CS588 Final: PointPillars for 3D Object Detections

Hongqing Liu[1], Weichen Liu[2], Jingwei Bao[3]

[1]) University of Illinois Urbana Champaign
hl85@illinois.edu
NetID: hl85
[2]) University of Illinois Urbana Champaign
@illinois.edu
NetID: wl45
[3]) University of Illinois Urbana Champaign
jb52@illinois.edu
NetID: jb52

Introduction

Visual Simultaneous Localization and Mapping (SLAM) is a critical technology in various applications, such as robotics, navigation, and augmented reality. It enables devices to understand their surroundings, build accurate maps, and navigate through complex environments. Over the past few years, RGB-D based SLAM systems have gained popularity due to their ability to provide detailed 3D reconstruction models and precise camera poses for tracking and mapping. This technology is particularly beneficial for indoor environments, where depth information is critical for creating accurate maps [5]. However, most existing methods are tailored for indoor scenarios and struggle to cope with the challenges posed by outdoor settings [2]. In this work, we aim to address this limitation and develop an effective real-time visual SLAM approach capable of generating detailed 3D maps for outdoor scenes.

Problem Definition

The primary goal of our research is to develop an RGB-based visual SLAM system for outdoor environments. The existing state-of-the-art work, Nice-SLAM [6], is limited to indoor settings due to its reliance on RGB-D image streams, which provide accurate depth information only within a small range. Outdoor environments pose significant challenges for RGB-D based methods, as sensors struggle to obtain precise depth information for distant objects, ultimately affecting the reconstruction and mapping capabilities of previous methods [1].

Outdoor environments present a different set of challenges compared to indoor settings. They typically have a larger scale and dynamic elements such as moving objects [1] and changing weather conditions [3]. Additionally, outdoor

scenes often contain repeating textures and patterns, which can cause ambiguity in feature matching and pose estimation [4]. Moreover, the availability of accurate depth information is essential for creating precise maps, but it is challenging to obtain in outdoor settings due to limitations in sensor range and accuracy.

Method

Our objective is to develop a visual SLAM for outdoor environments. In this section, we first present the model structure, which includes the depth prediction architecture and the SLAM architecture. Next, we describe the fusion of estimated depth and captured depth. We then introduce the weighted sample to enhance tracking. Finally, we explain how the tracker model filter improves adaptability to outdoor scenes and maintains robust performance.

Model Structure

We propose a novel visual SLAM model for outdoor environments that consists of the depth prediction architecture and the SLAM architecture, as shown in Fig 1. The depth prediction architecture is based on DeepPruner, which takes two RGB images as input and predicts depth information. We then fuse the estimated depth with the captured depth from the depth camera. The SLAM architecture builds upon the foundation of Nice-SLAM, which takes the generated RGBD information as input to track the camera pose and create a map.

The depth prediction architecture in our model employs the DeepPruner algorithm to estimate stereo depth from two RGB images. Utilizing differentiable PatchMatch for depth prediction, the algorithm is composed of three layers. In the particle sampling layer, each pixel is randomly assigned K values. The propagation layer spreads disparity information to adjacent pixels, while the evaluation layer determines the most accurate depth information for each pixel. The model then aggregates the cost and refines the final depth output.

When a camera can only capture stereo RGB information, the depth prediction architecture generates reliable depth information for the SLAM system. If the camera captures depth information, the estimated depth serves as supplementary information to enhance overall depth accuracy. For instance, the ZED2 camera has a depth capture range of up to 20 meters, which is insufficient for RGBD SLAM. By predicting depth beyond this range, our depth prediction architecture effectively bridges this gap. We fine-tune DeepPruner using our dataset of 1000 data points over 500 epochs, enabling the model to learn the camera's intrinsic and extrinsic parameters, thereby optimizing its performance for the ZED 2 camera. The fusion of estimated and captured depth significantly improves the model's performance compared to relying on either predicted or captured depth alone.

