# 3DGS-Calib: 3D Gaussian Splatting for Multimodal SpatioTemporal Calibration

Quentin Herau[1,3], Moussab Bennehar[1], Arthur Moreau[2], Nathan Piasco[1], Luis Roldão[1], Dzmitry Tsishkou[1], Cyrille Migniot[4], Pascal Vasseur[5] and Cédric Demonceaux[3]

***Abstract*— Reliable multimodal sensor fusion algorithms require accurate spatiotemporal calibration. Recently, targetless calibration techniques based on implicit neural representations have proven to provide precise and robust results. Nevertheless, such methods are inherently slow to train given the high computational overhead caused by the large number of sampled points required for volume rendering. With the recent introduction of 3D Gaussian Splatting as a faster alternative to implicit representation methods, we propose to leverage this new rendering approach to achieve faster multi-sensor calibration. We introduce 3DGS-Calib, a new calibration method that relies on the speed and rendering accuracy of 3D Gaussian Splatting to achieve multimodal spatiotemporal calibration that is accurate, robust, and with a substantial speed-up compared to methods relying on implicit neural representations. We demonstrate the superiority of our proposal with experimental results on sequences from KITTI-360, a widely used driving dataset.**

## I. INTRODUCTION

Sensors are essential in robotics and intelligent systems when it comes to providing knowledge about the surrounding environment. Depending on the type of the sensor, different information modalities are provided (e.g. geometric range information with LiDAR and color information with RGB cameras). Combining all information from multiple sensors allows scene understanding to be maximized, enabling improved performance in crucial tasks such as localization, mapping, and object detection. However, to correctly merge and exploit the data coming from the various sensors, it is necessary to first perform an accurate spatial and temporal calibration phase.

The classical strategy of performing spatial calibration is through the use of one or more targets of known characteristics carefully positioned around the scene, such as checkerboards [1] or other custom targets [2], [3]. The goal of these methods is to have multiple acquisitions of the target with both sensors (e.g. LiDAR depth sensor and RGB cameras) and to find matching information within the two modalities (plane detection in the LiDAR point cloud and corner detection in the camera RGB images). By finding correspondences, these methods can determine the correct spatial calibration. Classical methods are accurate and robust, but hardly scalable as they require manual placement of targets in addition to structural assumptions about the surrounding environment. Such requirements are incompatible with in-the-wild re-calibration where targets may not be available, or placing them might be impractical.

To remove these constraints, targetless calibration methods directly exploit existing information within the scene without requiring specific user-placed targets. These methods can either rely on specific features naturally present in the scene (e.g. edges [4], [5] or intensity [6], [7]) or extract features automatically by relying on neural networks [8], [9]. Although these methods do not require manual user-placed targets, they still depend on the presence of either specific features in the scene or are supervised using data provided by a sensor setup with known calibration parameters.

Recently, a new family of calibration methods based on implicit neural representations (i.e. NeRF: Neural Radiance Fields) has emerged [10], [11], [12] demonstrating impressive calibration results without requiring as many priors as classical methods. NeRF-based calibration strategies build a geometrically consistent and continuous volumetric representation of the scene enforcing consistency between all the sensors. Thus, these approaches achieve robust and accurate calibration without the limitations of the previously mentioned methods. Nevertheless, the extended training times associated with a NeRF model create a significant limitation hindering their adoption in real-world applications.

Recently, 3D Gaussian Splatting (3DGS) [13] was introduced as a novel 3D scene representation with significantly faster training times than NeRF and real-time rendering, while providing on-par rendering quality. Thanks to its advantages compared NeRF, 3D Gaussian Splatting has gained increasing popularity and has been applied to solve different tasks such as rendering human avatars [14] and reconstructing in-the-wild [15] or driving scenes [16].

Motivated by their advantages and given their shared properties with NeRF, we present in this paper **3DGS-Calib**, a novel targetless multimodal spatiotemporal calibration solution that capitalizes on the speed and quality of 3D Gaussian Splatting. By exploiting the LiDAR point cloud as a reference for the Gaussians' positions, we learn a continuous representation of the scene. Using this representation and enforcing both geometrical and photometric consistency between all the sensors enables us to achieve

[1] Noah's Ark, Huawei Paris Research Center, France. {Quentin.Herau, Nathan.Piasco, Moussab.Bennehar, Luis.Roldao, Dzmitry.Tsishkou}@huawei.com

[2] Noah's Ark, Huawei London Research Center, United Kingdom. Arthur.Moreau2@huawei.com

[3] ICB UMR CNRS 6303, Université de Bourgogne, France. {Quentin.Herau@etu., Cedric.Demonceaux@}u-bourgogne.fr

[4] ImViA UR 7535, Université de Bourgogne, France. Cyrille.Migniot@u-bourgogne.fr

[5] MIS UR 4290, Université de Picardie Jules Verne, France. Pascal.Vasseur@u-picardie.fr

accurate calibration requiring less training time compared to NeRF-based methods. Our method offers accurate and robust calibration and significantly outpaces existing NeRF-based methods.

