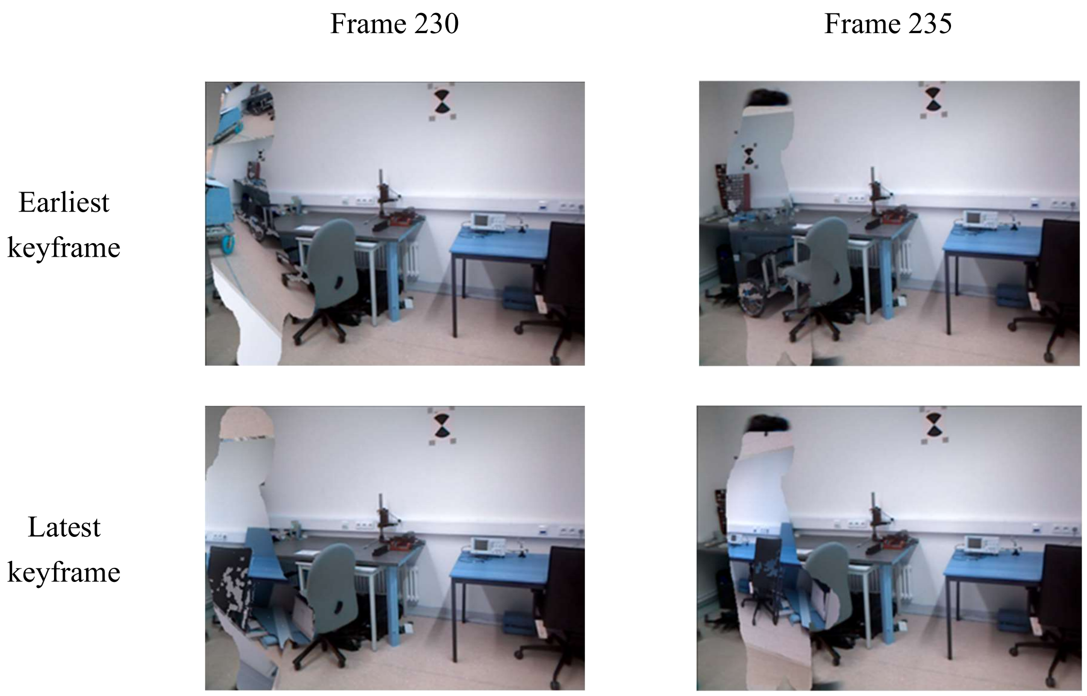


Fig. 11. Inpainted frames.

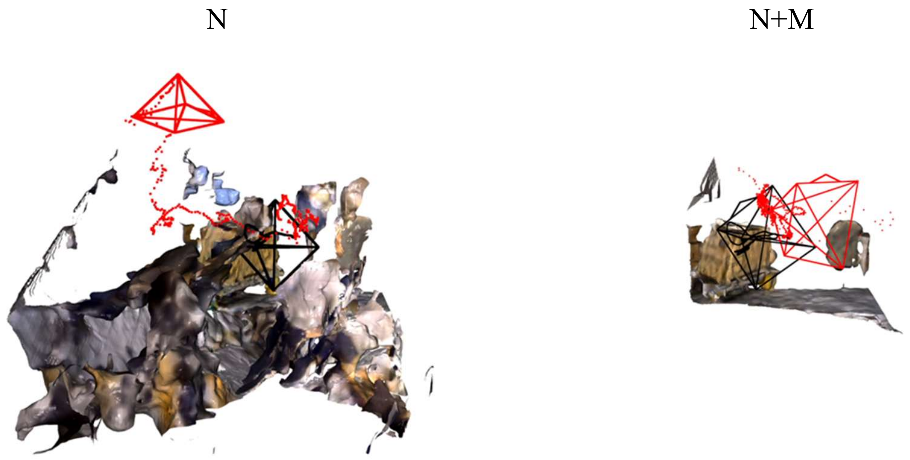


Fig. 12. Scene mesh reconstruction using NICE-SLAM (N) and NICE-SLAM + Mask R-CNN (N+M).