The SLAM architecture adopts the NICE-SLAM algorithm's architecture. This architecture simultaneously tracks the camera pose and constructs a map

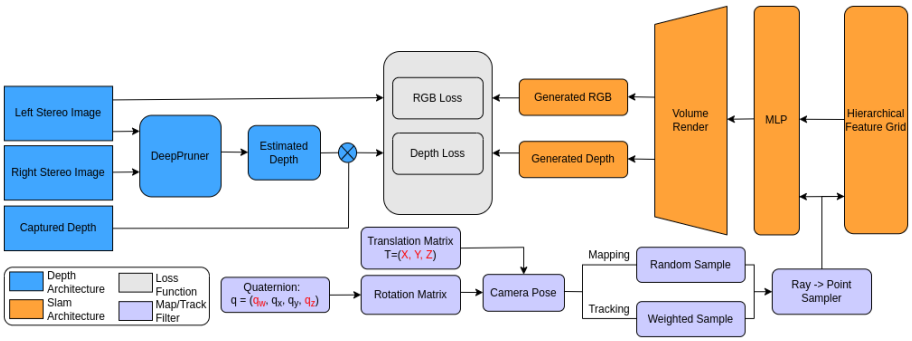


Figure 1: **Overview:** The model takes stereo images and the captured depth as input to track and create the map for outdoor scenes. Firstly, the Depth Architecture predicts depth based on stereo RGB images. Secondly, the depth fusion module fuses the captured and estimated depth with weights. Then, the fused depth and left color image are sent to Slam Architecture to track and generate the map. The Slam Architecture predicts depth, color and density for sampled points in neural implicit space and renders back to RGB and depth images. In Map/Track filter, we implement weighted sample and tracker model filter to enhance the tracking. **Tracking:** The model updates the q_m, q_z in Quaternion and X, Y, Z in translation matrix through back propogation of losses. Besides, weighted sample module filters useless information to improve the tracking. **Mapping:** The model only updates feature grids with backpropagation of losses.

using color and depth inputs. The map is stored in a hierarchical Feature Grid, consisting of coarse, middle, and fine levels. Each level encodes geometric information, while the fine layer also incorporates color information.

To render images from the implicit space, the architecture proceeds through several steps. First, it samples pixels from the image under an assumed camera pose and casts rays for each pixel. Second, it samples 3D points along the ray and calculates trilinear interpolation features for each point based on neighboring grid point features. Third, these features are input into a pre-trained decoder network to predict color, depth, and density for each sampled point. Finally, the architecture employs a differentiable color rendering process to render the RGB image and depth image for each pixel.

The architecture optimizes grid features and camera pose by minimizing the reconstruction loss between rendered and ground truth key frames. The system operates on three separate threads. The tracking thread focuses on fixing the features of the hierarchical feature grid while training the camera pose. The remaining two threads are assigned to mapping, where the system holds the camera pose constant and trains the hierarchical feature grids at the coarse, middle, and fine levels.

By training grid features and camera pose simultaneously on different threads with varying frequencies, the system can effectively execute the V-SLAM algorithm in real-time, while maintaining a more formal and readable presentation of the process.

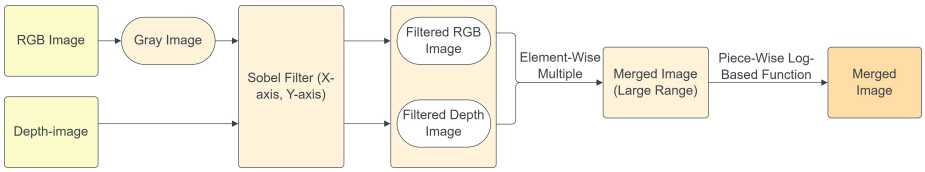


Figure 2: Pipeline of the weighted sample method

Depth fusion module

We introduce a depth fusion module to fuse the estimated depth images and the captured depth images.

The primary limitation of the captured depth image is its restricted detecting range from 0.4 to 20 meters, which is constrained by the depth sensor's technical specifications. However, the ground truth depth image offers highly precise distance measurements for objects detected within the 20-meter range. Additionally, some areas within 20 meters may display an infinite distance due to reflection.

The advantages of the estimated depth image include providing reasonably accurate depth information for uniform areas and objects beyond 20 meters, leading to more accurate geometric relationships and object distances. However,

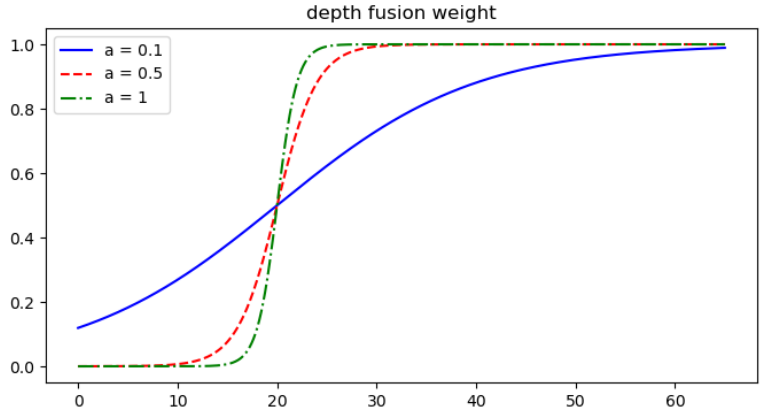


Figure 3: Depth fusion weight. This image shows curves that depict three different weights used for depth fusion.

its precision for objects within 20 meters is not as high as the captured depth images.