In summary, our main contributions are as follows:

- To the best of our knowledge, we propose the first multimodal spatiotemporal calibration method based on 3DGS [13], which allows significant speed-up while providing better accuracy and robustness than NeRF-based methods,
- We evaluate our method on sequences from KITTI-360 [17], a widely used driving dataset, and show that our method provides better results on driving scenes for LiDAR/Camera calibration, without the need for any supervision or specific features in the scene.

## II. RELATED WORK

**Targetless Calibration.** Classical targetless multimodal calibration methods rely on specific features present in the scene that can be matched between the different modality sensors (e.g. LiDARs and RGB cameras) to achieve the calibration goal. For instance, Taylor et al. [6] and Pandey et al. [7] propose to apply a correspondence between the grayscale level of the color images and the intensity of the LiDAR scans. Nevertheless, such a correlation is not always correct (e.g. in the case of shadows within the scene). On the other hand, Napier et al. [4] and Yuan et al. [5] match edges detected in both the images and the LiDAR scans, to align the scene structures extracted from the different modalities. However, point cloud-based edge detection degrades with the sparsity of the acquisition, requiring the sensors to stay still during the process to gather more data or needing more expensive systems.

Other lines of work like CalibNet [8] and LCCNet [9] rely on the ability of neural networks to automatically extract high-level features for calibration with deep learning models. Such methods can predict the correct calibration between LiDAR scan and image pairs in real-time, enabling online re-calibration. Nevertheless, a labeled dataset is needed to train the models, creating a bias towards the type of scenes used for training. Furthermore, as shown in [12], these methods are setup-specific and need to be re-trained if there is a considerable difference with the training data.

More recently, Neural Radiance Fields (NeRF) [18] were introduced enabling an implicit representation of a 3D scene learned within a neural network. Seminal works have exploited this representation to solve a wide range of applications such as novel view synthesis [19], [20], localization [21], surface reconstruction [22] or pose registration [23], [24]. Notably, some recent works propose to rely on NeRF for multimodal sensor calibration [10], [11], [12]. This is done by exploiting the fully differentiable structure of the NeRF model to simultaneously learn a volumetric scene representation while regressing the correct calibration parameters to optimize the rendering quality. INF [10] proposes to first train the density of the scene with the LiDAR scans, before calibrating the extrinsic parameters of a 360° camera by learning the color information from the captured images. MOISST [11] performs spatial and temporal calibration of multiple LiDARs and cameras by building a trajectory with the poses of a reference sensor and training a NeRF with the information provided by all the sensors. SOAC [12], while requiring a camera as the reference sensor, further improves the robustness and accuracy by using multiple NeRFs and taking into account the overlap between the sensors over the sequence. While these methods can provide high accuracy and robustness, they require relatively long training times (1-2 hours [12]) even when exploiting more efficient hash-based encoding techniques [20].

**3D Gaussian Splatting.** 3D Gaussian splatting [13] is a novel volumetric rendering method that exploits the high efficiency of splatting 3D Gaussians on a 2D image to obtain real-time rendering of novel views. Unlike NeRF, 3DGS features an explicit representation composed of a high number of Gaussians each defined by its position, color, scale, and opacity. These parameters are trained with gradient descent by applying a loss function between the rendered image and its ground-truth counterpart provided by the camera to minimize the photometric error. Since it is based on an explicit representation, 3DGS requires an initial point cloud to define the Gaussians at the beginning of the training, which is commonly obtained through Structure-from-Motion. During training, the model is allowed to add new Gaussians to create more details or prune them if deemed useless for the representation.

As the rasterization of 3D Gaussian splatting is differentiable with respect to the positions of the Gaussians, some methods propose to propagate the gradients to the image poses to solve tasks like SLAM [25], [26] or image registration [27]. These methods use the depth information from RGB-D cameras [25], [26], or the predicted monocular depth [27] to create a point cloud for the initialization of the Gaussians. In these methods, during the pose optimization/localization step, the model minimizes the photometric cost function by optimizing the pose of each newly introduced image of the sequence with respect to the already existing parameterized Gaussians. The number of Gaussians and their parameters in these approaches are fixed to prevent the model from compensating the incorrect pose with a degenerate scene representation. Once a new image has been registered, it can be used with its obtained camera pose to train the representation and eventually add new Gaussians. This strategy can also be transposed to the calibration of multimodal systems which, as far as we know, has not yet been addressed. Thus, we have decided to tackle this subject in this work.

## III. METHOD

In this paper, we formulate the multi-modal calibration problem as follows. Given the poses of the LiDAR, which can be derived either through reliance on LiDAR-based SLAM [28] or simply via the Iterative Closest Point (ICP)

