ODGS-SLAM: Omnidirectional Gaussian Splatting SLAM

Supplementary Material

S1. Derivation Details

S1.1. 3D Gaussian Splatting

Following [3], our SLAM system employs 3DGS [1] to represent the scene as a set of N anisotropic 3D Gaussians $\mathcal{G}_i$, $i \in N$, defined by a covariance matrix $\mathbf{\Sigma}_i \in \mathbb{R}^{3\times3}$, its center $\boldsymbol{\mu}_i$, and opacity α_i. Instead of using spherical harmonics to represent view-dependent radiance, each Gaussian is assigned one single color $\mathbf{c}_i$, to prioritize speed over rendering quality. For rendering, the 3D Gaussians are projected onto the image plane, given the camera pose $\mathbf{T}_\text{c}$ in world coordinates and the projection π. Following Zwicker *et al*. [9], the projection of $\mathbf{\Sigma}_i$ is calculated as:

$$\mathbf{\Sigma}_i' = \mathbf{J}_i \mathbf{W} \mathbf{\Sigma}_i \mathbf{W}^T \mathbf{J}_i^T, \tag{S1}$$

with $\mathbf{W}$ the rotational components of $\mathbf{T}_\text{c}$, and $\mathbf{J}_i$ the Jacobian of the local affine approximation of the projective transformation. The 2D covariance matrix $\mathbf{\Sigma}_{\text{2D}i}'$ for the 2D Gaussian can then be obtained from $\mathbf{\Sigma}_i'$ by dropping the last row and column. The 2D projection of $\boldsymbol{\mu}_i$ is obtained by:

$$\boldsymbol{\mu}_i' = \pi\left(\mathbf{T}_\text{c} \boldsymbol{\mu}_i\right). \tag{S2}$$

All Gaussians are reprojected to 2D and ordered from front to back. The color of a pixel $\mathbf{p} = (u, v)$ is determined as:

$$\mathbf{c}_\text{p}(\mathbf{p}) = \sum_{i \in N} \mathbf{c}_i w_i(\mathbf{p}) \prod_{j=1}^{i-1} (1 - w_j(\mathbf{p})), \tag{S3}$$

with $w_i(\mathbf{p})$ calculated from opacity α_i as well as the contribution of the i-th Gaussian to the pixel position $\mathbf{p}$:

$$w_i(\mathbf{p}) = \alpha_i \exp\left(-\frac{1}{2} \left(\mathbf{p} - \boldsymbol{\mu}_i'\right)^T {\mathbf{\Sigma}_{\text{2D}i}'}^{-1} \left(\mathbf{p} - \boldsymbol{\mu}_i'\right)\right). \tag{S4}$$

Similarly, the depth $d_\text{p}(\mathbf{p})$ can be synthesized, by replacing $\mathbf{c}_i$ in Eq. (S3) with the distance d_i from the camera center to $\boldsymbol{\mu}_i$. Analogously, the silhouette $s_\text{p}(\mathbf{p})$ can also be computed, by replacing $\mathbf{c}_i$ with 1. The latter denotes the coverage of each pixel with Gaussians $\mathcal{G}_i$. Now, a color image $I(\mathcal{G}, \mathbf{T}_\text{c})$, a depth image $D(\mathcal{G}, \mathbf{T}_\text{c})$, and a silhouette image $S(\mathcal{G}, \mathbf{T}_\text{c})$ can be rendered for any camera position $\mathbf{T}_\text{c}$ from the Gaussians $\mathcal{G}_i$. The rendering process is fully differentiable. The Gaussian parameters can be optimized to minimize the error between the rendered images and provided RGB/RGBD input frames via their derived gradients. Also, matrices $\mathbf{\Sigma}_i$ remain positive definite by splitting into rotation $\mathbf{R}_i$ and scaling $\mathbf{S}_i$ [1]:

$$\mathbf{\Sigma}_i = \mathbf{R}_i \mathbf{S}_i \mathbf{S}_i^T \mathbf{R}_i^T. \tag{S5}$$

In order to support independent optimization and to save memory, rotations $\mathbf{R}_i$ are stored as quaternions $\mathbf{q}_i$ and $\mathbf{S}_i$ as 3D vectors $\mathbf{s}_i$. As in [3], the extrinsic camera parameters for a respective frame are included. Now, the camera position of the input image can also be estimated. This enables the usage of the same Gaussian map representation for mapping as well as for camera tracking (see paper; Sec. 3.2).

An adaptive density control is run every $\text{it}_\text{adc} = 150$ iterations. New Gaussians are generated by cloning or splitting in under- and over-populated volumes, respectively, indicated by large positional gradients [1]. Further, Gaussians with α_i below threshold $\epsilon_\alpha = 0.7$ are deleted. Note that in the further explanations below, index i of all Gaussian parameters is omitted for readability.

S1.2. Gradient calculation for pose estimation

As discussed in the paper, the camera position $\mathbf{T}_\text{c}^k$ for a new frame $\mathcal{F}^k$ is estimated by minimizing the tracking loss $\mathcal{L}_\text{track}$, see Eqs. (17) and (18), with respect to $\mathbf{T}_\text{c}^k$ utilizing backpropagation through the Gaussian rasterizer. In the paper, the first parts of the derivation of the analytical gradients of the photometric E_pho and geometric error E_geo is shown. Starting from that, the full analytical derivation is provided here.

Following [7] it is assumed that Gaussian colors are independent of the camera position, and the gradient with respect to $\mathbf{\Sigma}_{\text{o,2D}i}'$ is dropped for efficiency, we get the following:

$$\begin{aligned}\frac{\partial E_\text{pho}}{\partial \mathbf{T}_\text{c}} &= \frac{\partial E_\text{pho}}{\partial \mathbf{c}_\text{p}} \sum_{i \in N} \left(\mathbf{c}_i \frac{\partial \mathbf{c}_\text{p}}{\partial w_i} \frac{\partial w_i}{\partial \mathbf{T}_\text{c}}\right) \\ &= \frac{\partial E_\text{pho}}{\partial \mathbf{c}_\text{p}} \sum_{i \in N} \mathbf{c}_i \frac{\partial \mathbf{c}_\text{p}}{\partial w_i} \left(\frac{\partial w_i}{\partial \boldsymbol{\mu}_{\text{o}i}'} \frac{\partial \boldsymbol{\mu}_{\text{o}i}'}{\partial \mathbf{T}_\text{c}}\right).\end{aligned} \tag{S6}$$

Here, the interesting term with respect to the pose gradient is $\frac{\partial \boldsymbol{\mu}_{\text{o}i}'}{\partial \mathbf{T}_\text{c}}$. All other terms are commonly used in 3DGS and are calculated recursively in the backward render step. Therefore, only the derivation of $\frac{\partial \boldsymbol{\mu}_{\text{o}i}'}{\partial \mathbf{T}_\text{c}}$ is shown in the following.

The camera pose $\mathbf{T}_\text{c}$ can be split up into the rotational part $\mathbf{R}_\text{c}$ and the translation $\mathbf{t}_\text{c}$. Using the camera pose, a Gaussian mean is given in world coordinates as $\boldsymbol{\mu}_i = (x_i, y_i, z_i)^T$ and can be transformed into the camera coordinate system with

$$\boldsymbol{\mu}_i^\text{c} = \mathbf{T}_\text{c} \boldsymbol{\mu}_i = \mathbf{R}_\text{c} \boldsymbol{\mu}_i + \mathbf{t}_\text{c} = (x_i^\text{c}, y_i^\text{c}, z_i^\text{c})^T. \tag{S7}$$

This is used to calculate the gradients of the Gaussian mean with respect to the pose $\frac{\partial \boldsymbol{\mu}_{\text{o}i}'}{\partial \mathbf{T}_\text{c}}$, which can be decomposed

into: (a) $\frac{\partial \boldsymbol{\mu}'_{oi}}{\partial \mathbf{t}_c}$ and (b) $\frac{\partial \boldsymbol{\mu}'_{oi}}{\partial \mathbf{R}_c}$. Both of these gradients are derived in the following.