To obtain a more accurate depth image while maintaining geometric continuity, we propose integrating the estimated depth image and captured depth image using a weighted approach. We employ the captured depth to calculate weights for each pixel.

The equation 1 and visualization of the fusion weight Fig 3 are presented below.

$$\text{weight}(x, a) = \frac{1}{1 + e^{-a(x-20)}} \tag{1}$$

The formulation of the merged depth is shown as equation 2.

$$\text{M} = \text{weight} * \text{P} + (1 - \text{weight}) * \text{C} \tag{2}$$

where the M means merged depth fusion, P means predicted depth image, C means captured depth image. We assign a higher weight to the ground truth depth image when the distance value is less than 20 meters, as it is known to be more accurate in this range. On the other hand, we assign a higher weight to the estimated depth image for distances beyond 20 meters, where it is expected to provide a more reasonable representation of the scene geometry.

Weighted sample for tracking

We introduce the weighted sample method to prioritize sampling pixels on objects with more distinctive shapes and features. The pipeline of the weighted sample method is presented in Figure 2, consisting of two parts: generating the weighted map and sampling based on the weighted map.

In the first part, the RGB image is converted to grayscale, where both the RGB grayscale image and depth image are filtered with a Gaussian blur filter to reduce noise. Subsequently, a Sobel filter is applied to filter the RGB grayscale and depth images along the x-axis and y-axis, and then weighted combined on each image. The RGB grayscale image effectively filters out most of the lane, while the depth image efficiently filters out most sky pixels. Next, the two filtered images are multiplied to obtain the merged weighted map. The underlying rationale is the "or" operation, where sharper parts have larger values and smoother parts have lower values. During testing, it was discovered that the value for the larger parts was too high, which leads to poor weighted results in the following step. In order to tackle this problem, a piecewise function , as shown in Equation 3, was employed to attenuate the discrepancies present in the weighted sample image.

$$f(x) = \begin{cases} x & \text{if } x < 1 \\ 1 + \ln(x) & \text{otherwise} \end{cases} \tag{3}$$

In the second part, a list is created to accumulate weighted values in weighted map, it is then normalized to a range between 0 and 1. Next, n random values between 0 and 1 are generated. For each weighted sample, the index of the first accumulated value that larger than the corresponding random value is selected. The principle for this method is that pixels with larger weighted values are more likely to be chosen, allowing for the sampling of more points in areas with sharper features than in smoother regions.

The majority of the pixels in a camera scene are captured by a vehicle corresponding to the lane. When the car is moving forward, the depth and RGB images of the lane exhibit relatively minor variations, thereby posing a challenge for the system to accurately track the camera pose. The tracking pipeline, as described in section 3.1, involves generating sample rays by sampling pixels in an image using an assumed camera pose, and creating sample points along those rays to render the image. By comparing and training based on the render rgb-d and ground truth rgb-d, the system can find the optimal camera pose. If the sampling of pixels is random, the majority of them will correspond to pixels of the lane. Since the ground truth depth value and RGB information of the lane are not changed significantly when the car is moving forward, the system cannot effectively track the camera pose.

Tracker model filter

We introduce the tracker model filter to decrease the training parameters by transforming the projection matrix from 2 rotation quaternion parameters and 3 translation parameters.

Due to the increased velocity of the camera mounted on a moving vehicle, tracking objects in an outdoor environment becomes significantly more challenging. Utilizing the original tracker model needs more iterations to attain optimal performance. This consequently impairs the system's real-time capabilities as

the tracker must be executed in each frame. Given that the motion models of both the camera and lidar sensors conform to the Dubin car motion model, it is possible to leverage certain a priori assumptions for the tracker.

There are two optimal strategies that can be employed to improve tracking performance in the Dubin car motion model. Firstly, due to the fact that translation typically changes faster than rotation, the system separates the two parameters and allocates a larger learning rate to translation. Secondly, the camera's rotation is limited to its z-axis (from down to up) according to the car model. To reduce the number of parameters during tracking, the rotation matrix can be converted to a quaternion and only the z and w values need to be trained.