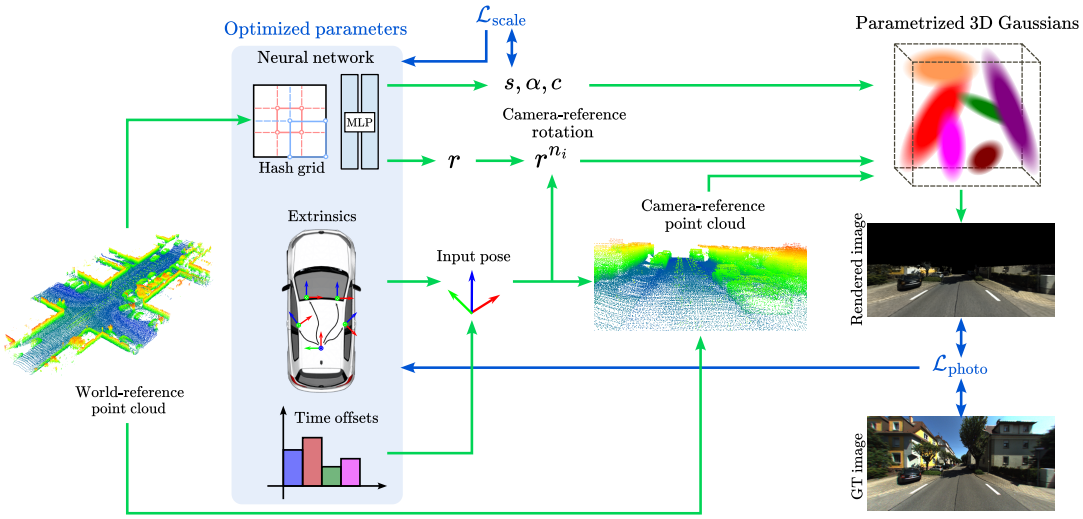


Fig. 1. **Pipeline of 3DGS-Calib:** The Gaussians' positions are given as input to the neural network which predicts their parameters. In parallel, the calibration parameters provide the input pose that transforms the Gaussians from the world frame to the image frame. Then, the 3D Gaussians are splatted using their predicted parameters to generate the rendered image. This image is compared to its ground-truth (GT) counterpart to compute the photometric loss. Finally, the gradients are backpropagated to the neural network and the calibration parameters.

TABLE I

NOTATIONS USED IN THIS PAPER.

| Notation | Meaning |
|---|---|
| $C$ | Set of cameras |
| $\{F_i\}_{i \in C}$ | Sets of frames captured by the camera $i \in C$ |
| $t^{n_i} \in \mathbb{R}^+$ | Timestamp of images $n_i \in F_i$ relative to the camera $i \in C$ |
| $\delta_i \in \mathbb{R}$ | Time offset between the LiDAR and the camera $i \in C$ |
| ${}_wT^i(t) \in \mathbb{R}^{4\times4}$ | The pose of camera $i \in C$ at time $t$ (the time is relative to sensor $i$'s own clock) in the world reference frame |
| ${}_wT^L(t) \in \mathbb{R}^{4\times4}$ | Pose of the LiDAR at time $t$ |
| ${}_LT^i \in \mathbb{R}^{4\times4}$ | Transformation matrix from camera $i$ to LiDAR |
| $X \in \mathbb{R}^{3\times G}$ | Accumulated LiDAR point positions in the world reference, with $G$ the number of points |

algorithm [29], [30], in addition to initial noisy/inaccurate priors of the spatial and temporal calibration parameters w.r.t. the LiDAR, our goal is to obtain the correct calibration parameters. In contrast to NeRF-based methods where one of the cameras is often considered as the reference sensor [11], our method uses the LiDAR as its reference sensor since we rely on its point cloud to initialize the 3D Gaussians' positions. Our goal is to obtain the correct spatiotemporal calibration of the cameras on the vehicle w.r.t. the LiDAR. The overall pipeline is described in Figure 1.

### A. Notations and Background

The notations used throughout this paper are summarized in Table I. Our objective is to determine the optimal spatial transformations ${}_L\hat{T}^i$ and time offsets $\hat{\delta}_i$ w.r.t. the LiDAR for the various sensors to achieve calibration. Drawing from similar methodologies employed in MOISST [11] and SOAC [12], we establish a continuous trajectory, denoted as $\mathcal{T}_L$, for the LiDAR from its discrete poses. Linear interpolation is used for pose translation, while spherical linear interpolation (SLERP [31]) is employed for rotation. This trajectory is represented as a time-dependent function, ${}_wT^L(t) = \mathcal{T}_L(t)$ which provides the LiDAR pose for any given time $t$. By leveraging the extrinsic transformations and time offsets between the cameras and the LiDAR, we are able to calculate the absolute pose of camera $i$ at particular timestamps using the following equation:

$$ {}_wT^i(t^{n_i} + \delta_i) = \mathcal{T}_L(t^{n_i} + \delta_i)\, {}_LT^i. \tag{1} $$

From now on, we designate the absolute pose of camera $i$ computed from its extrinsic as $T^{n_i} = {}_wT^i(t^{n_i} + \delta_i)$.