Ad (a): Since translation $\mathbf{t}_c$ is related to $\boldsymbol{\mu}_i^c$, the chain rule of derivation can be applied to get:

$$\frac{\partial \boldsymbol{\mu}'_{oi}}{\partial \mathbf{t}_c} = \frac{\partial \boldsymbol{\mu}'_{oi}}{\partial \boldsymbol{\mu}_i^c} \frac{\partial \boldsymbol{\mu}_i^c}{\partial \mathbf{t}_c}. \tag{S8}$$

The first term on the left side, $\frac{\partial \boldsymbol{\mu}'_{oi}}{\partial \boldsymbol{\mu}_i^c}$, is the derivative of the projected point with respect to the 3D position and matches the Jacobian $\mathbf{J}_o$ calculated earlier in Eq. (8). The second term $\frac{\partial \boldsymbol{\mu}_i^c}{\partial \mathbf{t}_c}$ can be calculated from the transformation to camera coordinates in Eq. (S7) and results in:

$$\frac{\partial \boldsymbol{\mu}_i^c}{\partial \mathbf{t}_c} = \mathbf{I}_3, \tag{S9}$$

where $\mathbf{I}_3$ is the 3×3 identity matrix. Combining these two parts results in the final gradient with respect to the translation:

$$\frac{\partial \boldsymbol{\mu}'_{oi}}{\partial \mathbf{t}_c} = \frac{\partial \boldsymbol{\mu}'_{oi}}{\partial \boldsymbol{\mu}_i^c} \cdot \mathbf{I}_3 = \mathbf{J}_o. \tag{S10}$$

Ad (b): For memory efficiency, $\mathbf{R}_c$ is stored as a unit quaternion $\mathbf{q}_c = \left(q_r^c, q_i^c, q_j^c, q_k^c\right)^T$ with the following quaternion-to-matrix conversion:

$$\mathbf{R}(q) = 2 \begin{bmatrix} \frac{1}{2} - (q_j^2 + q_k^2) & (q_i q_j - q_r q_k) & (q_i q_k + q_r q_j) \\ (q_i q_j + q_r q_k) & \frac{1}{2} - (q_i^2 + q_k^2) & (q_j q_k - q_r q_i) \\ (q_i q_k - q_r q_j) & (q_j q_k + q_r q_i) & \frac{1}{2} - (q_i^2 + q_j^2) \end{bmatrix}. \tag{S11}$$

Starting from that, similar as for the translation, the chain rule is applied to split the term:

$$\frac{\partial \boldsymbol{\mu}'_{oi}}{\partial \mathbf{q}_c} = \frac{\partial \boldsymbol{\mu}'_{oi}}{\partial \boldsymbol{\mu}_i^c} \frac{\partial \boldsymbol{\mu}_i^c}{\partial \mathbf{q}_c}. \tag{S12}$$

The first part $\frac{\partial \boldsymbol{\mu}'_{oi}}{\partial \boldsymbol{\mu}_i^c}$ is exactly the same as in the translation calculation before. The second part $\frac{\partial \boldsymbol{\mu}_i^c}{\partial \mathbf{q}_c}$ is calculated using Eq. (S11) to substitute $\mathbf{R}$ in Eq. (S7).

Each of the components of the quaternion $\left(q_r^c, q_i^c, q_j^c, q_k^c\right)^T$ contributes to the rotation. So, the derivation for each component is calculated separately:

$$\frac{\partial \boldsymbol{\mu}_i^c}{\partial q_r^c} = 2 \begin{bmatrix} 0 & -q_k^c y_i & q_j^c z_i \\ q_k^c x_i & 0 & -q_i^c z_i \\ -q_j^c x_i & q_i^c y_i & 0 \end{bmatrix}, \tag{S13}$$

$$\frac{\partial \boldsymbol{\mu}_i^c}{\partial q_i^c} = 2 \begin{bmatrix} 0 & q_j^c y_i & q_k^c z_i \\ q_j^c x_i & -2q_i^c y_i & -q_r^c z_i \\ q_k^c x_i & q_r^c y_i & -2q_i^c z_i \end{bmatrix}, \tag{S14}$$

$$\frac{\partial \boldsymbol{\mu}_i^c}{\partial q_j^c} = 2 \begin{bmatrix} -2q_j^c x_i & q_i^c y_i & q_r^c z_i \\ q_i^c x_i & 0 & q_k^c z_i \\ -q_r^c x_i & q_k^c y_i & -2q_j^c z_i \end{bmatrix}, \tag{S15}$$

$$\frac{\partial \boldsymbol{\mu}_i^c}{\partial q_k^c} = 2 \begin{bmatrix} -2q_k^c x_i & -q_r^c y_i & q_i^c z_i \\ q_r^c x_i & -2q_k^c y_i & q_j^c z_i \\ q_i^c x_i & q_j^c y_i & 0 \end{bmatrix}. \tag{S16}$$

Multiplying the two, gives the final gradient with respect to the rotation $\mathbf{q}_c$.

Similar as for the color, the following term can be derived for depth:

$$\frac{\partial E_{\text{geo}}}{\partial \mathbf{T}_c} = \frac{\partial E_{\text{geo}}}{\partial d_p} \sum_{i \in N} \left(\frac{\partial d_p}{\partial d_i} \frac{\partial d_i}{\partial \mathbf{T}_c} w_i \prod_{j=1}^{i-1} (1 - w_j) + d_i \frac{\partial d_p}{\partial w_i} \frac{\partial w_i}{\partial \mathbf{T}_c} \right). \tag{S17}$$

It can be seen that the second term inside of the sum (*i.e.*, the gradient with respect to w_i) is very similar as in the derivation of the gradient with respect to the color; see Eq. (S6). $\frac{\partial w_i}{\partial \mathbf{T}_c}$ was already shown and the other gradients are calculated similar as for the color, using the depth instead. For the remaining term, again, only the second part is specific for the camera pose estimation and is therefore derived further. The other part can be calculated in the same way as the gradient $\frac{\partial \mathbf{c}_p}{\partial \mathbf{c}_i}$ for the color term and is standard for 3DGS.

Again, we can apply the chain rule and get

$$\frac{\partial d_i}{\partial \mathbf{T}_c} = \frac{\partial d_i}{\partial \boldsymbol{\mu}_i^c} \frac{\partial \boldsymbol{\mu}_i^c}{\partial \mathbf{T}_c}. \tag{S18}$$

From $\boldsymbol{\mu}_i^c$ the depth d_i of Gaussian i is calculated as

$$d_i = ||\boldsymbol{\mu}_i^c||. \tag{S19}$$

Using this, the first term in Eq. (S18) can be calculated:

$$\frac{\partial d_i}{\partial \boldsymbol{\mu}_i^c} = \left(\frac{\partial}{\partial x_i}, \frac{\partial}{\partial y_i}, \frac{\partial}{\partial z_i} \right)^T \sqrt{x_i^2 + y_i^2 + z_i^2} = \frac{\boldsymbol{\mu}_i^c}{d_i}. \tag{S20}$$

Further, $\frac{\partial \boldsymbol{\mu}_i^c}{\partial \mathbf{T}_c}$ can be split up in a translation and rotation part, $\frac{\partial \boldsymbol{\mu}_i^c}{\partial \mathbf{t}_c}$ and $\frac{\partial \boldsymbol{\mu}_i^c}{\partial \mathbf{R}_c}$, with

$$\frac{\partial \boldsymbol{\mu}_i^c}{\partial \mathbf{t}_c} = \mathbf{I}_3. \tag{S21}$$

Since the depth is calculated from the Gaussian mean in camera space, it is rotation invariant and, therefore, the rotational part is zero.

S2. Parameters

- Valid pixel intensity threshold: $\lambda_{\text{rgb}} = 0.01$
- Edge threshold scale: $\lambda_{\text{edge}} = 1.1$
- Minimum depth: $d_{\text{min}} = 0.01$
- Maximum depth: $d_{\text{max}} = 100$
- Minimum opacity: $s_{\text{min}} = 0.95$
- Tracking iteration: $\text{it}_{\text{track}} = \begin{cases} 100 & \text{outdoor} \\ 50 & \text{other} \end{cases}$
- Tracking convergence threshold (*i.e.*, minimum relative pose-change to stop tracking iterations): $\epsilon_{\text{pose-conv}} = 10^{-5}$
- Weighting parameter for loss calculation: $\lambda_{\text{pho}} = 0.95$
- Position learning rate schedule: $\eta_p = \begin{cases} 0.0016 & \text{init} \\ 1.6 \times 10^{-6} & \text{final} \end{cases}$
- Number of active keyframes: $n_{\mathcal{W}_k} = 8$
- Median depth scale for new keyframe: $\lambda_{\text{cov}} = 0.05$
- Co-visibility overlap for new keyframe: $\tau_{\text{cov,max}} = 0.9$
- New keyframe mapping iterations: $\text{it}_{\text{kf}} = 10$
- Max neighbor distance for keyframe removal: $\tau_{\text{dist}} = \begin{cases} 0.1 & \text{RGB} \\ 0.75 & \text{RGBD} \\ 7.5 & \text{RGBD outdoor} \end{cases}$
- Similarity threshold for keyframe removal: $\tau_{\text{sim}} = \begin{cases} 0.1 & \text{RGB} \\ 0.5 & \text{RGBD} \end{cases}$
- Pixel fraction for initial point selection: $k_{\text{init}} = 32$
- Pixel fraction for point selection: $k_{\text{run}} = 32$
- Initial mapping iterations: $n_{\text{init}} = \text{it}_{\text{init}} = 1050$
- Isometric loss mixing factor: $\lambda_{\text{iso}} = 10$
- Feature learning rate (appearance/color): $\eta_f = 0.0025$
- Opacity learning rate: $\eta_\alpha = 0.05$
- α-threshold for Gaussian removal: $\epsilon_\alpha = 0.7$
- α-threshold for Gaussian removal during initialization: $\epsilon_{\alpha\ \text{init}} = 0.005$
- Mapping iterations between adaptive density control: $\text{it}_{\text{adc}} = 150$
- Densification gradient threshold: $\tau_{\text{dens}} = 7.5 \times 10^{-5}$
- Gaussian extent (scene scale for filtering/clone-split-prune behavior): $\gamma_g = 1.0$

S3. Dataset

S3.1. Real world dataset

The real sequences were capture by two camera systems: an *Insta360 Pro* and an *Insta360 X4*. To enable free camera movement, the systems were attached on a *myAGV* by *Elephant Robotics*[1]; remotely controlled. The robot is specifically designed for recording SLAM sequences and comes with four mecanum wheels; allowing for turning while maintaining its position. Note that we did not use internal sensors of the myAGV. The camera systems were attached via a standard 1/4-20 UNC thread, which was mounted at the top side of the myAGV; Fig. S1 illustrates the setup. The larger *Insta360 Pro* was mounted directly and the smaller *Insta360 X4* with an additional telescopic stick; reducing occlusions, but introducing more vibrations.