Given a quaternion $q = (q_w, q_x, q_y, q_z)$, the corresponding rotation matrix R can be calculated as follows:

$$R = \begin{bmatrix} 1 - 2q_y^2 - 2q_z^2 & 2q_xq_y - 2q_wq_z & 2q_xq_z + 2q_wq_y \\ 2q_xq_y + 2q_wq_z & 1 - 2q_x^2 - 2q_z^2 & 2q_yq_z - 2q_wq_x \\ 2q_xq_z - 2q_wq_y & 2q_yq_z + 2q_wq_x & 1 - 2q_x^2 - 2q_y^2 \end{bmatrix}$$

In light of these considerations, a new step has been added to the tracker training pipeline. Firstly, the input variables are two rotation quaternion parameters (z and w) and three translation parameters. Subsequently, these variables are transformed into the projection matrix, which provides the camera pose required by the system to generate sample rays.

Loss

Inherited from the idea of Nice-SLAM, our model consists of three threads to speed up the optimization:

One thread for depth and color optimization, the color loss $\mathcal{L}_p$ is L_1 loss between the observation I and the generated color $\hat{I}$. $\mathcal{L}_d^c$ and $\mathcal{L}_d^f$ are L_1 loss between the observation D and the generated depth $\hat{D}$ in coarse and fine level.

One thread for mapping, the reconstruction render loss $\mathcal{L}_r$ in eq: reconstruction loss is the combination of geometric loss in coarse level $\mathcal{L}_d^c$, fine level $\mathcal{L}_d^f$ and color loss $\mathcal{L}_p$ with weight λ_m to M pixels.

$$\mathcal{L}_r = \mathcal{L}_d^c + \mathcal{L}_d^f + \lambda_m \mathcal{L}_p \tag{4}$$

One thread for tracking, the track loss $\mathcal{L}_t$ in eq: track loss is the combination of modification of depth loss and color loss $\mathcal{L}_p$ with weight λ_t to T pixels.

$$\mathcal{L}_t = \frac{1}{T} \sum_{t=1}^{T} \frac{\left| D_m - \hat{D}_m^c \right|}{\sqrt{\hat{D}_{\text{var}}^c}} + \frac{\left| D_m - \hat{D}_m^f \right|}{\sqrt{\hat{D}_{\text{var}}^f}} + \lambda_t \mathcal{L}_p \tag{5}$$

Experiments

In this section, we first describe the setup environment. Then we explain the collection and preprocess of our own dataset. Lastly, we visualize the result of our modules and rendered maps. Finally, we made a comprehensive ablation experiment and analysis.

Implement details

We run our system on a desktop PC with a 2100MHz Intel i7-12700F CPU and an NVIDIA RTX 3090 Ti GPU. In all our experiments, we use the number of sampling points on a ray N-sample = 64 and N-surface = 32. We select K = 5 keyframes and sample M = 2000 pixels in each image. As we sample more points on the image and render more sample points, the mapping and tracking performance increases. Those parameters are the trade-off result between mapping performance and real-time capabilities.

Dataset

Highbay dataset: We create our own dataset for the model. There are two reasons to use the own dataset: Firstly, the dataset could be used to train the model. Then we could implement our model in the autonomous vehicle to test the performance in the real scene. Secondly, to render the predicted image and depth in the SLAM architecture, objects have to be static. Once objects move, the map will contain residual shadow. Therefore, we need an outdoor dataset with RGB and depth images for static scenes.

The dataset contains several scenes as the car moves. The scene is located at the Research Park of UIUC. We sample 1000 frames to finetune the DeepPrunner and 500 frames to run RGB-SLAM. The RGB and depth information was collected by ZED 2 camera which is set on an autonomous vehicle with ROS system and two 3090 GPUs. The depth information ranges from 0.4 to 20 meters, containing "NAN" which is caused by the difference between two images, and "INFINITY" which means too far to calculate. Therefore, we transform "NAN" to 0 meters and "INFINITY" to 65 meters which is much larger than the maximum range of the ZED 2. To fit the scale of the SLAM architecture, we transform depth from meters to millimeters and save with uint16 type.

Result

Qualitative and Quantitative Analysis

To evaluate on High-bay parking lot, we use a depth fusion SLAM algorithm with a depth fusion module, weighted sample, and tracker model filter. Our method is able to reconstruct the geometry in the rendered cube precisely within limited iterations. As shown in Figure 4, there are two frames of results with depth and color information. From left to right, we visualize the captured images, generated images, and residual images for color and depth.

The generated results successfully map the outdoor scene including cars and trees and filter things outside of the cube, such as the sky which represents gray. Besides, the correct render demonstrates the success of tracking.