### B. Scene representation with 3D Gaussians

3DGS [13] models the scene using parameterized 3D Gaussians, which is an explicit representation, unlike NeRF [18]. The Gaussians are parameterized by their centers $\mu \in \mathbb{R}^{3\times G}$, their opacities $\alpha \in \mathbb{R}^G$, their rotation quaternions $r \in \mathbb{R}^{4\times G}$, their 3D scales $s \in \mathbb{R}^{3\times G}$ and their RGB colors $c \in \mathbb{R}^{3\times G}$. In our case, as we focus on calibration and not on rendering quality, we consider the colors as view-independent. This helps in avoiding degenerate cases where color changes from different viewing directions can compensate for noisy poses. As we have access to the LiDAR scans and their associated poses, it means that we can accumulate these scans to obtain a point cloud of the sequence. This point cloud $X$ is used to define the positions of the Gaussians. As these points are supposed to represent a real 3D point seen by the LiDAR, we do not optimize their positions, which we consider accurate from the beginning. Thus, $\mu$ parameters are not needed in our case as we replace them using $X$. While 3DGS learns the scene geometry (i.e. Gaussians' positions) from known camera poses, our key idea is to register the camera poses on the scene geometry from the LiDAR. In the same way as the 3DGS SLAM methods [25], [26], we also remove densification and pruning, to prevent the model from creating or removing Gaussians which would allow the model to overfit to each image with incorrect geometry. For each training step, by using the calibration parameters to determine the input pose, we transform the Gaussians to the

image frame:

$$X^{n_i} = (T^{n_i})^{-1} X, \tag{2}$$

with $X^{n_i}$ being the Gaussians' centers in the camera frame associated with image $n_i$. Similarly, the anisotropic Gaussians are rotated using:

$$r^{n_i} = (R^{n_i})^{-1} r, \tag{3}$$

with $R^{n_i}$ being the rotation matrix extracted from $T^{n_i}$ and $r^{n_i}$ the Gaussian rotations in the camera frame associated with image $n_i$. Then, the 3D Gaussians are splatted to render an image according to the intrinsic parameters of the camera. The same photometric loss $\mathcal{L}_{photo}$ function as the original 3DGS paper [13] is used:

$$\mathcal{L}_{\text{photo}} = (1 - \lambda_1)\mathcal{L}_1 + \lambda_1 \mathcal{L}_{\text{D-SSIM}}, \tag{4}$$

with $\lambda_1 = 0.2$ in our experiments. The goal is to find the optimal parameters such as:

$$\begin{aligned} &\left\{ {}_L\hat{T}^i, \hat{\delta}_i \right\}_{i \in C}, \hat{\alpha}, \hat{s}, \hat{r}, \hat{c} = \\ &\underset{\left\{ {}_L T^i, \delta_i, \alpha, s, r, c \right\}}{\text{argmin}} \left( \mathcal{L}_{\text{photo}}(\Phi(X^{n_i}, \alpha, s, r^{n_i}, c), n_i) \right) \end{aligned} \tag{5}$$

with $\Phi$ the rendering function from [32]. The gradients are then backpropagated to both the Gaussian parameters, and the calibration parameters through the input pose.

### C. Gaussian parameters with Neural Network

Learning independent Gaussians' parameters while modifying the pose of several sensors leads to a suboptimal training process. This is mainly because every change in the calibration parameters will require each Gaussian to adapt its information about the scene separately. To address this issue, we learn the Gaussians' parameters using a neural network that provides a regularized scene representation shared across sensors. We use the model from Instant-NGP [20], which takes as input the 3D position of a Gaussian. We add 4 heads at the output of the Multilayer Perceptron (MLP) to predict the color, opacity, scale, and rotation parameters of the Gaussian. This allows the spatially close Gaussians to have similar parameters, which is coherent with the nature of real-world scenes where the colors and shapes are continuous on large areas (ex: walls, roads). Thus, we define the function $F(X|\Theta)$ which takes as input the 3D positions of the Gaussian, and returns the Gaussian parameters according to the network parameters $\Theta$. We then redefine Eq. 5 as follows:

$$\begin{aligned} &\left\{ {}_L\hat{T}^i, \hat{\delta}_i \right\}_{i \in C}, \hat{\Theta} = \\ &\underset{\left\{ {}_L T^i, \delta_i, \Theta \right\}}{\text{argmin}} \left( \mathcal{L}_{\text{photo}}(\Phi(X^{n_i}, F(X|\Theta)), n_i) \right) \end{aligned} \tag{6}$$

As the Gaussians' positions are transformed to the camera frame, we remove the points behind the camera before inference, allowing a considerable time speed-up.

### D. Preprocessing and optimization

To improve both the performance and the training speed of our method, we adopt the following enhancing design decisions:

*1) Accumulated LiDAR downsampling:* When accumulating the LiDAR scans, as multiple passes are done in the same area over time, this results in a large number of redundant points, which not only does not improve the rendering quality but also slows down the training process, as each Gaussian requires an MLP inference. By using voxel-based downsampling, we can obtain a homogeneously dense point cloud at the desired resolution. This solution allows considerable speed-up of the calibration process. As we keep one point per voxel, we also enforce the scale of each Gaussian to be smaller than the voxel size. This helps the rendered result to be sharper, by preventing the Gaussians from overlapping each other.

*2) Coarse-to-fine voxelization:* Since the Gaussians' parameters are obtained through the neural network inference, the training time is proportional to the number of the used points. During the beginning of the calibration process, as we are still correcting large miscalibration errors, it is not necessary to have a high level of detail in the rendered images. Thus, we can use larger voxels for the downsampling. As the training progresses, we increase the resolution since we are getting more precise calibration. This allows the first part of the training to run much faster.

*3) Image cropping:* The Gaussians' positions in our method are exclusively determined by the LiDAR points which, in typical driving sensor setups, mostly cover the road and the lower/closer structures of the environment. The LiDAR points are unable to provide the structure of far elements and the sky. In light of this, we only consider the bottom half of the images when calculating the losses.