Figure S1. The *Insta360 Pro* mounted on the *Elephant robotics myAGV* robot (left). The mounted *Insta360 X4* camera using a telescopic extension stick (right). A 1/4-20 UNC connection point was attached to the top side of the robot to mount the cameras.

To generate positional data, the sequences were recorded in a room equipped with an *OptiTrack*[2] system; a high-precision motion capture system that employs multiple synchronized infrared cameras to triangulate the 3D position and orientation of retro-reflective markers. It provides sub-millimeter accuracy and low-latency tracking. Therefore, three reflective markers were placed on the robot. The output is position per marker and a user defined oriented center position of a rigid body. Here, the center was placed at the mount point. Time-synchronization between positional data and the recorded video was achieved by manually aligning the movement patterns. The *OptiTrack* system measurement area covered four by three meters. Various objects with different shapes, sizes, and colors were placed to challenge the SLAM system's capabilities with respect to occlusions (tracking stability) and reconstruction of fine grain features (mapping quality); see Figure Fig. S2.

S3.1.1. Hardware and Recording Output

Five sequences were filmed with the *Insta360 Pro* camera; recorded in the single-camera mode. Per sequence, six separate vertical fisheye videos at 1440×1920 pixels resolution and 30 fps were created. Videos were converted to image sequences via `ffmpeg`. The six separate frames were stitched into one omnidirectional image of 1920×960 pixels resolution. Stitched panorama images and depth were acquired using [4]; extended to support the OpenCV fisheye camera model. The individual fisheye images were

[1] Elephant Robotics myAGV: elephantrobotics.com/en/myagv-en.

[2] OptiTrack: optitrack.com.

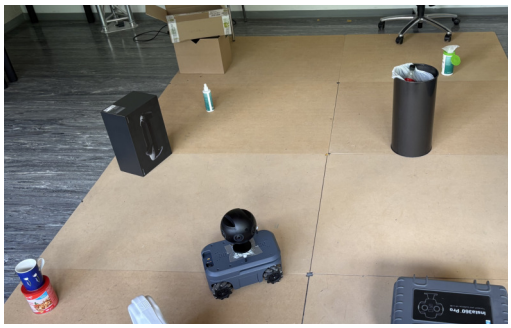


Figure S2. Objects with different shapes, sizes, and colors were arranged for the real world dataset to challenge the SLAM system.

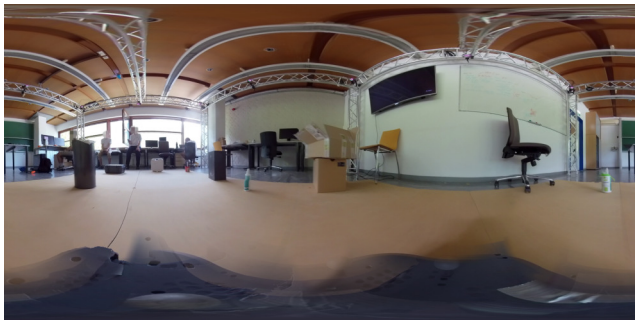
un-distorted to be compatible to perspective SLAM systems. Two *exploration* sequences were created, where the robot freely navigates through the scene; including loops and abrupt stationary periods to mimic realistic scenarios. They have 703 and 954 frames, respectively. Three *testing* sequences were created to specifically assess: isolated directional movement, forward-backward motion, and retracing previously traversed trajectories. They have 405, 354, and 486 frames, respectively. Figure Fig. S3 shows examples for all available inputs for the Insta360 Pro sequences.

The *Insta360 X4* camera can provide robustly stitched 360° images, with minimal artifacts at overlapping boundaries. It can combine the two individual fisheye recordings. The resulting videos have a resolution of 3840×1920 pixels at 60 fps and were also converted to images. For these recordings no stereo data is available. Thus, no depth data is available, here. Furthermore, the lack of individual lens data prevents un-distortion into perspective images. Only SLAM systems supporting equirectangular input could be employed. Eight sequences were created with the *Insta360 X4*: four mounted directly to the robot and four with the camera mounted higher via the telescopic extension. For each setup, one *exploration* sequence and three *testing* sequences were generated; similarly to the sequences described above. Frame counts are 1428, 899, 485, and 588 (direct) and 2792, 839, 736, and 1549 (extension), respectively. Example images from the Insta360 X4 sequences for both mount types are shown in Fig. S4.

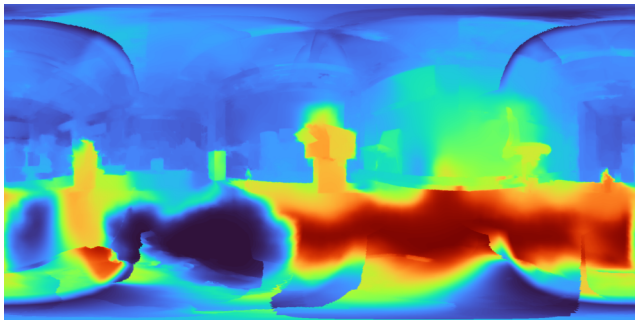
S3.2. Virtual dataset

We rendered virtual sequences created in *Blender* via its renderer *Cycles*[3] for full scene and motion control and best ground truth quality. Therefore, we extended Cycles to support realistic fish eye lens distortions taking measured k values from our camera calibration as input [5]; a work initially started via a bachelor thesis [2]. We used the intrinsics and

[3]Blender: blender.org, Cycles: cycles-renderer.org

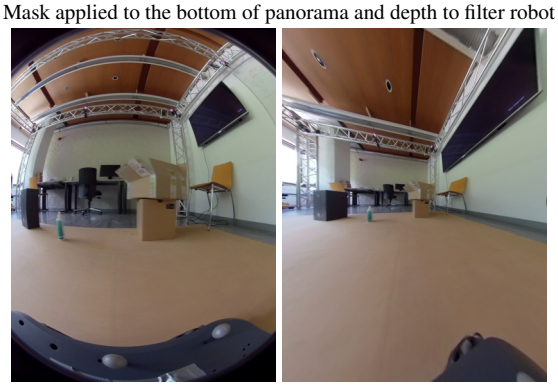


Panorama stitched with [4]



Depth estimated with [4]

Mask applied to the bottom of panorama and depth to filter robot



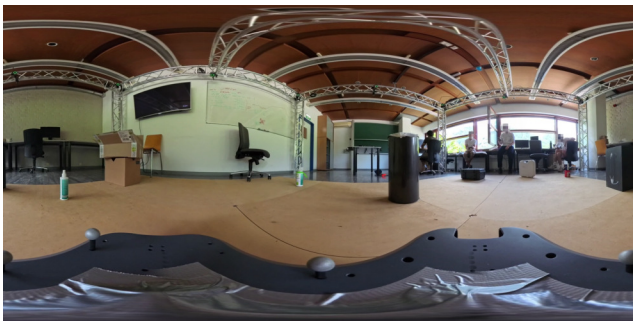
Raw camera image Undistorted camera image

Figure S3. Example images from the Insta360 Pro sequences.

extrinsics parameters of the *Insta360 Pro* camera obtained with MC-Calib. In *Blender*, an omnidirectional six-camera setup was created, animated, and rendered for each test sequence. Each of the individual cameras was rendered at 1440×1920 pixels and 30 fps. Further, a panoramic camera for rendering 360° images with 3840×1920 pixels at 30 fps was added. For the evaluation we mostly directly used the panoramic outputs; unless stated otherwise. The chosen formats are compatible with output of the *Insta360 Pro*. Depth data is provided besides the RGB image sequence. Finally, a standard perspective camera with 113° vertical and 97°

Panorama extension mount

Mask applied to bottom of panorama in extension mount sequences



Panorama direct mount

Mask applied to bottom of panorama in direct mount sequences

Figure S4. Example images from the Insta360 X4 sequences.

horizontal FoV was added and rendered at 720×960 pixels as well, to enable the evaluation of perspective systems.

To support various scenarios we created sequences for two scene types: indoor and outdoor.

Indoor: The *Italian Flat* scene by Flavio della Tommasa [6][4] was utilized. It represents a photo-realistic kitchen and a living room ($3.5 \times 10\ m$) containing fine grain details, reflections, and translucent materials to challenge SLAM systems. Two *exploration* sequences were developed. Firstly, one was designed to test mapping capabilities. It covers the entirety of both rooms from various angles with simple movements: starting in the kitchen, moving to the living room, covering it entirely, and returning to the kitchen. Secondly, a sequence in the living room features more complex and abrupt movements to test tracking capabilities. The camera leaves already observed regions and returns to them later to test global consistency. Three more sequences are designed to evaluate positional and rotational tracking accuracy as well as the key frame removal process. The first sequence moves the camera back and forth in x, y, and z direction, sequentially. The second rotates the camera constantly until returned to the start position. The last sequence is a one axial forward movement, followed by a 180°turn, and returning to the start position. The five sequences provide 1500, 1400, 1200, 500, and 250 frames. Examples for all available inputs from the rendered indoor sequences are depicted in Fig. S5.