As shown Best Result in Figure 1, our method has a low reconstruction render loss and in Figure 2 out tracking loss is also very low compared with the ground truth pose.

iteration	Best Result	No Depth fusion
0	6.7866	6.1715
1	0.2422	0.3644
50	0.2823	0.6633
100	0.3643	0.7124
150	0.4498	1.0352
200	0.4604	1.3453
250	0.4763	1.4083

Table 1: Reconstruction loss on parking lot

iteration	Best Result	No Module 2	No Module 3
0	0.0145	0.0241	0.0179
1	0.0212	0.1026	0.1147
50	0.0134	0.1153	0.1324
100	0.0131	0.2435	0.2298
150	0.0351	0.4532	0.3907
200	0.0236	0.4732	0.3670
250	0.0148	0.8254	0.3961

Table 2: Tracking loss on parking lot; Module 2: Tracker model filter; Module 3: Weighted sample filter

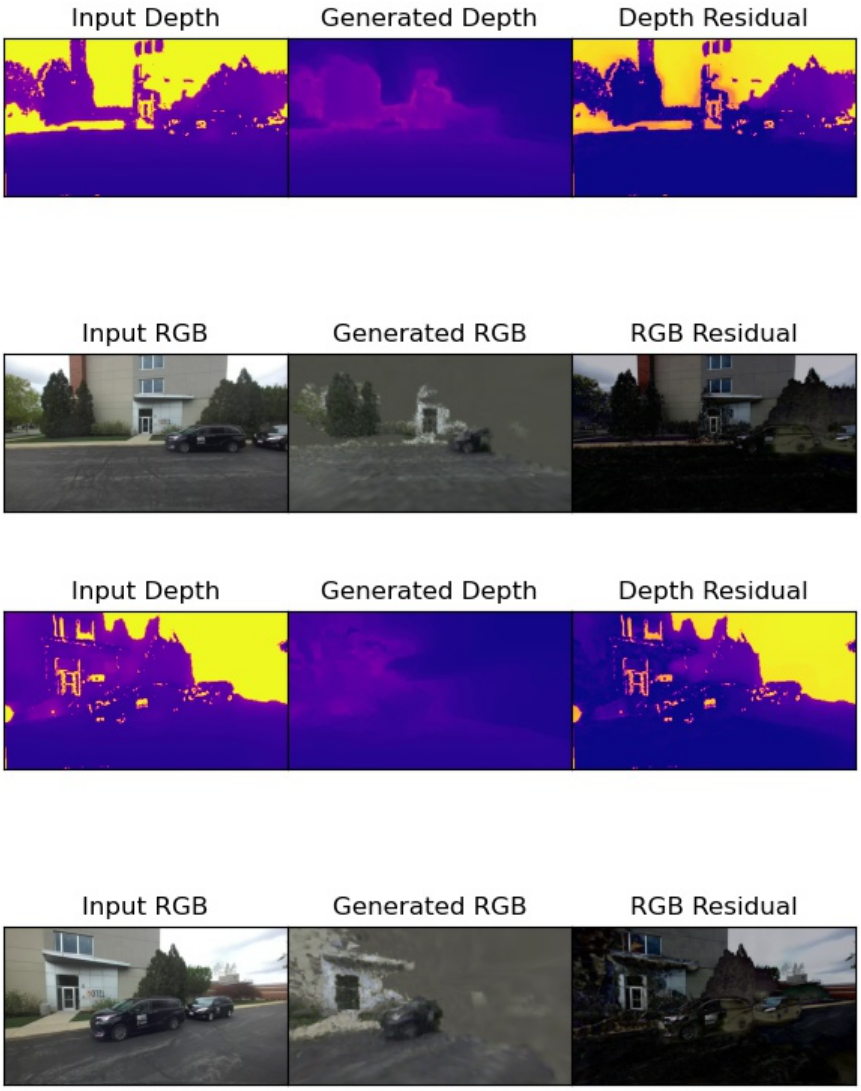


Figure 4: Reconstruction Results in frames 50 and 150 on the parking lot.

Ablation Experiments

Ablation on depth fusion

We implement the depth fusion module to enhance mapping. we present visualizations using only captured depth in Figure 5. Compared to the merged depth in Figure 4, the map is blurred when only using the captured depth, and objects

beyond the cube like sky aren't filtered as gray. The reconstruction render loss of "No Depth fusion" in Figure 1 is much higher than the Best result which uses the Depth fusion Module.