3.3 Experiments on Self-Captured RGB-D Dataset

3.3.1. Camera Calibration

An RGBD camera is a type of depth camera that provides both depth (D) and color (RGB) data as the output in real time. The term RGB refers to the color model in which the red, green, and blue primary colors of light are added to create different colors that we perceive, while depth information is retrievable through a depth map which is created by a 3D depth sensor such as a stereo sensor or time of flight sensor. In our research, Intel RealSense depth camera D435i was used to generate color and depth data. Before we start our experiment, it is necessary to estimate the parameters required to relate 2D points in an image plane with 3D points in the world scene the camera is viewing. After the process of camera calibration is finished, the camera intrinsic matrix can be estimated:

$$\begin{bmatrix} 617.2 & 0.0 & 315.2 \\ 0.0 & 609.5 & 218.1 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{4}$$

And the distortion coefficients are [-0.078099, 1.391195, -0.008579, -0.000432, -4.555212].

3.3.2. NICE-SLAM

We designed three experiments to compare the above-mentioned techniques. The first experiment is to observe the performance of the original NICE-SLAM algorithm [1]. Then, in Section 3.3.3, we continue the experiment by adding the sign-agnostic (SA) optimization. Finally, the combination of NICE-SLAM, SA, and Mask R-CNN will be implemented, and compared in our third experiment in Section 3.3.4.

First, we demonstrated the performance of the original NICE-SLAM in Fig. 13. There are two parallel desks in the laboratory, and the shape of them can be clearly observed with the 3D reconstruction from NICE-SLAM. However, some details are not constructed well. For example, the mapping of wall bookshelves does not match the real environment, so the 3D space of our laboratory environment is not well-constructed. This phenomenon implies that NICE-SLAM cannot accurately construct a mapping of the scene with detailed and complicated information, such as large wall bookshelves with many objects.

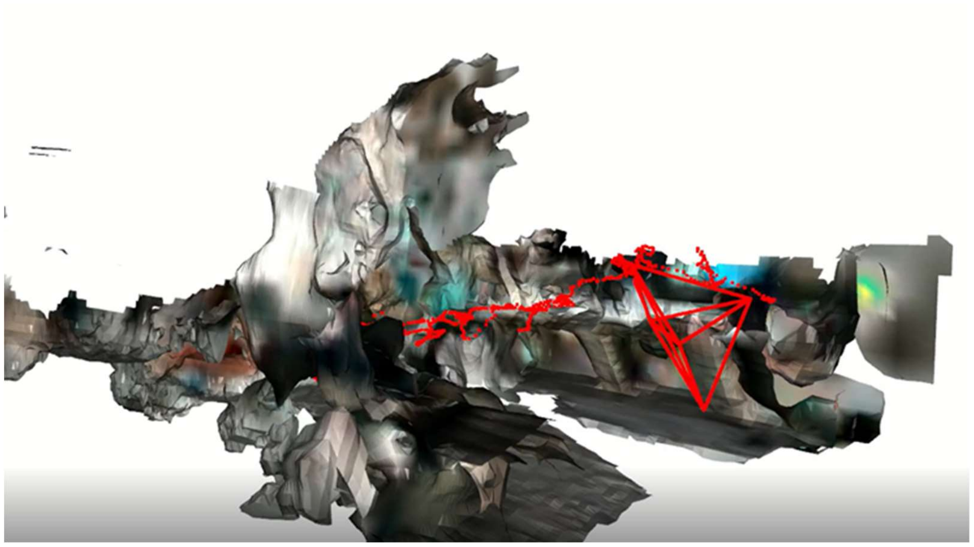


Fig. 13. Mapping from NICE-SLAM in the laboratory.