*4) Gaussians' scale regularization:* During the training, to avoid very thin Gaussians, which usually create artifacts in the representation, we apply a scale regularization loss similar to [26], which prevents a high scale difference along the 3 axes. This is achieved using the following regularization loss:

$$\mathcal{L}_{\text{scale}} = \sum_{g=1}^{G} \| s_g - \tilde{s}_g \|_1, \tag{7}$$

with $G$ being the total number of the Gaussians, and $\tilde{s}$ the mean of the scale along the 3 axis. Hence, our model's loss function becomes:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{photo}} + \lambda_2 \mathcal{L}_{scale}, \tag{8}$$

with $\lambda_2 = 0.001$ across all our tests.

## IV. EXPERIMENTS

### A. Dataset

For our evaluation, we use the KITTI-360 driving dataset [17], which contains images from 2 front cameras and 2 side fish-eye cameras in addition to point clouds from a 360° LiDAR. We build 3 sequences with different characteristics to evaluate our method in different conditions. The built sequences are described in Table II. We skip 1 out of 2 frames, which results in 40 frames for each sensor in every sequence.

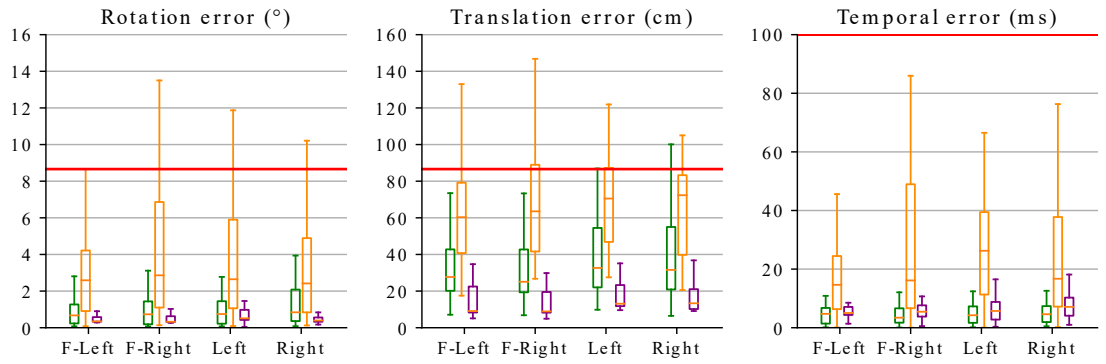


Fig. 2. Results for MOISST, MOISST /w cropping and 3DGS-Calib as box plots. The red lines show the initial error (Best viewed in color).

TABLE II

SELECTED FRAMES FOR EACH KITTI-360 SEQUENCE.

| Sequence | Run | Starting frame | Ending frame | Description |
|---|---|---|---|---|
| 1 | 0009 | 980 | 1058 | Straight line |
| 2 | 0009 | 2854 | 2932 | Large rotation |
| 3 | 0010 | 3390 | 3468 | Small zigzag |

### B. Training details

Our model is trained for a total number of 6000 steps. We initialize Gaussians' positions using a downsampled point cloud with a voxel size of 10 cm for 4000 steps. Then, we reduce the downsampling voxel size to 5 cm and train for 1000 steps. Finally, for the finest rendering details and calibration parameters, we use a voxel size of 2 cm for 1000 steps. To stabilize the training, we warm up for 500 steps before kicking off the calibration process. The colors are randomly initialized, and the scales, rotations and opacities are initialized as in the original 3DGS implementation.
As proposed in MOISST [11] and SOAC [12], a weight decay of $10^{-4}$ is applied on the hash grid to allow better convergence during the whole training. We use the Adam optimizer and apply a learning rate of $10^{-4}$ for rotation, $5 \times 10^{-3}$ for translation, and $10^{-3}$ for temporal correction. These learning rates are linearly decayed by 0.1 times the initial value at the end of the training.

When comparing with MOISST, we use the same parameters as the original paper, except that we limit the number of epochs to 10, as we observed that the model convergences. We also start the depth loss from the beginning of the training, as the LiDAR is assumed to have correct poses, and thus provide correct geometry. For fairness, we compare our method with MOISST using both the whole image and its bottom half as it is done in our method. The reasoning behind this is to provide both methods with the same prior and knowledge about the sensor setup. For the initial noise, we set an error of 5° and 50 cm on all 3 axes, which can be either positive or negative. In the same way, we add a 100 ms time offset on the sensors to simulate synchronization errors. All experiments are run with 10 random seeds before being averaged. We run all methods with the same GPU whose performance is equivalent to an NVIDIA RTX 3090.

### C. Spatiotemporal calibration

For spatiotemporal calibration, we compare our method to MOISST, as it can use the LiDAR as the reference sensor (in contrast to SOAC [12]), and provide both spatial and temporal calibration. In Figure 2, we provide the results of our method in boxplots[1] against MOISST trained with the whole image or only the bottom half of the images (w/ cropping). We also provide the training time per sequence in Table III. Our method provides higher accuracy than MOISST, while requiring a much lower training time. The fact that MOISST with cropping performs much worse than without shows that it heavily relies on the RGB image correspondences to regularize the calibration, thus removing the top half of the images eliminates a lot of information for MOISST to exploit. With our progressive voxel resolution, the training time is 10 times shorter than MOISST, and 5 times shorter than MOISST with cropping.