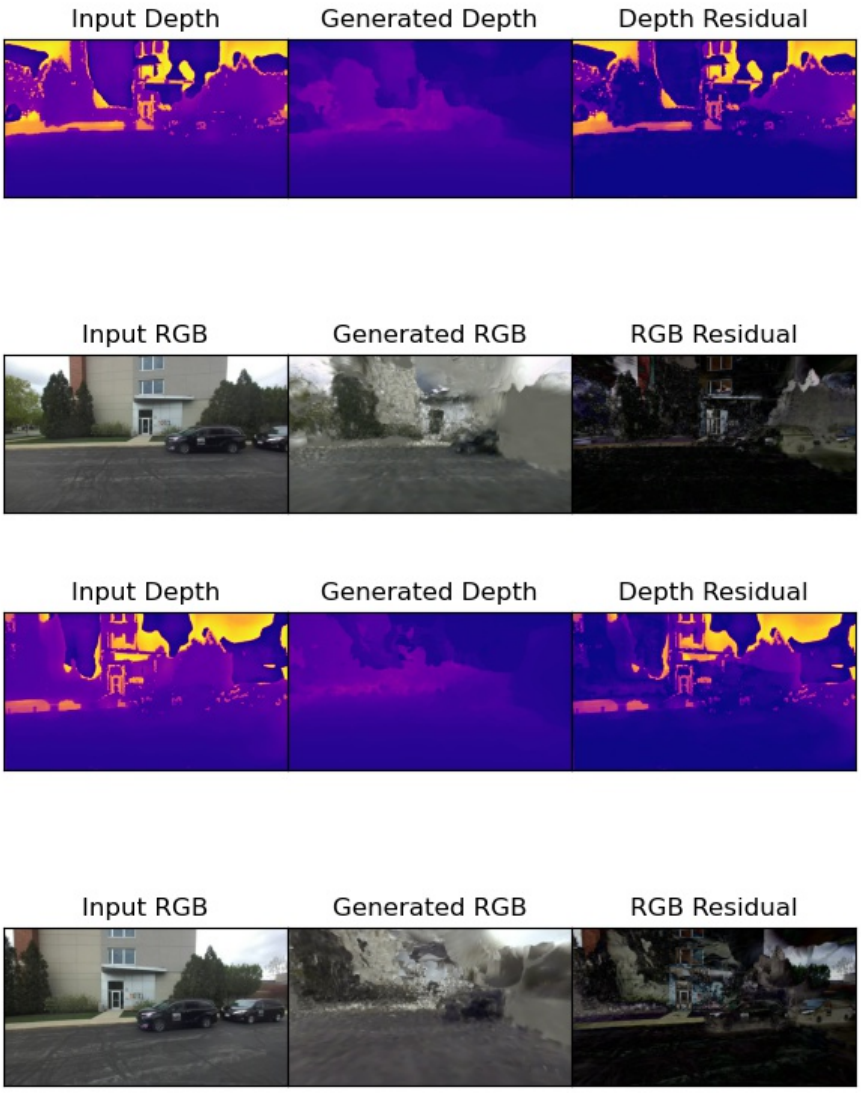


Figure 5: Mapping results of only using captured depth information for frames 50 and 150

In Figure 6, we visualize the captured, estimated and merged depth images: the left top is the captured depth image, the right top is the estimated depth

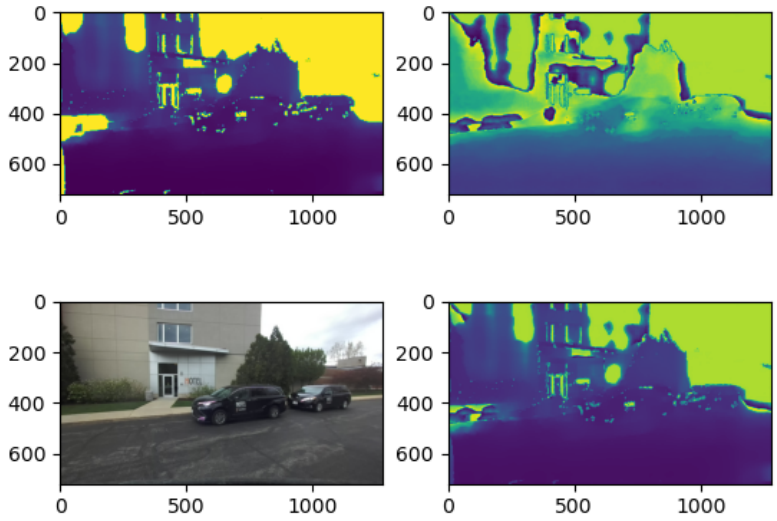


Figure 6: The Comparison of captured, estimated, and merged depth information

image, the left bottom is captured RGB image and the right bottom is merged depth image. The captured depth image has a bad geometry representation for objects over 20 meters and smooth objects in the range of 10 to 20, the merged depth image presents a more reasonable representation of the scene geometry.

