Supplementary Document for: BAD SLAM: Bundle Adjusted Direct RGB-D SLAM

Thomas Schöps[1] Torsten Sattler[2] Marc Pollefeys[1,3]
[1]Department of Computer Science, ETH Zürich
[2]Department of Electrical Engineering, Chalmers University of Technology [3]Microsoft, Zurich

In this supplementary document, we first present an additional technical detail of our algorithm, and additional evaluation results, which do not fit into the main paper for space reasons. We then describe how we created our new RGB-D SLAM benchmark, and the lessons learned while doing so. This may be helpful for other researchers who plan to record similar datasets or who want to use the raw data, that we also plan to publish, to improve upon our calibration. We also provide a supplementary video which shows our SLAM system running on several datasets, as well as an overview of all datasets of our new benchmark.

1. Additional details

Here, we detail the computation of the surfel border points $\mathbf{s}_1$ and $\mathbf{s}_2$, which are used for computing the photometric descriptor, based on a surfel's position $\mathbf{p}_s$, normal $\mathbf{n}_s$, and radius r_s. First, taking the cross product of $\mathbf{n}_s$ in global space with an arbitrary fixed vector $\mathbf{v}$ (*e.g.*, $(1,0,0)^T$), and scaling it to the surfel's radius r_s gives: $\mathbf{s}_1 = \mathbf{p}_s + r_s\|\mathbf{n}_s \times \mathbf{v}\|_2^{-1}(\mathbf{n}_s \times \mathbf{v})$. If $\mathbf{n}_s$ and $\mathbf{v}$ are nearly parallel, a second arbitrary and fixed vector (*e.g.*, $(0,1,0)^T$) is used instead. $\mathbf{s}_2$ is then obtained from $\mathbf{s}_1$ as $\mathbf{s}_2 = \mathbf{p}_s + (\mathbf{n}_s \times (\mathbf{s}_1 - \mathbf{p}_s))$. This choice of $\mathbf{s}_1$ and $\mathbf{s}_2$ makes the point locations on the surfel, $\mathbf{s}_1 - \mathbf{p}_s$ and $\mathbf{s}_2 - \mathbf{p}_s$, invariant to surfel position changes, while normal changes will almost always cause smooth changes.

2. Additional SLAM results

2.1. Quantitative results

In the paper, we use the Absolute Trajectory Error (ATE) RMSE for all evaluations. In this supplementary document, we additionally present results on our new benchmark for relative translation and rotation evaluation metrics [5]. Both metrics are evaluated over all subsequences for a given length within a dataset. We use lengths of 0.5, 1.0, 1.5, and 2.0 meters. For the translational error, the translation between the first and last pose in a subsequence is compared between the ground truth and estimated trajectory. The error is computed as the ratio between the translation difference to the subsequence length, and given in percent. For the rotational error, analogously the difference in rotation is computed and given in degrees per meters of subsequence length. The results are plotted in Fig. 1 (analogously to Fig. 6 in the paper). It can be seen that the rankings generally do not depend on the choice of metric, and our method is best regardless of which metric is used.

2.2. Qualitative results

In Fig. 10 and Fig. 11 (at the end of this document), we show additional example reconstructions of datasets from our benchmark (similar to Fig. 1 in the paper), which did not fit into the paper. These qualitative results give an impression of the variety of our benchmark datasets. Furthermore, they illustrate the reconstruction quality achieved by BAD SLAM. As in the paper, all sequences are processed with a surfel sparsity setting of one surfel per 4×4 pixels. If denser reconstructions are desired, this can be changed to obtain denser reconstructions than shown here. Since texture reconstruction was not the focus of our work, we neither apply white-balancing nor homogenize colors of different images observing the same point, apart from averaging.

2.3. Performance

In the ablation study in the paper (Fig. 5, top), a PCG-based Gauss-Newton solver is included. Analogously to Fig. 4 in the paper, we show a performance graph for it in Fig. 4. It uses 10 inner iterations for the PCG step. Clearly, individual iterations take significantly longer than with the alternating optimization scheme presented in the paper. While the quality of the resulting trajectories under real-time conditions is shown to be similar for our benchmark dataset in the paper, bundle adjustment iterations (and thus trajectory corrections) become infrequent for longer datasets with the PCG-based Gauss-Newton solver.