[4]Blender Demo Files: blender.org/download/demo-files

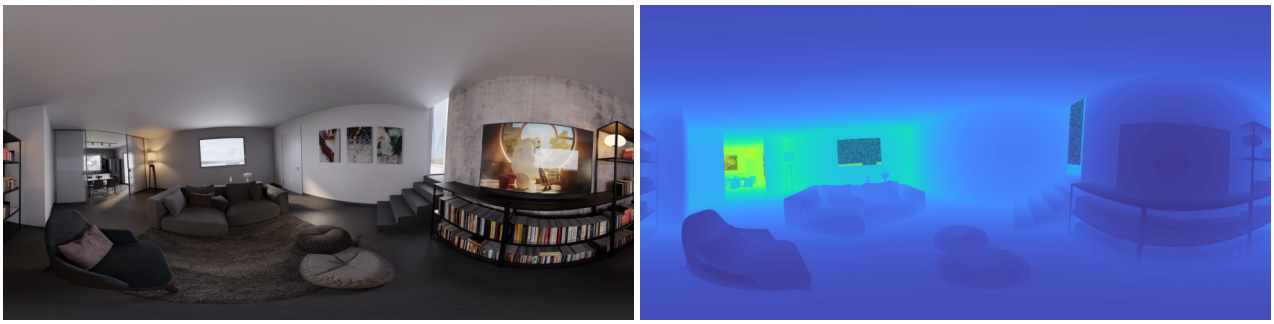
Outdoor: Here, the virtual urban scene from [8] was employed. It replicates an urban environment with buildings of various sizes. It already comes with an exploratory animation including challenges such as loops, sharp turns, re-visiting of already observed regions, and a visible sky. Compared to the indoor scenes, the distances in the outdoor scene are far greater and at the sky no tracking information can be computed (infinite distance). We replaced the single camera by an omnidirectional six-camera setup mimicking the *Insta360 Pro* and a long sequence of 3000 frames. Fig. S6 shows examples of all available input for the rendered outdoor sequence.

S4. Trajectory Plots

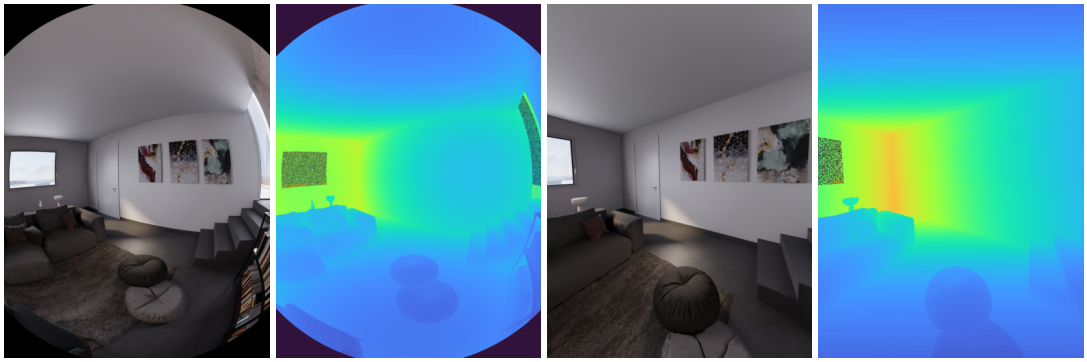
In addition to the quantitative evaluation in the main paper, the resulting mean trajectories for selected sequences and all applicable methods are compared in Figs. S7 to S12. See Secs. 4 and 5 in the main paper for the naming conventions of methods and sequences. Note, that the trajectory for a method are obtained by calculating the mean position for each frame. For RGBD runs the resulting mean trajectories are rotation-aligned to minimize the error to the ground truth. RGB runs are additionally scale-aligned. The resulting trajectories are then plotted together with the ground truth.

S5. Renderings

Further, to provide a qualitative evaluation of the rendering performance, the render results of some selected keyframes are presented in Figs. S13 and S15. The difference images are created by calculating the pixel-wise difference and then enhancing the contrast by multiplying the resulting intensity values by a gain factor of two (for better visualization). For a qualitative comparison, similar keyframes were rendered with MonoGS and GASL too. These are depicted in Figs. S14 and S16. Further, to show how the map looks from unseen positions, corresponding renders are provided in Fig. S17.

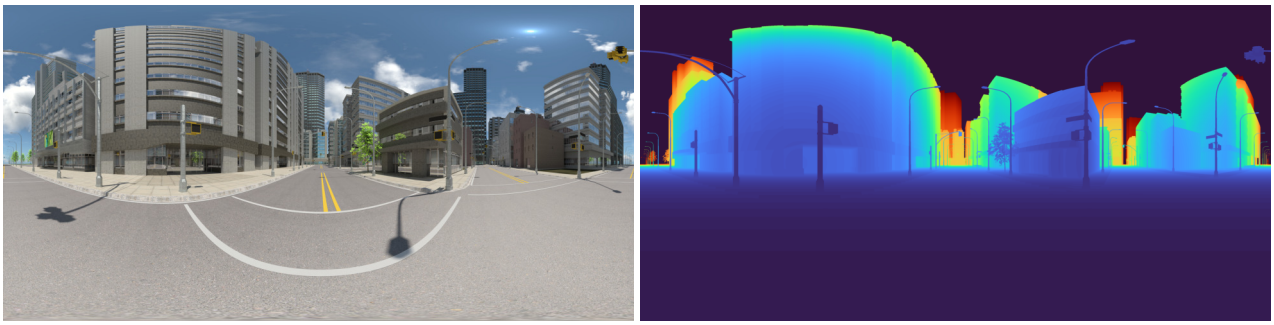


Cycles panorama render | Panorama depth

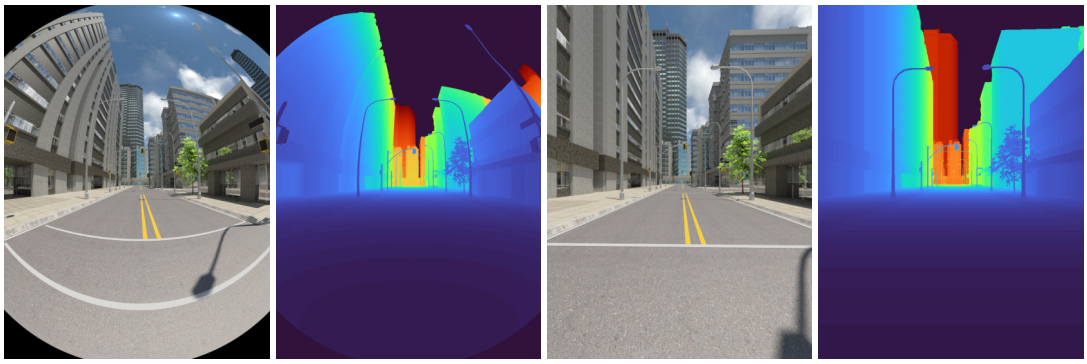


Cycles Insta360 Pro raw render | Insta360 Pro raw depth | Cycles pinhole render | Pinhole depth

Figure S5. Example images from the rendered indoor sequences.



Cycles panorama render | Panorama depth



Cycles Insta360 Pro raw render | Insta360 Pro raw depth | Cycles pinhole render | Pinhole depth

Figure S6. Example images from the rendered outdoor sequence.

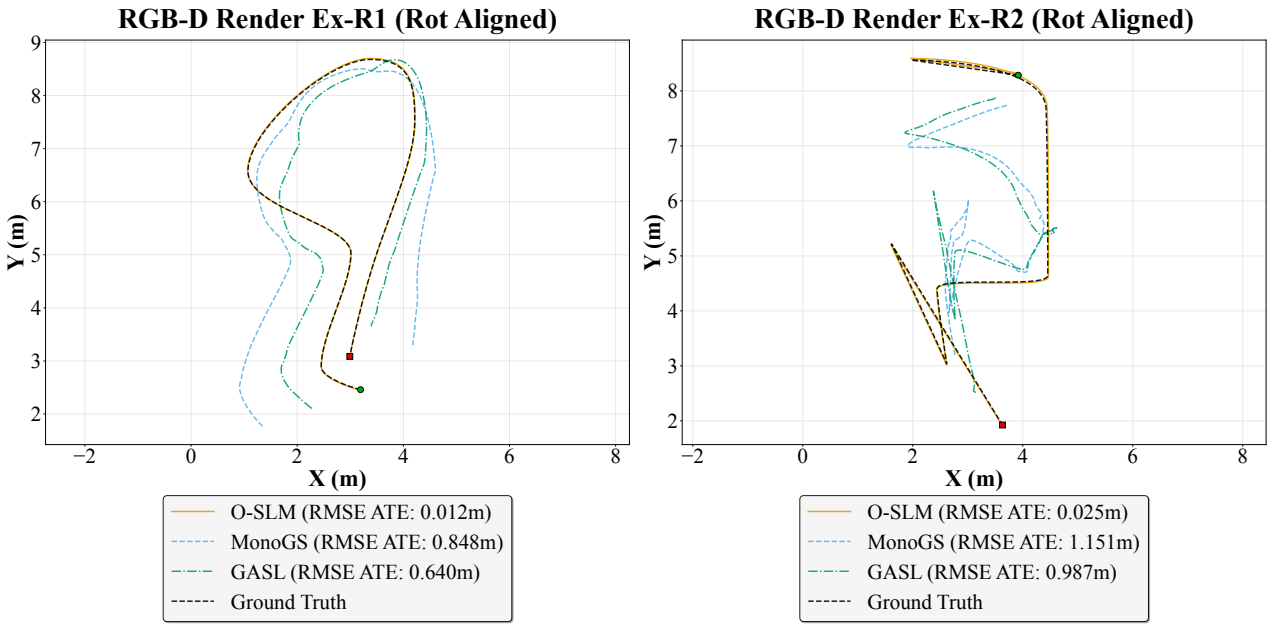


Figure S7. RGBD exploration sequences in the rendered room scene. Both 3DGS-SLAM methods using perspective input images struggle with tracking, in particular for the more complex movements in exploration sequence two (see right plot).