3.3.3. SA Optimization

While NICE-SLAM utilizing feature decoders from ConvONet [4], we optimized NICE-SLAM with sign-agnostic (SA) optimization. We compared the mapping results of the scene with one chair by these two approaches, as shown in Fig. 14 and Fig. 15.

Either the original NICE-SLAM or the modified form (N+SA) successfully reconstructed the scene close to the real world. Moreover, the wall mapped from the original NICE-SLAM is smoother. However, it cannot be concluded that the original NICE-SLAM is better because the above two mesh reconstruction is based on two different sets of hyperparameters. For example, the yaml file has a hyperparameter called "bound" to define the 3D space we have, and the suitable value for NICE-SLAM is smaller than the modified SLAM. That is to say, the hyperparameters enabling the 3D reconstruction from NICE-SLAM cannot be run by N+SA, and vice versa. This phenomenon indicates that it is hard to tune the hyperparameters based on the original NICE-SLAM. The mapping results shown in Fig. 14 and Fig. 15 is based on the best hyperparameters we had found, which may be further improved.

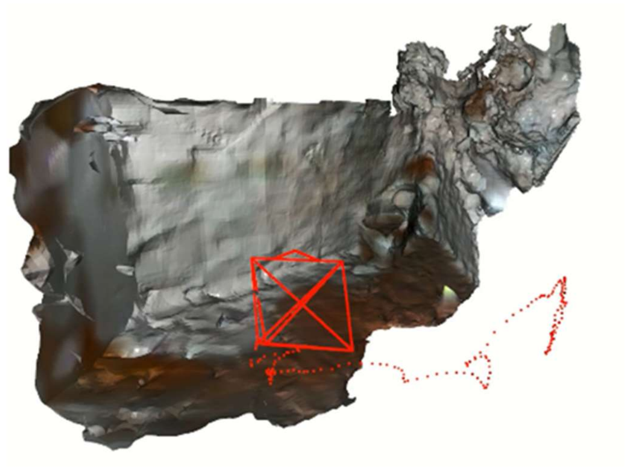


Fig. 14. Mapping from NICE-SLAM in an indoor scene.

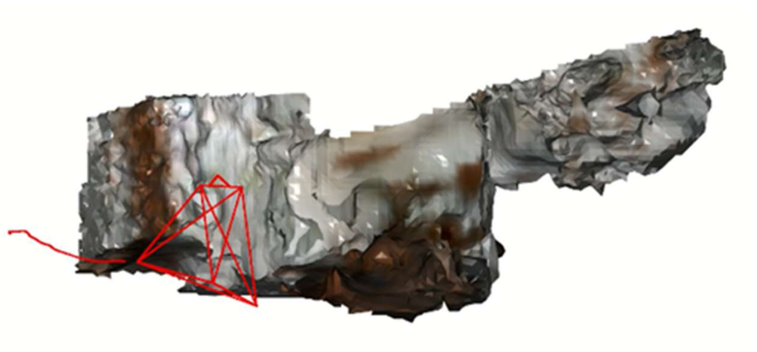


Fig. 15. Mapping with SA optimization in an indoor scene.

3.3.4. Dynamic Environments

In this experiment, a person held an RGB-D camera, and this person followed another person behind. Therefore, one person will always exist in every input image; and we implement Mask R-CNN [8] to remove the human and its corresponding depth information. Next, we perform the SLAM algorithm. The camera pose should be calculated, and also 3D reconstruction should be estimated. The ideal scenario is that no human exists in our 3D map.