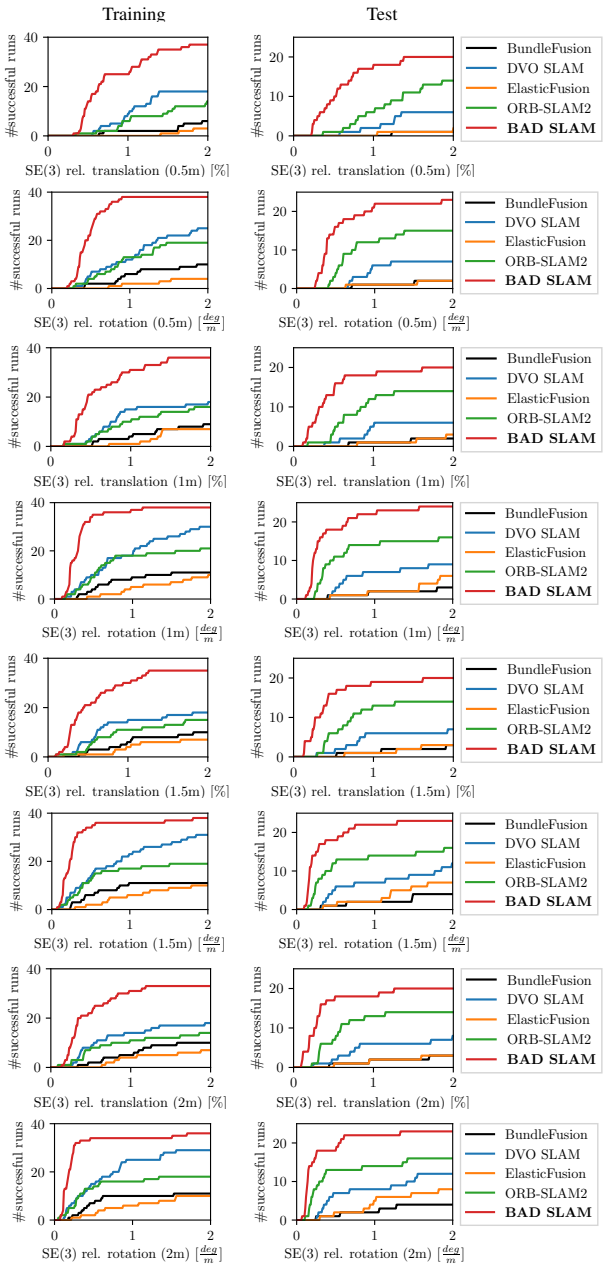

Figure 1. Evaluation on our benchmark's training and test datasets. For a given threshold on the error metric (x-axis), the graphs show the number of datasets for which the method has a smaller error.

3. Dataset recording

We used a custom RGB-D sensor, built using a multi-camera system, to record RGB-D SLAM benchmark datasets. Ground truth was recorded with a Vicon system, except for a few training datasets for which it was obtained using Structure-from-Motion, as described in Sec. 5 of our paper. In this section, we describe the devices we used and the recording process for our datasets.

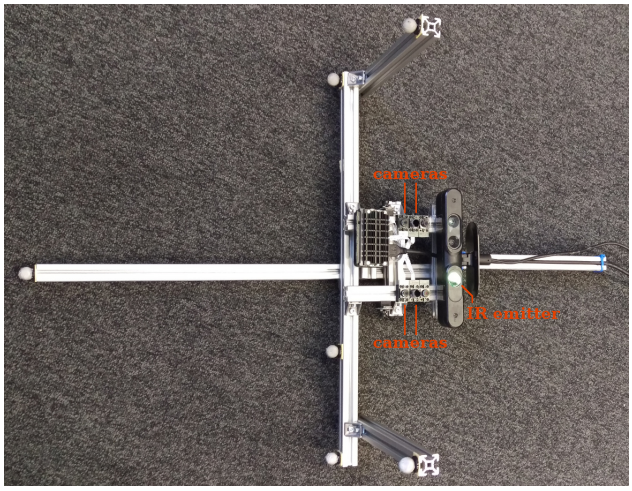

Figure 2. Camera rig used to record the benchmark datasets. The Vicon markers were placed far apart with the aim of increasing the orientation tracking accuracy. Out of all cameras on the rig, only the four labeled ones, which are two color and two infrared cameras, are directly used for the benchmark datasets.