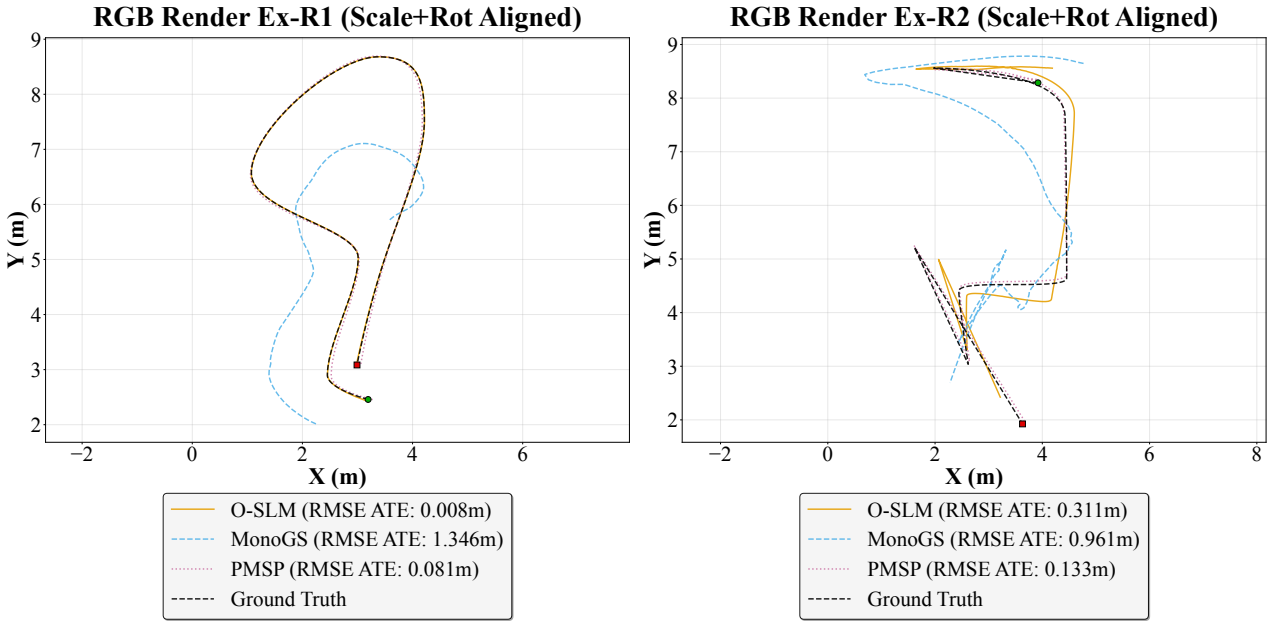


Figure S8. RGB exploration sequences in the rendered room scene. It can be seen, that MonoGS struggles with tracking. This is most likely, due to the missing scale information. Also for O-SLM, a similar, but much less sever effect can be observed in exploration 2 (right).

S6. Parameter Study

The influence of different parameters on the system performance was investigated in a small parameter study. The image size, point cloud sampling rate k_{run}, tracking iterations it_{track}, and densification/tracking-convergence thresholds τ_{dens} and $\epsilon_{\text{pose-conv}}$ were selected as most promising parameters for reducing the runtime. Tab. S1 shows the different parameter sets that were tested, with *Control* indicating the parameters used for the main part of this work. Each parameter set was run three times using RGB and RGBD data on the full exploration one (Ex1) synthetic room scene. Results are reported in Figs. S18 and S19 and Tab. S2.

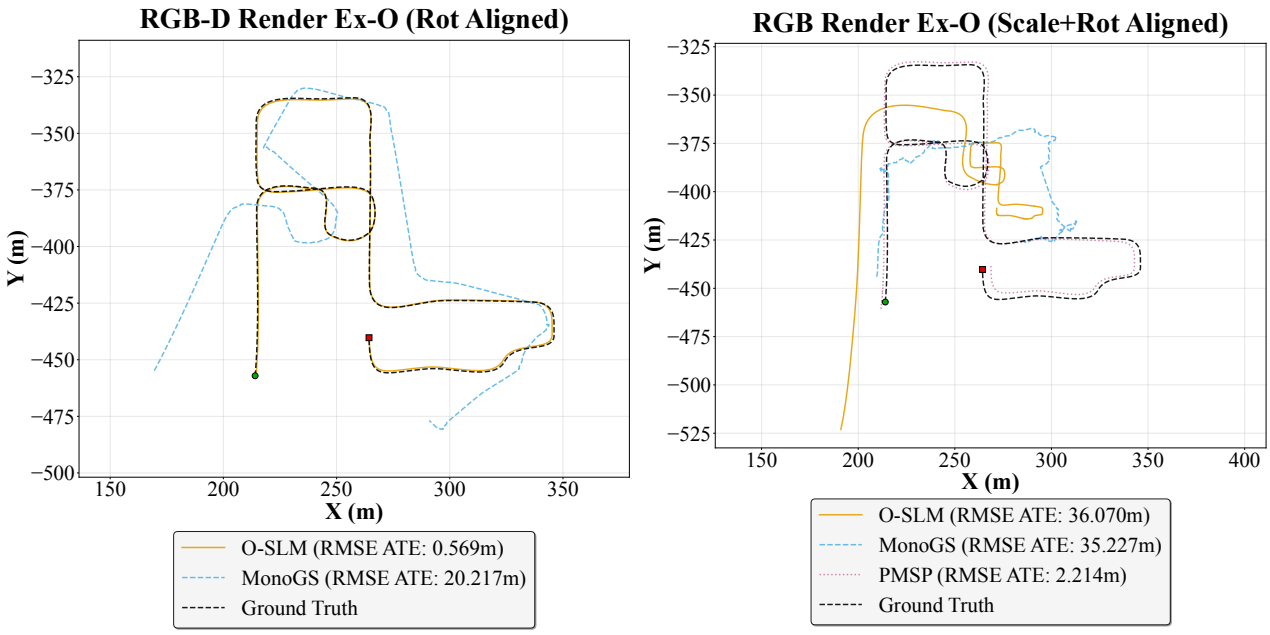


Figure S9. Rendered outdoor exploration sequences using RGBD (left) and RGB (right) as input. For RGBD, GASL is not included, since it could not handle the large scene due to memory consumption. For RGB, only PMSP handles tracking well. O-SLM most likely has issues with estimating scale constantly over the whole run. Similar issues could also be observed for the other 3DGS-SLAM methods.