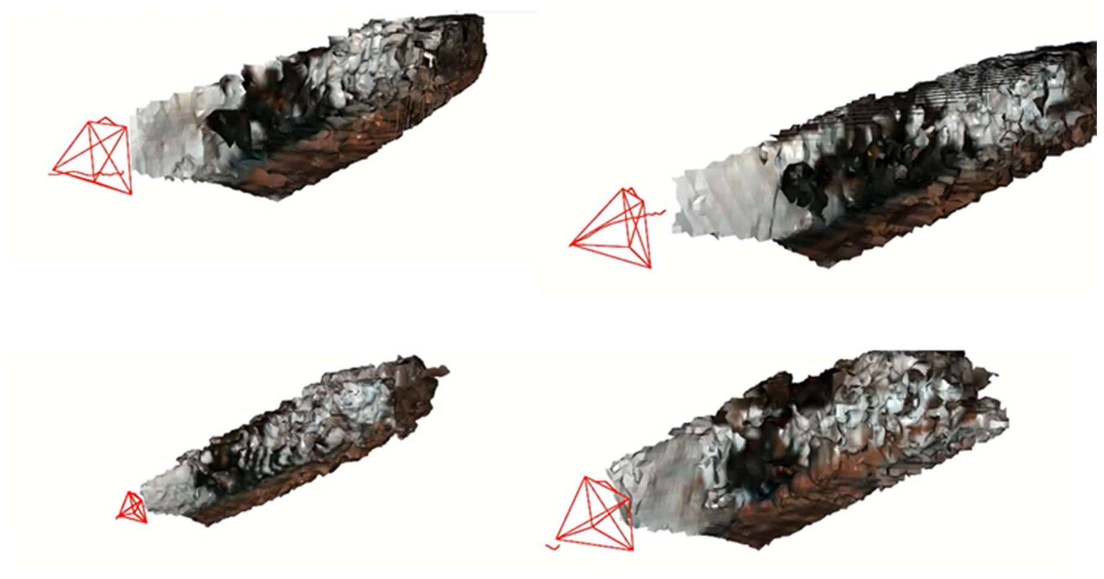


Fig. 16. Walking in the hallway:
(top left) N; (top right) N+SA; (bottom left) N+M; (bottom right) SAD-SLAM (N+SA+M)

The mapping results are compared in Fig. 16. If a dynamic object exists in the scene, and if this object is not removed, then camera pose estimation gives wrong results. However, if Mask R-CNN is implemented, then the human information is neglected, the camera poses are well-estimated, and the mapping is fine-tuned without human beings.

4. Discussion

4.1 Tuning Hyperparameters

There are many hyperparameters in NICE-SLAM, and as we mentioned in Section 3.3.3, NICE-SLAM becomes unstable when adjusting some particular hyperparameters. For example, in the scene of a hallway without humans, the successive frames will not be well-registered in NICE-SLAM if the hyperparameters are not well-tuned. As shown in Fig. 17, NICE-SLAM gives wrong camera poses during the last few frames, and three hallways are shown in our final map while in fact only one hallway exists. Besides, the hyperparameter ‘bound’ in the yaml file should be tuned; otherwise, the wrong message will be displayed repeatedly, as shown in Fig. 18.

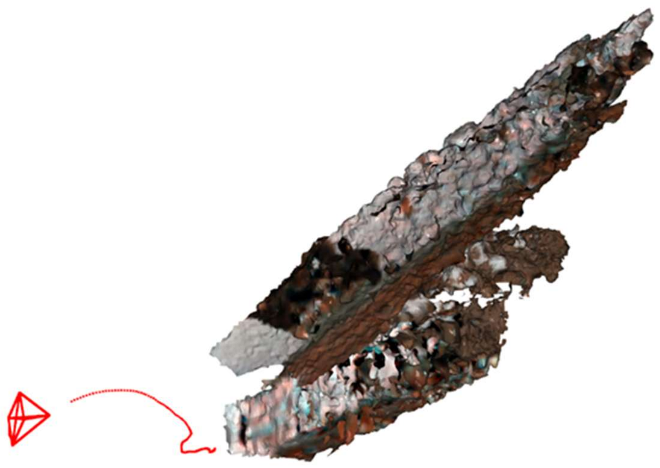
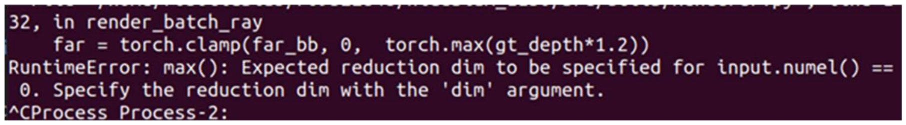