TABLE III

TRAINING TIME FOR EACH SEQUENCE IN SECONDS.

| Sequence | MOISST | MOISST w/ cropping | 3DGS-Calib |
|---|---|---|---|
| 1 | 5340 (×10.39) | 2707 (×5.25) | **514** |
| 2 | 5280 (×9.96) | 2700 (×5.09) | **530** |
| 3 | 5040 (×11.78) | 2695 (×6.31) | **428** |

(between parenthesis is the multiplication factor compared to our method.)

### D. LiDAR/Camera calibration

For the LiDAR/camera calibration task, we remove the temporal calibration component. We reduce the number of steps in our method to 3000 (2000 steps with 10 cm voxels, 500 with 5 cm, and 500 with 2 cm). In addition to MOISST, using the code provided by the authors, we add the comparison to a classical method based on mutual information proposed by Pandey et al. [7], and a deep learning-based method, LCCNet [9]. For LCCNet, We use the provided weights pre-trained on KITTI [33], a dataset similar to KITTI-360 [17], albeit with some differences in the setup. We evaluate this task using the front-left camera and the LiDAR of the sequences, and report the results in

[1]The boxes show the first quartile $Q_1$, median, and third quartile $Q_3$. The whiskers use 1.5 IQR (Interquartile range) above and below the box and stop at a value within the results.

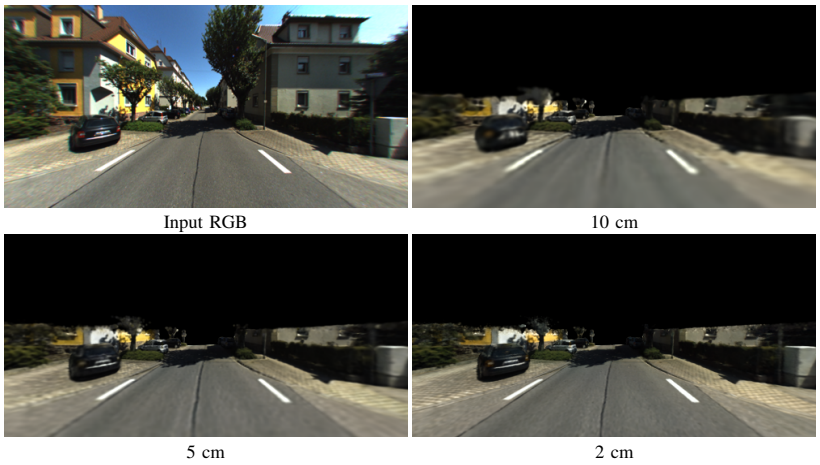


Fig. 3. **Rendering results with different voxel sizes:** The finer details of the scene require a smaller voxel size to be learned.

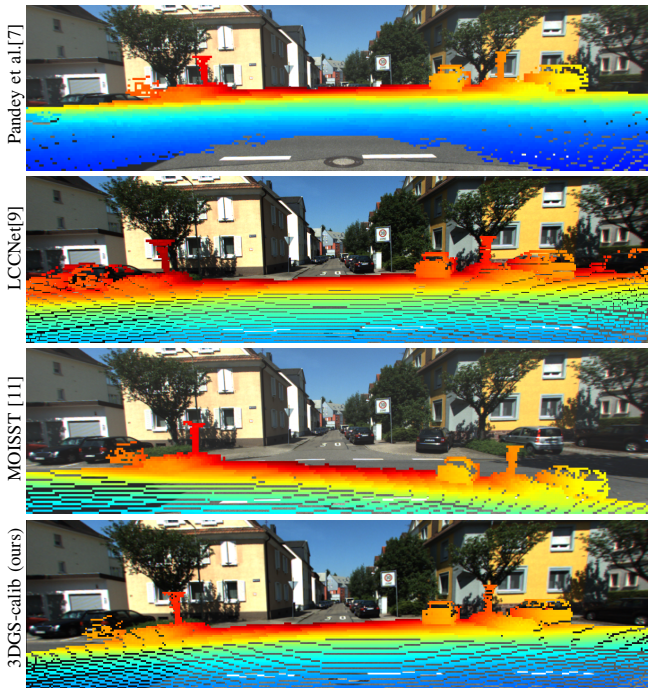


Fig. 4. **LiDAR/Camera calibration results:** Point cloud to camera reprojection results obtained from the compared methods.