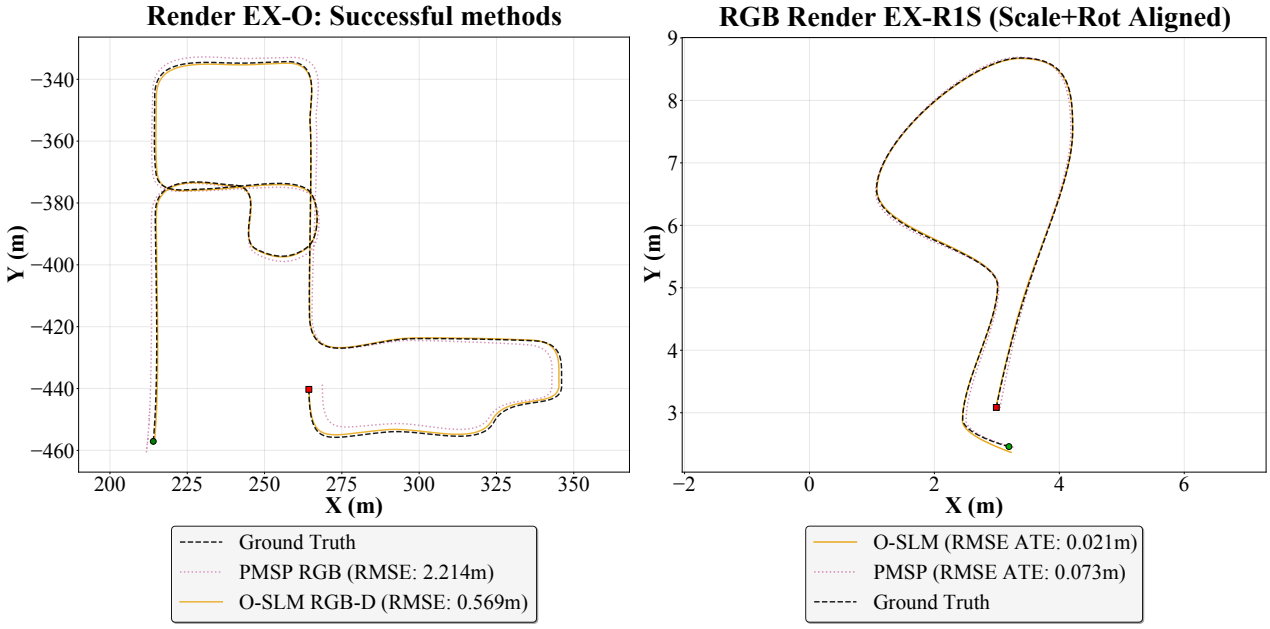
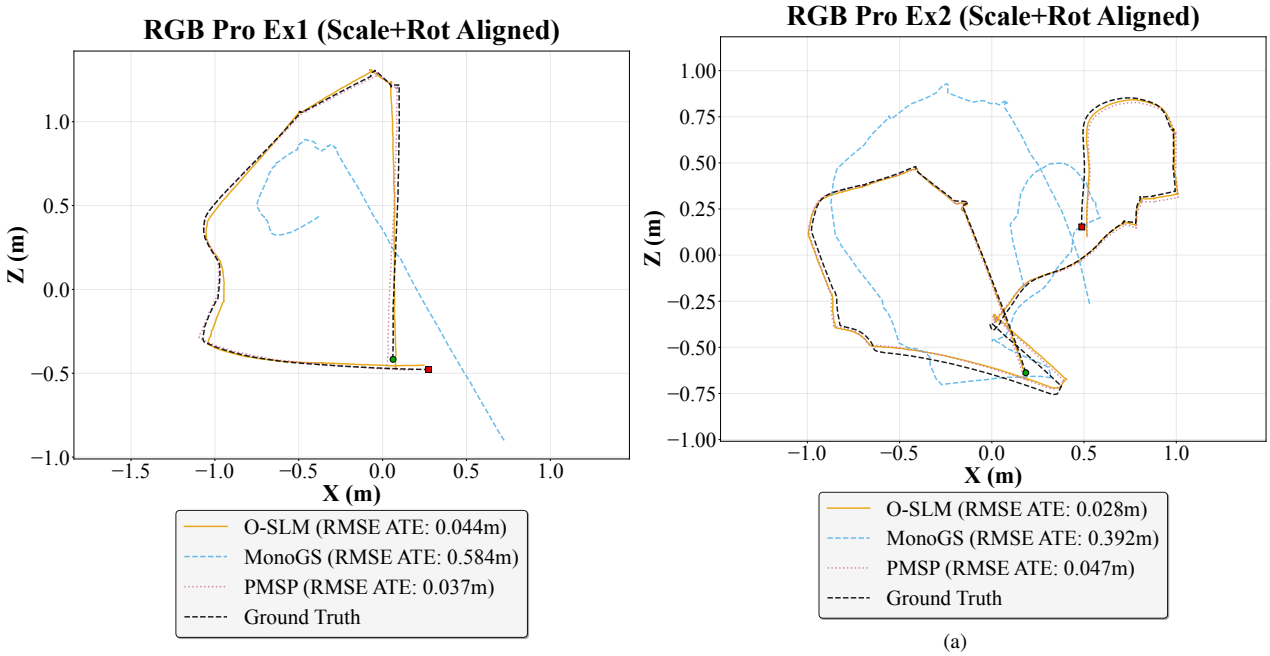


Figure S10. (left) Comparison of the only methods that successfully track the outdoor sequence. Note that PMSP uses RGB data, while O-SLM also requires depth. (right) Exploration 1 sequence in render room scene using the stitched panorama images, to show that methods also work for stitched panoramas.

S7. Real-World Ablation Study

In addition to the ablation study in the synthetic room scene (see paper; Sec. 5), evaluations on real-world sequences are currently also carried out. This is part of our ongoing work; feel free to contact the authors for up-to-date details.



(a)

Figure S11. RGB Insta360 Pro exploration sequences. Also here it can be seen that MonoGS generally manages to track the movements, but struggles with missing scale and as a result fails. Further, for MonoGS, the forward facing fisheye camera output of the Insta360 Pro was undistorted and used as input. Due to the strong distortion, this could be another factor contributing to its failure.

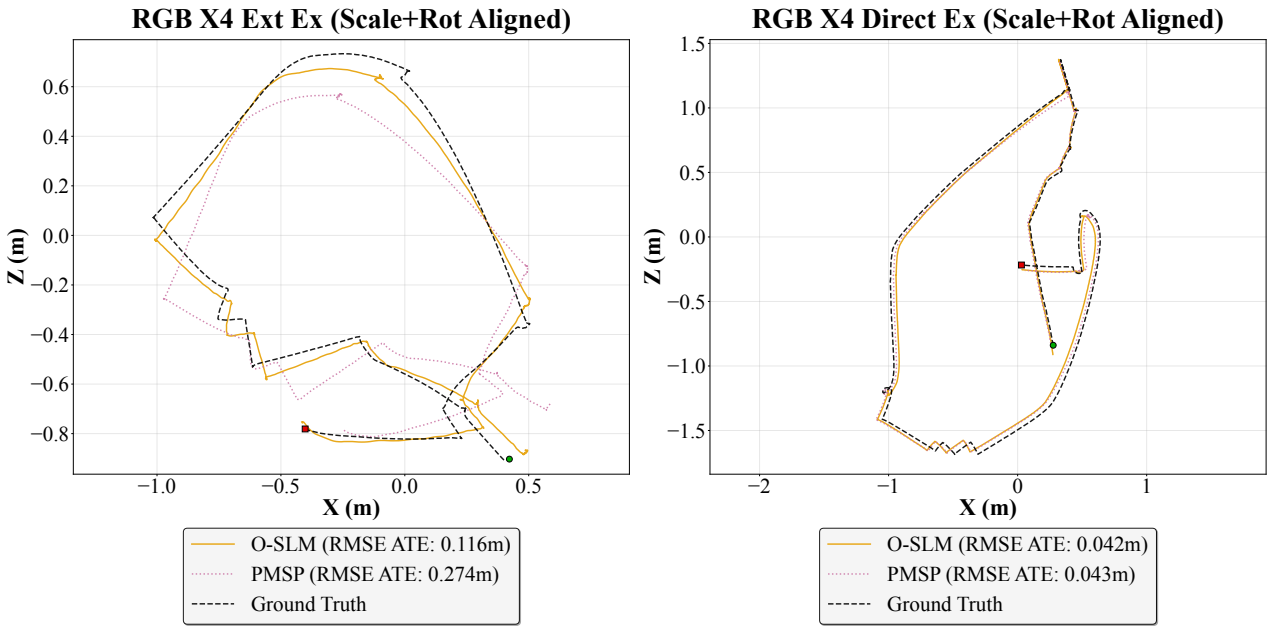


Figure S12. RGB exploration sequences captured with the Insta360 X4 for extension (left) and direct mount (right).

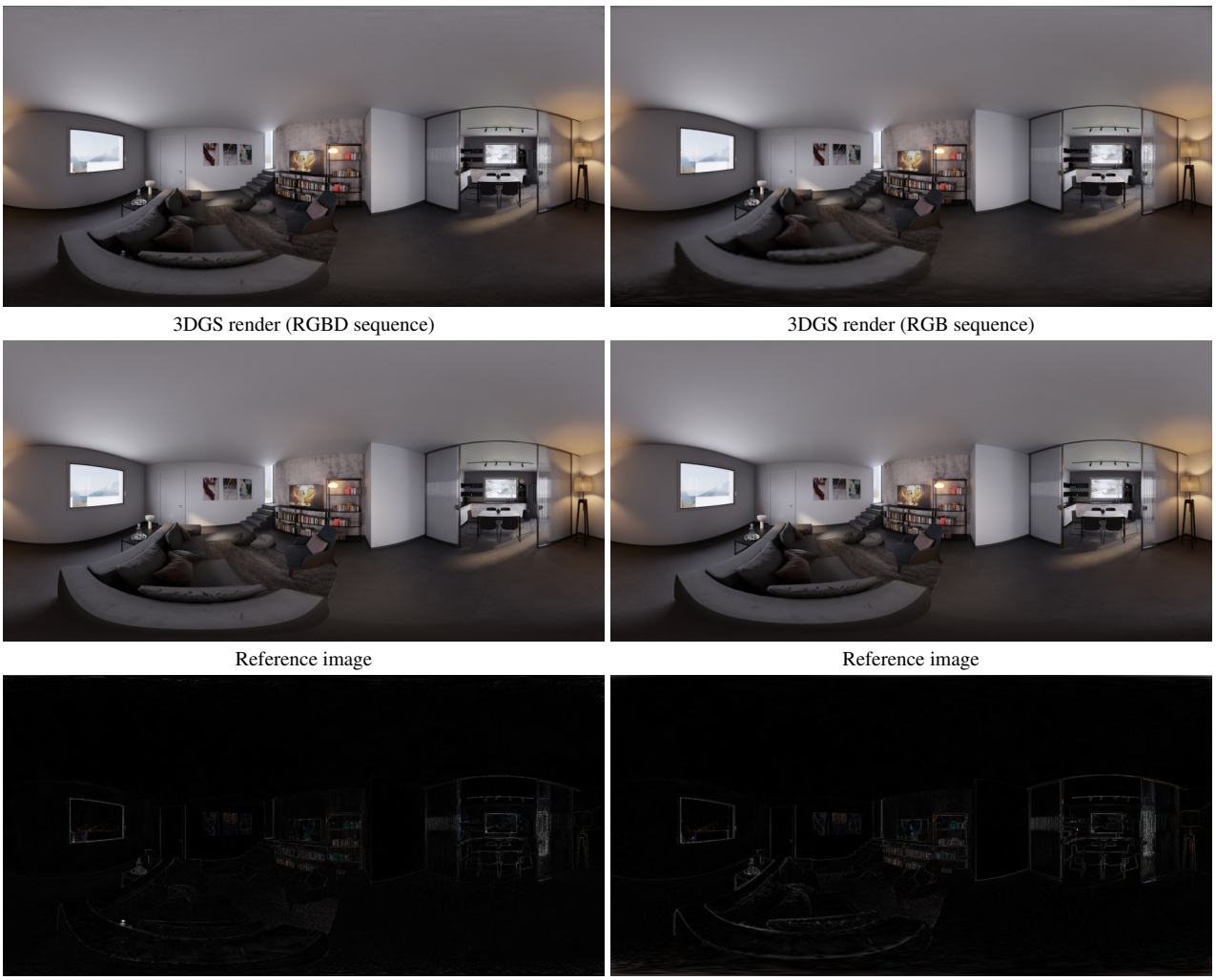
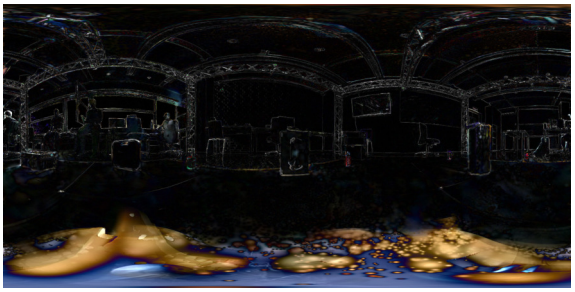