Fig. 17. Incomplete map constructed from NICE-SLAM with non-applicable hyperparameters.



```
32, in render_batch_ray
    far = torch.clamp(far_bb, 0,  torch.max(gt_depth*1.2))
RuntimeError: max(): Expected reduction dim to be specified for input.numel() ==
 0. Specify the reduction dim with the 'dim' argument.
^CProcess Process-2:
```

Fig. 18. The error message from non-applicable hyperparameters.

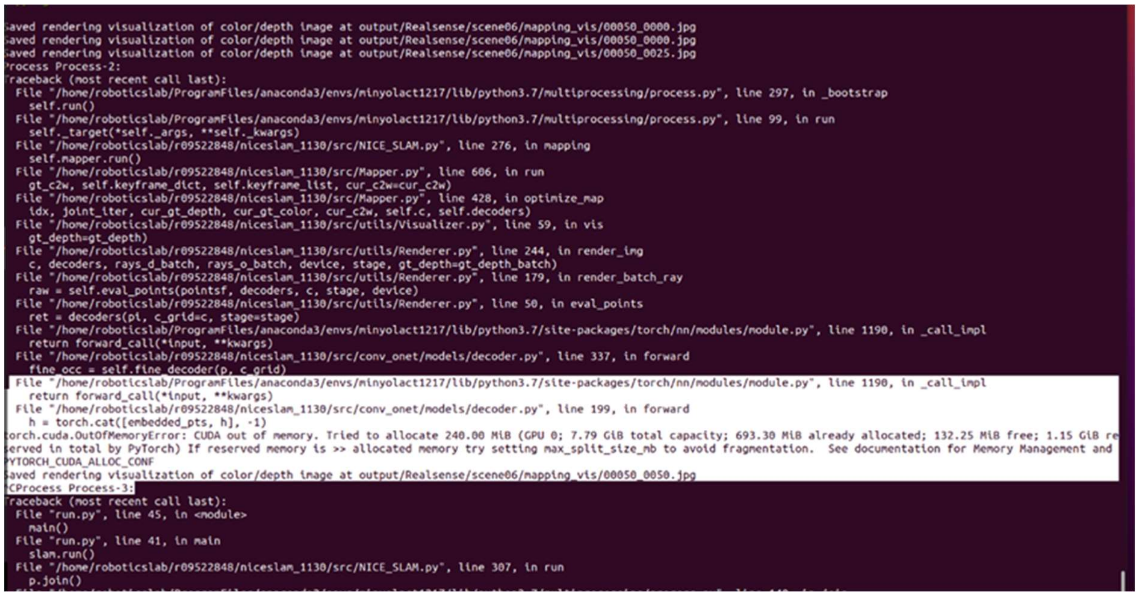
Besides, sign-agnostic (SA) is our optimization approach, and it is expected to have a better performance. However, as an additional optimization steps to NICE-SLAM, the running time depends on the number of the optimization iterations. This value also influenced our final map of SLAM. Therefore, finding a suitable number of iterations depends on the device we use or the dataset we compare, and it is practical to set ten or fewer iterations for most scenarios.

4.2 Scalability

The original NICE-SLAM claims its scalability in its abbreviated name with an alphabet ‘C’. For example, our first experiment shows this property if local information is comprised mainly of simple geometry, and obviously the local desks and desktops are shown in the resultant map. However, if local information includes complicated features, then NICE-SLAM might become unstable. Take our first experiment for example. If one frame captured by the RGB-D camera included a large-scale environment, the estimated trajectory gradually deviated from the ground truth and the environment was constructed without good registration. Therefore, we validated that local information with simple features is scalable to a large-scale map, but NICE-SLAM might have infeasible mapping results if each frame includes complicated information.