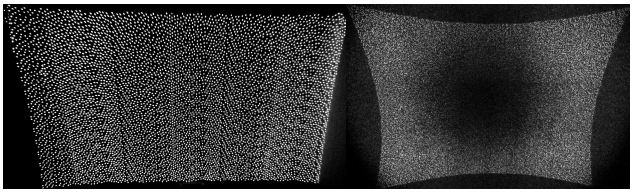

Figure 3. **Left:** Intel D435 infrared pattern. **Right:** Asus Xtion infrared pattern. Please note: the lack of dots in the center is only an artefact caused by the external camera used to record these photos to show the complete extent of the patterns, and does not appear in pictures recorded by our camera rig. Both images are contrast-enhanced.

3.1. Devices

Fig. 2 shows a photo of our camera rig. We used a camera system similar to the one described in [6] with up to eight synchronized cameras. We use four cameras for the datasets recorded in the Vicon system, and eight cameras for the datasets with Structure-from-Motion-based ground truth. As an infrared pattern emitter, we use an Asus Xtion Live Pro which is additionally mounted on the camera rig. We also considered using the emitter of the recent Intel D435 camera, but decided for the Xtion since its pattern is of higher resolution, as shown in Fig. 3.

Out of the eight synchronized cameras, two color cameras are arranged into a front-facing stereo camera pair. One camera of this pair provides the RGB component of the RGB-D camera, and optionally both cameras can be used for stereo SLAM. Two infrared cameras are arranged into a second stereo pair directly below the color cameras. These cameras observe the pattern projected by the Xtion. Their images are used for stereo matching to compute the depth component of the RGB-D camera. The remaining four syn-

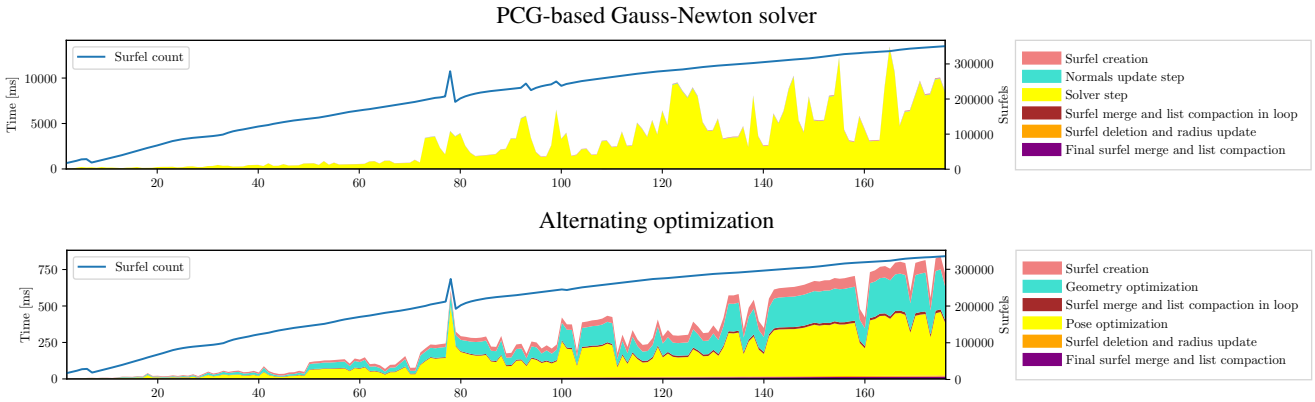

Figure 4. **Top:** Runtime of our bundle adjustment scheme using a PCG-based Gauss-Newton solver in milliseconds for the dataset shown in Fig. 1 of the paper (without skipping any BA iterations). **Bottom:** Corresponding plot of our alternating optimization scheme for comparison (replicating Fig. 4 of the paper). The number of keyframes is shown on the x-axis. Since we create one keyframe every 10 frames for ca. 27 Hz input, 370 ms of processing time are available for each keyframe; if BA takes longer, iterations are skipped in real-time mode. The spike in the surfel count corresponds to a loop closure. For the PCG-based solver, the runtime of all other steps becomes negligible in comparison. At slightly more than 70 keyframes, the increase in runtime is caused by the BA scheme doing more work per keyframe: It does up to the maximum of ten iterations per keyframe then, while it often converges after one or two iterations before. This is because the trajectory returns to an already visited place then, such that more adjustment is done compared to the pure exploration before.