Ablation on tracker model filter

The primary purpose of the tracker model filter method is to enhance tracking performance. As a result, we present visualizations of its mapping output in Figure 7 as a comparison with best result in Figure 4.

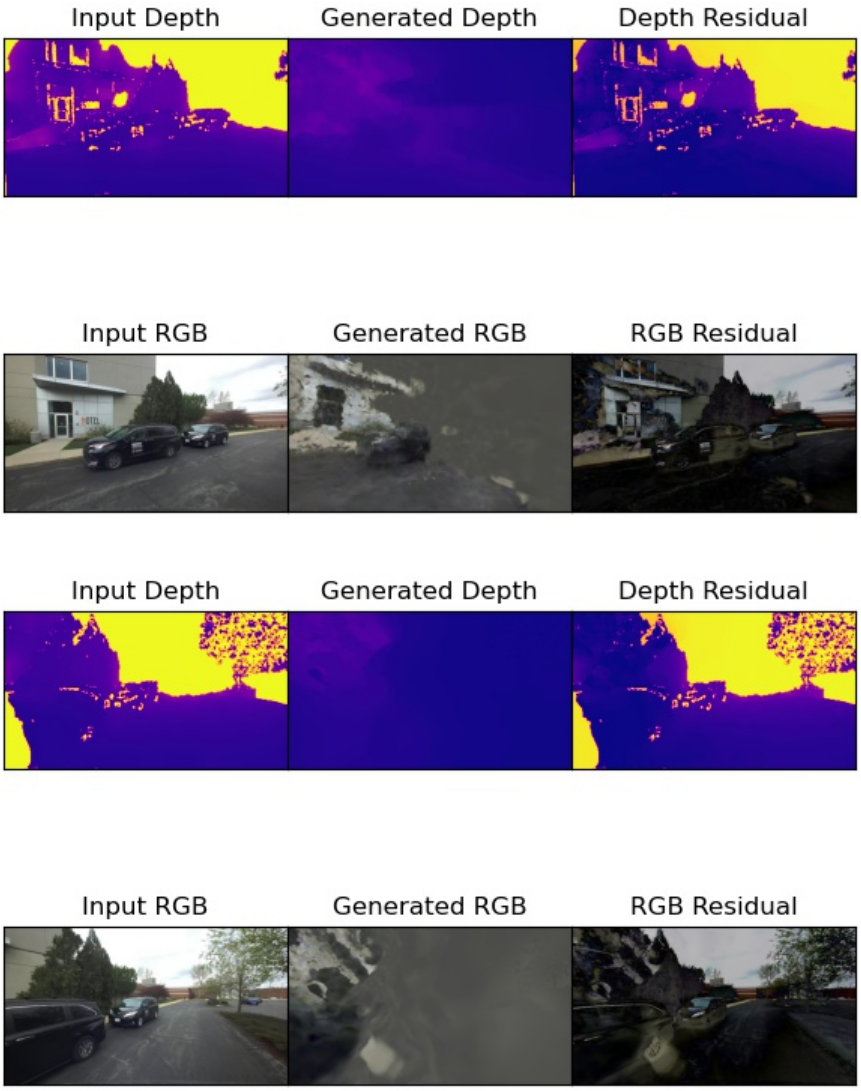


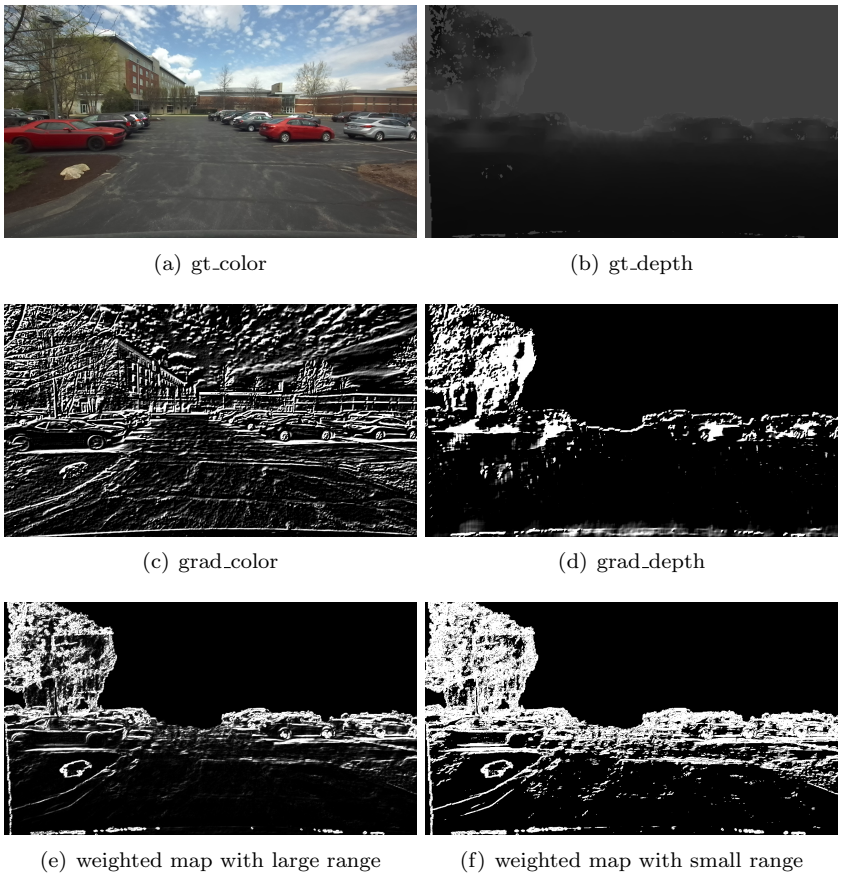
Figure 7: Mapping results of not using tracker model filter for frames 50 and 150

As shown in the result, without a tracker model, the system can not track the camera effectively and precisely in the outdoor environment, its estimated camera pose rotated along x-axis when we do not fix it to rotate around z-axis. Bad tracking result also leads to bad mapping result. It is important to note that the gray area is the part that is out of the bounding cube, so the system does not render that part. Besides, its corresponding tracking loss "No Module

2" in Figure 2 is much higher and unstable than Best Result which uses tracker model filter.

Ablation on weighted sample filter

The primary purpose of the weighted sample filter method is to enhance tracking performance. The corresponding tracking loss "No Module 3" is shown in Figure 2, which is increasing with the iteration. The sample weighted map of one frame is shown in Figure 8, the left top image is the ground truth color image, the right top is captured depth image, the left middle is the color image after sobel filter and the right middle is depth image after sobel filter, left bottom is weighted map with a large range and the right bottom is weighted map with small range. It is clear that after multiplying the gradient color and gradient depth images, the resulting weighted map can filter out the sky and lane. However, the edges have high values, which can cause the sample points to focus too heavily on those areas. After applying a piece-wise logarithmic operation, the map attenuates the discrepancies present in the weighted sample image. We utilize the result of the weighted sample points for tracking. Without this module, the tracking will collapse.



(a) gt_color (b) gt_depth

(c) grad_color (d) grad_depth

(e) weighted map with large range (f) weighted map with small range