Figure S13. Visual comparison of rendering and groundtruth input for one selected keyframe of a RGBD (left) and a RGB (right) synthetic room sequence. For the difference images, the contrast was increased by multiplying the pixel-wise differences with a gain of two for better visualization.

Figure S14. Groundtruth input and rendering results with other 3DGS-based SLAM methods in synthetic room sequence.

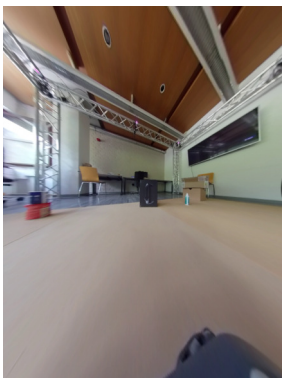
3DGS render (RGB sequence)

Reference image

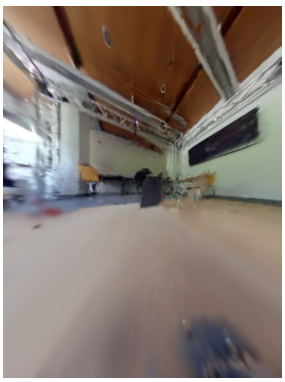


Pixel-wise difference (RGB sequence)

Figure S15. Visual comparison of rendering and groundtruth input for selected keyframe of a RGB Insta360 Pro sequence. The contrast in the difference was increased by multiplying with a gain of two for better visualization. Note that the large errors at the bottom are due to the robot, which is moving with the camera and is masked during tracking and mapping.



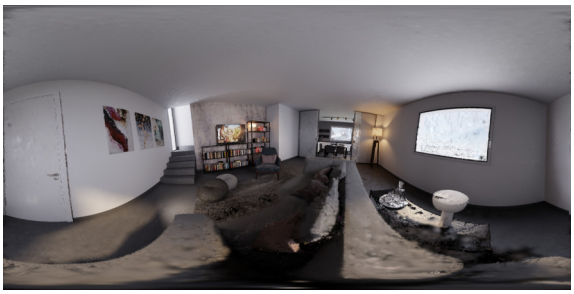
Reference image



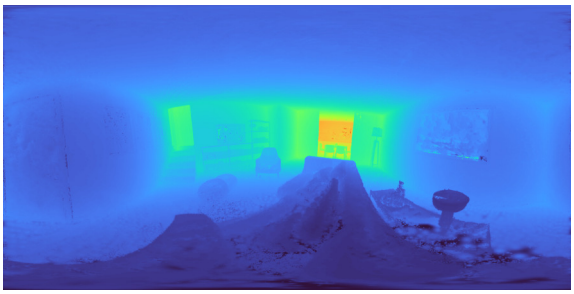
MonoGS render (RGB sequence)

Figure S16. Groundtruth input image and rendering result of MonoGS for similar keyframe as in Fig. S15

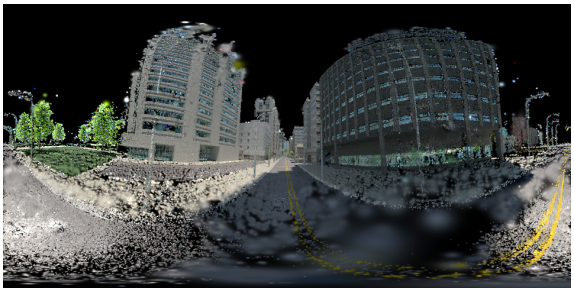
.



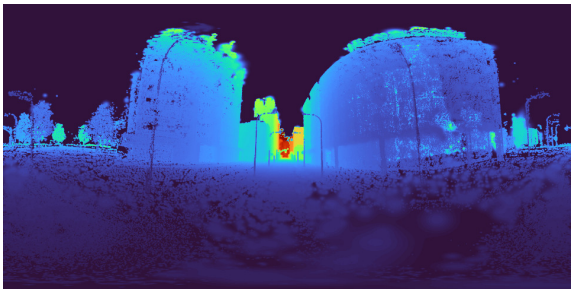
3DGS render indoor



3DGS depth indoor



3DGS render outdoor

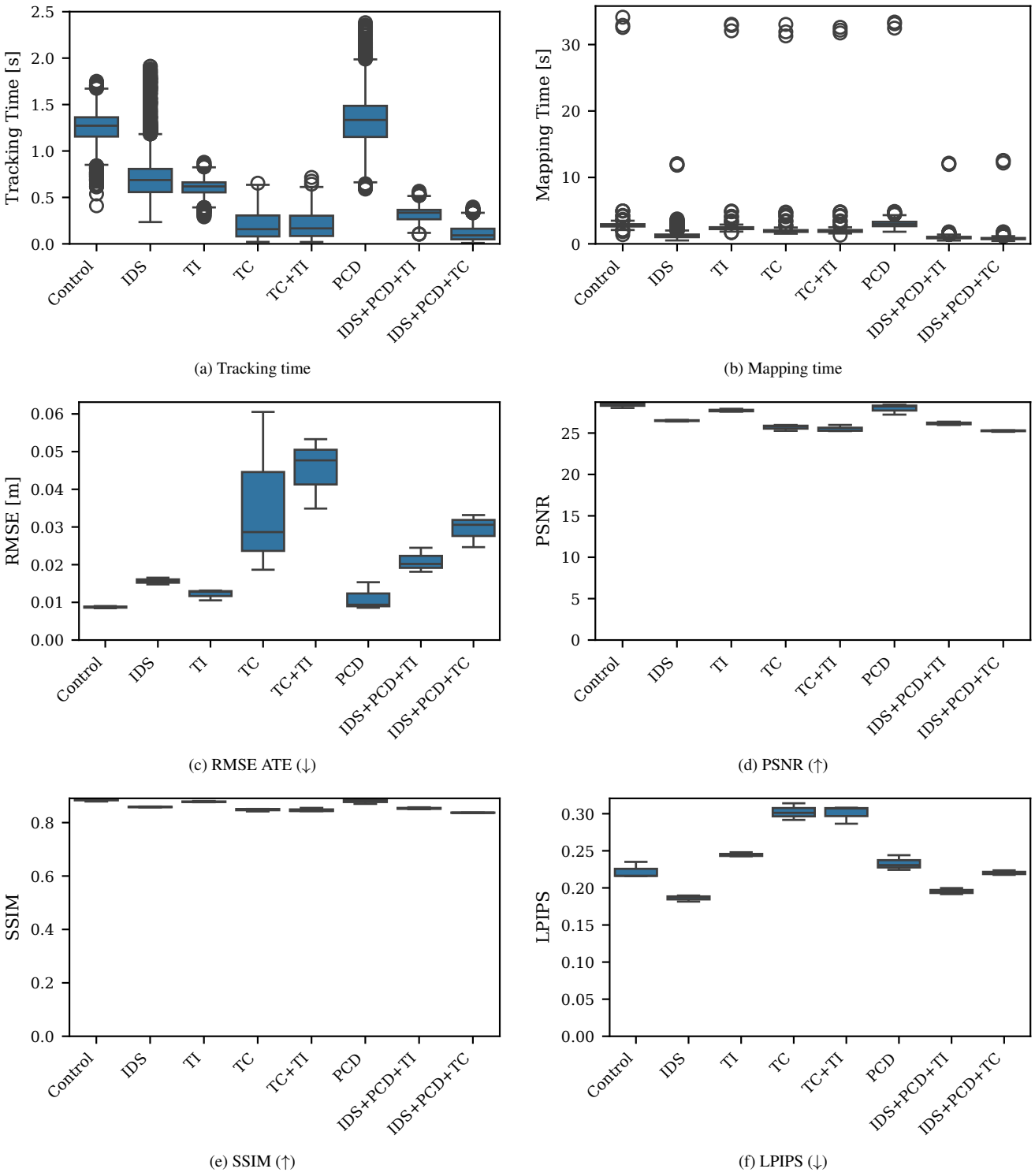


3DGS depth outdoor

Figure S17. 3DGS map renderings from unvisited viewpoints for maps generated with RGBD input.

Table S1. Parameter study configurations.