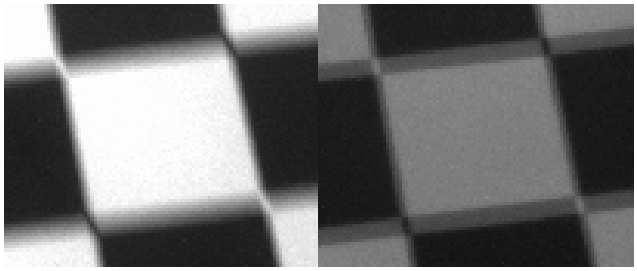

Figure 5. **Left:** Visible-light image of a checkerboard square with motion blur. **Right:** Infrared image recorded with the same exposure interval as the left image while the flashing infrared illumination of the Vicon system was active. Instead of continuous motion blur, two separate flashes are visible that happened during the exposure interval.

chronized cameras were only used for better localization of the training datasets for which ground truth was obtained using Structure-from-Motion, *c.f.* Sec. 5 of our paper.

We covered the camera rig with several passive marker balls such that its pose can be tracked by a Vicon motion capturing system (abbreviated as mocap system in the following). We tried to arrange the markers in a configuration with as little symmetry as possible to avoid the danger of the mocap system wrongly associating the markers. We also placed them such that all axes of rotation should be well observable and tried to keep occlusion to a minimum.

3.2. Calibration dataset recording

For creating our SLAM benchmark datasets, it was required to both calibrate the camera intrinsics and relative transformations within the camera rig, as well as to calibrate the time offset and relative transformation between the camera rig and the mocap system. In principle, it would be convenient to use a given calibration video sequence for both of these tasks at the same time. This would entail that the Vicon system must run during this sequence and the camera must be in motion to be able to perform the camera-Vicon calibration. Unfortunately, the flashing infrared light used by the mocap system illuminates the scene only during certain points in time. Under this type of illumination, the infrared camera images will not show a continuous exposure but instead snapshots of the scene at the times the infrared light flashed (see Fig. 5). In our experience, this is an issue for camera calibration: The features detected in these infrared images will likely not represent the projection of the feature at the middle of the exposure time, but at some other point in time during the exposure. Therefore, we recorded separate datasets for camera calibration and for camera-mocap synchronization. For the camera calibration datasets, we turned off the Vicon system and instead used sunlight to get a continuous infrared illumination to calibrate the infrared cameras. The infrared projector of the Xtion was also disabled for these datasets[1]. For the camera-mocap synchronization, where the infrared light of the Vicon system is needed, we simply did not use any of the infrared cameras.

3.3. Benchmark dataset recording

Directly before and after each recorded dataset sequence, we recorded a camera-mocap synchronization sequence. Afterwards, we additionally recorded a short camera calibration sequence for which we moved the camera very slowly. Having synchronization sequences before and af-

[1]Differing from the depth estimation in the Xtion camera itself, the Xtion's infrared projector only plays the role of an external light in our active stereo setup and is thus irrelevant for its calibration.

ter the dataset recording enables noticing when something goes wrong during the dataset recording, *e.g.*, the cameras in the rig move relative to each other. Since we aimed for a high-quality dataset, we tried to avoid even small perturbations in the calibration. Thus we used the final short camera calibration sequences to refine the relative transformations between cameras on the rig individually for each dataset. For these sequences, we moved the camera very slowly to (mostly) avoid the problem with flashing infrared lights described in the paragraph above, and discarded images with too fast estimated camera movement. Alternatively, we could have turned off the Vicon system each time such a sequence was recorded, or used a tripod to record static images. The former would have required an external infrared light that is well-suited for calibration, *i.e.*, as homogeneous as possible, which was not available unfortunately, aside from sunlight during certain times of the day. The latter alternative might increase the danger of changing the relative camera transformations due to the physical shock of putting the tripod with the camera on the floor.

4. Dataset processing pipeline

4.1. Image preprocessing

The camera system we used applies little processing to the images such that certain post-processing steps, which are often done internally by cameras, had to be done in software. First, we remove invalid image data at the image borders, and perform a standard flat-field correction step on the images to remove fixed pattern noise. For the latter, we recorded additional calibration datasets: Black recordings, for which cameras are supposed to observe no light, were done by covering the cameras with a cloth, which covers the cameras well even in bright surroundings. Flat-field recordings, for which cameras should receive homogeneous illumination, were done by recording an approximately homogeneously white wall.