4.3 GPU Usage and Real-time Performance

Since most AI-approach algorithms speed up the process with GPUs, NICE-SLAM also runs faster if GPUs are utilized. Most neural network architectures in NICE-SLAM run on GPUs, and GPU usage depends on some hyperparameters and versions of Python modules. While it is supposed that PyTorch, a Python package, should support all kinds of data running on GPUs, large amounts of data, however, causes the program to run out of memory. Our experience of tuning hyperparameters in the above-mentioned experiments tells us that larger bounds generate better results, while GeForce RTX 30 Series Graphics Card could maintain relatively small bounds in NICE-SLAM. Therefore, our GPU might be utilized to the fullest when running NICE-SLAM, as shown in Fig. 19. In addition, NICE-SLAM is written in Python with large neural network models, so the tasks of localization and mapping cannot be achieved in real-time performance. RTX 30 Series processes only one or two frames per second (FPS), while most visual SLAM algorithms support at least 30 FPS. Therefore, if supercomputers are not considered to perform the SLAM algorithm, the current NICE-SLAM cannot be implemented with real-time performance.



```
aved rendering visualization of color/depth image at output/Realsense/scene06/mapping_vis/00050_0000.jpg
aved rendering visualization of color/depth image at output/Realsense/scene06/mapping_vis/00050_0000.jpg
aved rendering visualization of color/depth image at output/Realsense/scene06/mapping_vis/00050_0025.jpg
rocess Process-2:
raceback (most recent call last):
  File "/home/roboticslab/ProgramFiles/anaconda3/envs/minyolact1217/lib/python3.7/multiprocessing/process.py", line 297, in _bootstrap
    self.run()
  File "/home/roboticslab/ProgramFiles/anaconda3/envs/minyolact1217/lib/python3.7/multiprocessing/process.py", line 99, in run
    self._target(*self._args, **self._kwargs)
  File "/home/roboticslab/r09522848/niceslam_1130/src/NICE_SLAM.py", line 276, in mapping
    self.mapper.run()
  File "/home/roboticslab/r09522848/niceslam_1130/src/Mapper.py", line 606, in run
    gt_c2w, self.keyframe_dict, self.keyframe_list, cur_c2w=cur_c2w)
  File "/home/roboticslab/r09522848/niceslam_1130/src/Mapper.py", line 428, in optimize_map
    idx, joint_iter, cur_gt_depth, cur_gt_color, cur_c2w, self.c, self.decoders)
  File "/home/roboticslab/r09522848/niceslam_1130/src/utils/Visualizer.py", line 59, in vis
    gt_depth=gt_depth)
  File "/home/roboticslab/r09522848/niceslam_1130/src/utils/Renderer.py", line 244, in render_img
    c, decoders, rays_d_batch, rays_o_batch, device, stage, gt_depth=gt_depth_batch)
  File "/home/roboticslab/r09522848/niceslam_1130/src/utils/Renderer.py", line 179, in render_batch_ray
    raw = self.eval_points(pointsf, decoders, c, stage, device)
  File "/home/roboticslab/r09522848/niceslam_1130/src/utils/Renderer.py", line 50, in eval_points
    ret = decoders(pi, c_grid=c, stage=stage)
  File "/home/roboticslab/ProgramFiles/anaconda3/envs/minyolact1217/lib/python3.7/site-packages/torch/nn/modules/module.py", line 1190, in _call_impl
    return forward_call(*input, **kwargs)
  File "/home/roboticslab/r09522848/niceslam_1130/src/conv_onet/models/decoder.py", line 337, in forward
    fine_occ = self.fine_decoder(p, c_grid)
  File "/home/roboticslab/ProgramFiles/anaconda3/envs/minyolact1217/lib/python3.7/site-packages/torch/nn/modules/module.py", line 1190, in _call_impl
    return forward_call(*input, **kwargs)
  File "/home/roboticslab/r09522848/niceslam_1130/src/conv_onet/models/decoder.py", line 199, in forward
    h = torch.cat([embedded_pts, h], -1)
orch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 240.00 MiB (GPU 0; 7.79 GiB total capacity; 693.30 MiB already allocated; 132.25 MiB free; 1.15 GiB re
erved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.  See documentation for Memory Management and
YTORCH_CUDA_ALLOC_CONF
aved rendering visualization of color/depth image at output/Realsense/scene06/mapping_vis/00050_0050.jpg
CProcess Process-3:
raceback (most recent call last):
  File "run.py", line 45, in <module>
    main()
  File "run.py", line 41, in main
    slam.run()
  File "/home/roboticslab/r09522848/niceslam_1130/src/NICE_SLAM.py", line 307, in run
    p.join()
```