Parameter Set	k_{img}	τ_{dens}	it_{track}	$\epsilon_{\text{pose-conv}}$	k_{run}
Control	2	7.5×10^{-5}	50	10^{-5}	32
IDS	4	1.25×10^{-4}	50	10^{-5}	32
TI	2	7.5×10^{-5}	25	10^{-5}	32
TC	2	7.5×10^{-5}	50	10^{-3}	32
TC+TI	2	7.5×10^{-5}	25	10^{-3}	32
PCD	2	7.5×10^{-5}	50	10^{-5}	64
IDS+PCD+TI	4	1.25×10^{-4}	25	10^{-5}	64
IDS+PCD+TC	4	1.25×10^{-4}	50	10^{-3}	64



(a) Tracking time

(b) Mapping time

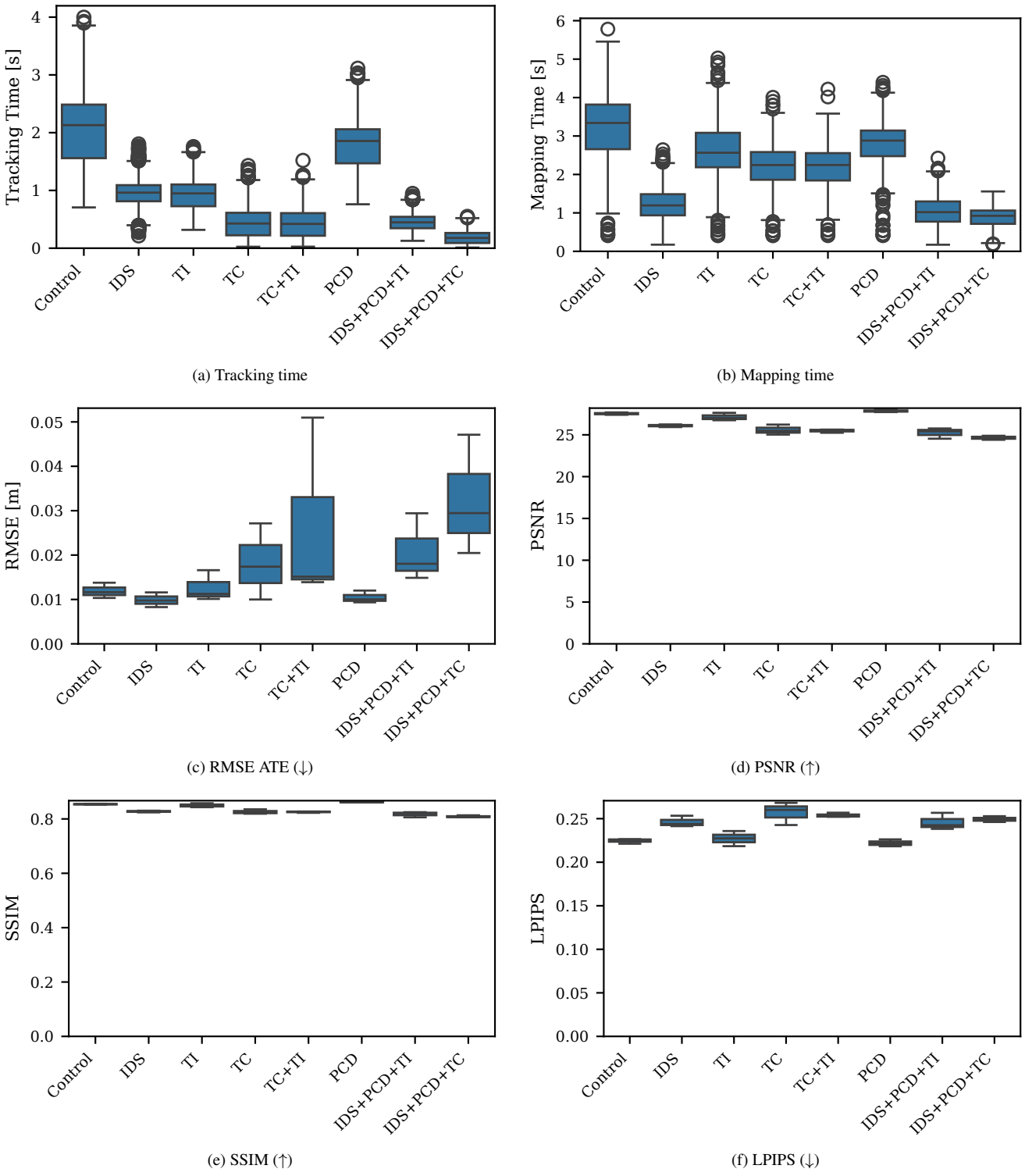
(c) RMSE ATE (↓)

(d) PSNR (↑)

(e) SSIM (↑)

(f) LPIPS (↓)

Figure S18. RGB parameter study results across different parameter configurations. See Tab. S1 for more detail about the different configurations.



(a) Tracking time

(b) Mapping time

(c) RMSE ATE (↓)

(d) PSNR (↑)

(e) SSIM (↑)

(f) LPIPS (↓)

Figure S19. RGBD parameter study results across different parameter configurations. See Tab. S1 for more detail about the different configurations.

Table S2. Parameter study results (mean values) for RGB and RGBD input. Best values per metric are highlighted in bold.

Metric	Unit	Control	IDS	TI	TC	TC+TI	PCD	IDS+PCD+TI	IDS+PCD+TC
					RGB				
FE RAM	MB	1942.1	1834.2	1931.3	1932.5	1931.0	1967.2	**1833.1**	1833.9
BE RAM	MB	1647.4	1607.8	1620.9	1608.9	1603.7	1676.0	1592.6	**1576.9**
FE GPU	MB	6908.3	5058.1	6235.7	5831.9	5932.0	7485.2	4645.3	**4528.3**
BE GPU	MB	7546.9	5369.3	6834.0	6331.7	6451.3	8113.2	4833.1	**4650.2**
Track time	s	1.259	0.723	0.607	0.193	0.198	1.339	0.321	**0.112**
Map time	s	3.036	1.301	2.731	2.382	2.388	3.206	0.983	**0.833**
FPS	Hz	0.589	0.878	0.961	1.613	1.597	0.565	1.364	**1.925**
RMSE ATE	m	**0.009**	0.016	0.012	0.036	0.045	0.011	0.021	0.029
PSNR (↑)	dB	**28.38**	26.50	27.73	25.68	25.51	27.95	26.16	25.26
SSIM (↑)	/	**0.885**	0.859	0.878	0.848	0.847	0.880	0.853	0.837
LPIPS (↓)	/	0.222	**0.186**	0.245	0.302	0.301	0.233	0.195	0.220
					RGBD				
FE RAM	MB	2386.6	2183.8	2316.1	2347.2	2314.7	2335.8	2148.8	**2127.2**
BE RAM	MB	2135.6	1930.6	2179.1	2151.4	2154.4	2154.6	1928.1	**1925.2**
FE GPU	MB	12600.7	7401.9	12161.0	11417.9	11576.9	10873.1	7088.4	**6956.3**
BE GPU	MB	13822.1	7784.1	13235.9	12339.5	12514.0	11911.4	7276.7	**7136.0**
Track time	s	2.066	0.944	0.948	0.433	0.427	1.794	0.451	**0.184**
Map time	s	3.268	1.224	2.654	2.240	2.222	2.797	1.069	**0.902**
FPS	Hz	0.332	0.537	0.534	0.750	0.753	0.371	0.757	**0.967**
RMSE ATE	m	0.012	**0.010**	0.013	0.018	0.027	0.010	0.021	0.032
PSNR (↑)	dB	27.53	26.09	27.13	25.58	25.48	**27.85**	25.24	24.65
SSIM (↑)	/	0.854	0.827	0.850	0.826	0.826	**0.862**	0.817	0.809
LPIPS (↓)	/	0.225	0.246	0.227	0.257	0.254	**0.222**	0.246	0.249

References

[1] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM TOG*, 42(4):1–14, 2023. 1

[2] Stefan Kuhnert. Virtual Multi-Camera Systems in Blender. Bachelor's thesis, University of Innsbruck, 2023. 4

[3] Hidenobu Matsuki et al. Gaussian Splatting SLAM. In *CVPR*, pages 18039–18048, 2024. 1

[4] Andreas Meuleman, Hyeonjoong Jang, Daniel S. Jeon, et al. Real-Time Sphere Sweeping Stereo From Multiview Fisheye Images. In *CVPR*, pages 11423–11432. Computer Vision Foundation / IEEE, 2021. 3, 4

[5] Stefan Spiss and Stefan Kuhnert. Wide Angle Lenses - Blender PlugIn for Cylces. https://github.com/stefan-spiss/blender-cv-cam-extensions, 2026. 4

[6] Flavio Della Tommasa. Italian Flat. `https://www.malapixel.com/mal-house.html`, accessed: 31.03.2025. 5

[7] Chi Yan, Delin Qu, Dan Xu, Bin Zhao, Zhigang Wang, Dong Wang, and Xuelong Li. GS-SLAM: Dense Visual SLAM with 3D Gaussian Splatting. In *CVPR*, pages 19595–19604, 2024. 1

[8] Zichao Zhang et al. Benefit of large field-of-view cameras for visual odometry. In *IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 801–808, 2016. 5

[9] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. EWA volume splatting. In *Proceedings Visualization*, page 29–538. IEEE, 2001. 1