Figure 8: weighted sample results

Conclusion

We present a novel method for outdoor scenes that incorporates a depth prediction architecture to refine the depth information and a Slam architecture to track and create the map. To overcome the challenge outdoor, we design a depth fusion module, weighted sample and tracker model filter, which dramatically enhance the model's performance. The main limitation of our method is the limited slam space due to the huge consumption of memory, which lead to grey space in generated results. Therefore, we plan to address this limitation in our future work by using a hash grid to replace the fixed feature grid with an adaptive grid.

Future Work

It appears that Nerf-based slams rely on direction-based slam techniques for tracking, which may result in significant computational demands on the tracking part. In that case, we are trying to add a point-based tracker as a extra parallel thread of the project. While this tracker is more real-time, it may possess lower accuracy than the direction-based method. In situations where the point-based tracker's tracking error is significant, the Nerf-based tracker will be utilized to correct the camera position. However, due to current hardware limitations, we are unable to add this extra parallel thread to achieve real-time performance. In the future, when hardware advances further, we will attempt this approach.

References

[1] César Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[2] Jakob Engel, Jürgen Sturm, and Daniel Cremers. Direct sparse odometry. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 40, pages 611–625. IEEE, 2018.

[3] Sourav Garg, Vijay Dhiman, Hanmei Wang, Michael Devy, and Aleš Leonardis. Adverse weather benchmark for slam. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10777–10784. IEEE, 2020.

[4] David G Lowe. Distinctive image features from scale-invariant keypoints. In *International journal of computer vision*, volume 60, pages 91–110, 2004.

[5] 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.

[6] 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*, pages 12786–12796, 2022.