TABLE IV

LIDAR/CAMERA CALIBRATION

| Method | Rotation (°) | Translation (cm) | Training time (s) |
|---|---|---|---|
| Pandey et al. [7] | $11.6 \pm 6.5$ | $90.7 \pm 29.5$ | - |
| LCCNet [9] | $1.86 \pm 0.09$ | $89.5 \pm 2.2$ | - |
| MOISST [11] | $4.92 \pm 0.5$ | $64.1 \pm 6.8$ | 1527 |
| MOISST w/ cropping | $6.01 \pm 1.2$ | $75.0 \pm 3.7$ | 860 |
| 3DGS-Calib (Ours) | $\mathbf{0.45 \pm 0.2}$ | $\mathbf{9.60 \pm 2.1}$ | **216** |

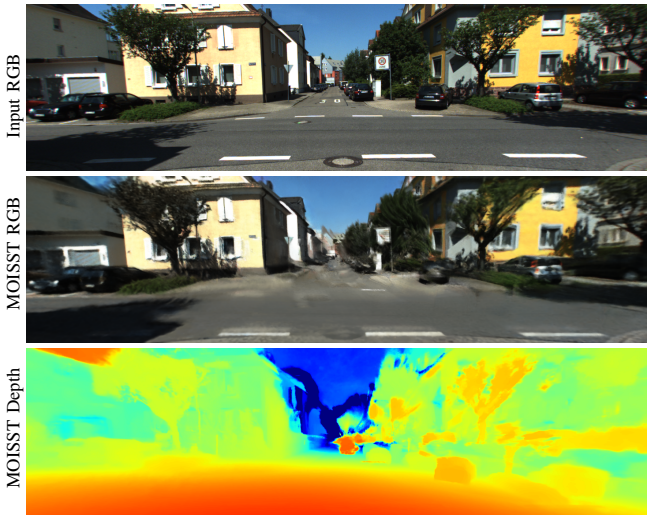


Fig. 5. **MOISST rendering with LiDAR/Camera calibration:** MOISST fails to merge the geometry of the trees and cars from the LiDAR with the geometry built with the images.

Table IV and Figure 4. As shown by [12], LCCNet weights are system-specific, explaining the results as it was pre-trained on KITTI, where the camera/LiDAR placement is different. Pandey et al. [7] is unable to provide satisfying results since in-the-wild sequences in urban areas have a lot of shadows or elements where there is no correlation between grayscale image and LiDAR reflectance. MOISST, when not using a camera as a reference sensor, seems unable to provide satisfying calibration, as it was not able to correctly merge the LiDAR geometry with the one built from the images (see Figure 5).

### *E. Ablation study*

We compile the calibration results and training times according to the voxel size, as well as the number of Gaussians for each resolution in Table VI and the number of points for each resolution in Table V. It can be noticed that a smaller voxel size allows to learn finer details (Cf. Figure 3), but requires much higher training time and more easily converges to local minimum. Our progressive voxel size reduction provides a balance between speed and performance. The rendering results for different voxel resolutions are provided in Figure 3.

TABLE V

NUMBER OF POINTS FOR EACH SEQUENCE AND VOXEL SIZE.

| Sequence | Original | 2cm | 5cm | 10cm |
|---|---|---|---|---|
| 1 | 4,068,707 | 3,289,412 | 1,295,133 | 394,154 |
| 2 | 4,025,853 | 3,351,941 | 1,431,667 | 484,690 |
| 3 | 4,220,417 | 2,948,428 | 903,399 | 245,549 |

TABLE VI

SPATIOTEMPORAL CALIBRATION ACCURACY AND TRAINING TIME FOR DIFFERENT VOXEL SIZES.

| Voxel size | Rotation (°) | Translation (cm) | Time offset (ms) | Training time (s) |
|---|---|---|---|---|
| 10 cm | **0.28** | 14.1 | 8.3 | **285**($\times$0.59) |
| 5 cm | **0.28** | <u>10.1</u> | <u>7.2</u> | 620($\times$1.26) |
| 2 cm | 0.38 | **9.0** | 7.9 | 1390($\times$2.85) |
| Ours | <u>0.31</u> | 10.3 | **6.7** | <u>490</u> |

(between parenthesis is the multiplication factor compared to our method.)

## V. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a novel targetless multimodal and multi-sensor calibration method that takes advantage of the speed of 3D Gaussian splatting to provide an accurate and robust calibration with shorter training time compared to NeRF-based methods. Our experimental evaluations on the KITTI-360 dataset demonstrate that we achieve better calibration results than both classical and recent NeRF-based methods while being orders of magnitude faster. In the future, we expect to remove the assumption of the LiDAR points being concentrated in the bottom half of the images, to allow better adaptability to uncommon types of LiDAR or unusual camera dispositions.

## REFERENCES


[1] A. Geiger, F. Moosmann, Ö. Car, and B. Schuster, "Automatic camera and range sensor calibration using a single shot," in *IEEE international conference on robotics and automation (RA-L)*, 2012, pp. 3936–3943.

[2] C. Guindel, J. Beltrán, D. Martín, and F. García, "Automatic extrinsic calibration for lidar-stereo vehicle sensor setups," in *IEEE international conference on intelligent transportation systems (ITSC)*, 2017, pp. 1–6.

[3] Z. Pusztai and L. Hajder, "Accurate calibration of LiDAR-camera systems using ordinary boxes," in *IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, 2017, pp. 394–402.

[4] A. Napier, P. Corke, and P. Newman, "Cross-calibration of push-broom 2d lidars and cameras in natural scenes," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 3679–3684.

[5] C. Yuan, X. Liu, X. Hong, and F. Zhang, "Pixel-level extrinsic self calibration of high resolution lidar and camera in targetless environments," *IEEE Robotics and Automation Letters (RA-L)*, vol. 6, no. 4, pp. 7517–7524, 2021.