Fig. 19. Running NICE-SLAM out of memory.

4.4 Inpainting

To remove people from the scene, the inpainting ability is implemented in this project. However, as the previous section has suggested, there are few incorrect fillings. And this issue is partly improved in our project. For example, the strategy of selecting a keyframe as the source for background inpainting should be determined, and to get a better result, it is found that the earliest keyframe should be selected. Although our approach mentioned in the previous section has improved some of the inpainting issues, some incorrect results still exist. Therefore, both instance segmentation and inpainting are challenging tasks, and the SLAM performance will be significantly influenced by how each frame is segmented and inpainted.

4.5 Dynamic Scenes

Moving object removal is one of the objectives that our project has, and we achieve this by removing people with instance segmentation. In the architecture of Mask R-CNN, every instance in a frame can be masked. Technically, we implement instance segmentation to remove some masks with specific labels, such as person. However, there are many objects that are not static, so it becomes trivial to define the targeted labels before we implement Mask R-CNN. Besides, we could have prior knowledge that humans can walk, but an instance is not equivalent to a moving object. In other words, if a SLAM algorithm aims to map the dynamic scene, a direct approach to moving object removal could reduce the computation time of the pipeline, while instance segmentation is an alternative but possibly not an efficient solution. Therefore, it is expected that the proposed SLAM algorithm is robust to dynamic scenes, and it is suggested that a direct approach to moving object removal should be considered first when real-time performance is required.

Reference

[1] Zhu, Z., Peng, S., Larsson, V., Xu, W., Bao, H., Cui, Z., ... & Pollefeys, M. (2022). Nice-slam: Neural implicit scalable encoding for slam. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 12786-12796).

[2] Dai, A., Chang, A. X., Savva, M., Halber, M., Funkhouser, T., & Nießner, M. (2017). Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 5828-5839).

[3] Straub, J., Whelan, T., Ma, L., Chen, Y., Wijmans, E., Green, S., ... & Newcombe, R. (2019). The Replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv*:1906.05797.

[4] Peng, S., Niemeyer, M., Mescheder, L., Pollefeys, M., & Geiger, A. (2020, August). Convolutional occupancy networks. In *European Conference on Computer Vision* (pp. 523-540). Springer, Cham.

[5] Tang, J., Lei, J., Xu, D., Ma, F., Jia, K., & Zhang, L. (2021). Sa-convonet: Sign-agnostic optimization of convolutional occupancy networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 6504-6513).

[6] Palazzolo, E., Behley, J., Lottes, P., Giguere, P., & Stachniss, C. (2019, November). Refusion: 3d reconstruction in dynamic environments for rgb-d cameras exploiting

residuals. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 7855-7862)*. IEEE.

[7] Bescos, B., Fácil, J. M., Civera, J., & Neira, J. (2018). DynaSLAM: Tracking, mapping, and inpainting in dynamic scenes. *IEEE Robotics and Automation Letters*, 3(4), 4076-4083.

[8] He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision* (pp. 2961-2969).