[6] Z. Taylor and J. Nieto, "A mutual information approach to automatic calibration of camera and lidar in natural environments," in *Australian Conference on Robotics and Automation*, 2012, pp. 3–5.

[7] G. Pandey, J. McBride, S. Savarese, and R. Eustice, "Automatic targetless extrinsic calibration of a 3d lidar and camera by maximizing mutual information," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2012.

[8] G. Iyer, R. K. Ram, J. K. Murthy, and K. M. Krishna, "CalibNet: Geometrically supervised extrinsic calibration using 3d spatial transformer networks," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1110–1117.

[9] X. Lv, B. Wang, Z. Dou, D. Ye, and S. Wang, "LCCNet: LiDAR and camera self-calibration using cost volume network," in *CVPRW*, 2021, pp. 2894–2901.

[10] S. Zhou, S. Xie, R. Ishikawa, K. Sakurada, M. Onishi, and T. Oishi, "INF: Implicit Neural Fusion for LiDAR and Camera," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

[11] Q. Herau, N. Piasco, M. Bennehar, L. Roldão, D. Tsishkou, C. Migniot, P. Vasseur, and C. Demonceaux, "MOISST: Multimodal Optimization of Implicit Scene for SpatioTemporal calibration," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 1810–1817.

[12] Q. Herau, N. Piasco, M. Bennehar, L. Roldão, D. Tsishkou, C. Migniot, P. Vasseur, and C. Demonceaux, "SOAC: Spatio-Temporal Overlap-Aware Multi-Sensor Calibration using Neural Radiance Fields," *arXiv preprint arXiv:2311.15803*, 2023.

[13] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics (ToG)*, vol. 42, no. 4, 2023.

[14] A. Moreau, J. Song, H. Dhamo, R. Shaw, Y. Zhou, and E. Pérez-Pellitero, "Human gaussian splatting: Real-time rendering of animatable avatars," *arXiv:2311.17113 [cs.CV]*, 2023.

[15] H. Dahmani, M. Bennehar, N. Piasco, L. Roldal, and D. Tsishkou, "Swag: Splatting in the wild images with appearance-conditioned gaussians," *arXiv preprint arXiv*, 2024.

[16] X. Zhou, Z. Lin, X. Shan, Y. Wang, D. Sun, and M.-H. Yang, "Drivinggaussian: Composite gaussian splatting for surrounding dynamic autonomous driving scenes," *arXiv preprint arXiv:2312.07920*, 2023.

[17] Y. Liao, J. Xie, and A. Geiger, "KITTI-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d," *IEEE TPAMI*, vol. 45, no. 3, pp. 3292–3310, 2022.

[18] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021.

[19] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, "Mip-NeRF 360: Unbounded anti-aliased neural radiance fields," in *CVPR*, 2022, pp. 5470–5479.

[20] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM TOG*, vol. 41, no. 4, pp. 1–15, 2022.

[21] A. Moreau, N. Piasco, M. Bennehar, D. Tsishkou, B. Stanciulescu, and A. de La Fortelle, "Crossfire: Camera relocalization on self-supervised features from an implicit representation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

[22] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, and W. Wang, "Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction," *NeurIPS*, 2021.

[23] Z. Wang, S. Wu, W. Xie, M. Chen, and V. A. Prisacariu, "Nerf–: Neural radiance fields without known camera parameters," *arXiv preprint arXiv:2102.07064*, 2021.

[24] C.-H. Lin, W.-C. Ma, A. Torralba, and S. Lucey, "Barf: Bundle-adjusting neural radiance fields," in *ICCV*, 2021.

[25] C. Yan, D. Qu, D. Wang, D. Xu, Z. Wang, B. Zhao, and X. Li, "Gs-slam: Dense visual slam with 3d gaussian splatting," *arXiv preprint arXiv:2311.11700*, 2023.

[26] H. Matsuki, R. Murai, P. H. Kelly, and A. J. Davison, "Gaussian splatting slam," *arXiv preprint arXiv:2312.06741*, 2023.

[27] Y. Fu, S. Liu, A. Kulkarni, J. Kautz, A. A. Efros, and X. Wang, "Colmap-free 3d gaussian splatting," *arXiv preprint arXiv:2312.07504*, 2023.

[28] T. Shan and B. Englot, "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 4758–4765.

[29] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-d point sets," *IEEE Transactions on pattern analysis and machine intelligence*, no. 5, pp. 698–700, 1987.

[30] I. Vizzo, T. Guadagnino, B. Mersch, L. Wiesmann, J. Behley, and C. Stachniss, "Kiss-icp: In defense of point-to-point icp–simple, accurate, and robust registration if done the right way," *IEEE Robotics and Automation Letters (RA-L)*, vol. 8, no. 2, pp. 1029–1036, 2023.

[31] K. Shoemake, "Animating rotation with quaternion curves," in *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, 1985, pp. 245–254.

[32] M. Zwicker, H. Pfister, J. Van Baar, and M. Gross, "Ewa volume splatting," in *Proceedings Visualization, 2001. VIS'01.* IEEE, 2001, pp. 29–538.

[33] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.