All black images are averaged to obtain a black current image B. From each homogeneously-illuminated image, we first subtract the black current image B. For each pixel in the image, we then average the pixel values in a region around this pixel and divide this value by the pixel's intensity to obtain a correction factor indicating the factor that must be applied to the pixel's value to (locally) obtain a homogeneous image. We chose to use a local region instead of the whole image for this in order to account for the images not being perfectly homogeneous. As a tradeoff, the resulting calibration however does not entirely remove vignetting that may be present in the images. For each pixel, we average all computed correction factors using the geometric mean (which should be used instead of the arithmetic mean to average factors), which yields a gain image G. Using the resulting calibration, images I can be corrected by first subtracting the black current image from them and then applying the correction factors to each pixel: $G \cdot (I - B)$. Qualitatively, we noticed that overall this model seemed to significantly improve the image quality, while however also creating a few outliers since the model could not explain the behavior of all pixels.

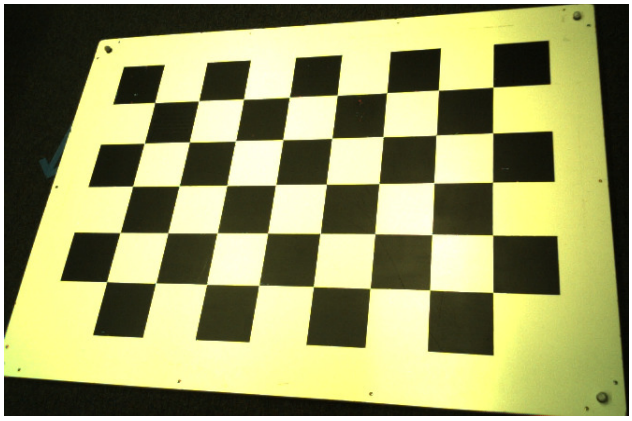

Figure 6. The checkerboard used for calibration, with mocap markers in three corners, as recorded by one of the color cameras. The checkerboard pattern is equally visible in the infrared images.

After flat-field correction, we replace dead pixels (which are masked out for flat-field correction) with interpolated neighbor values. Finally, for RGB images, we apply Debayering of the raw Bayer-pattern images using the Adaptive Homogeneity-Directed algorithm [8] since it gives good results (*c.f.* for example the survey [7]). We do not use any white-balancing on the images since this would likely make the SLAM algorithms' task harder.

4.2. Calibration pattern tracking

We used a calibration pattern for camera calibration (as opposed to using Structure-from-Motion), since a pattern allows for accurate and outlier-free feature localization. We chose to use a checkerboard pattern due to the availability of a large wooden checkerboard in our lab (shown in Fig. 6). Due to symmetry, the poses of partial views of a checkerboard may be ambiguous (and depending on the checkerboard's resolution, even complete views may be ambiguous). At the same time, using partial views is very helpful for getting sufficient calibration data close to image borders. Therefore, it was required to track the checkerboard over time in the calibration videos (instead of only detecting it in each frame individually) to resolve the ambiguities.

We tracked the checkerboard in each image by first using a heuristical coarse corner detection and tracking stage, as well as a corner refinement stage which refines each detected checkerboard corner location individually.

For corner refinement, we used an approach based on symmetry: For a pixel close to a checkerboard corner, mirroring the pixel's location across the corner location is sup-

posed to yield a pixel with the same checkerboard color (since camera distortions should be negligible locally). We thus define a cost function which measures how well this constraint is fulfilled for all pixels within a local window around a corner location hypothesis. To increase the robustness against non-uniform lighting, we do not compare pixel colors directly but compare intensity gradient magnitudes. This yields the following cost function, depending on the corner hypothesis location $\mathbf{q}$:

$$C(\mathbf{q}) = \sum_{\mathbf{p} \in W(\mathbf{q})} \left(||\nabla I(\mathbf{q}+\mathbf{p})||_2 - ||\nabla I(\mathbf{q}-\mathbf{p})||_2 \right)^2 \ . \quad (1)$$

The intensity gradient image is denoted ∇I (computed by centered finite differences), and the set of vectors from the local window's center to each other pixel within the window is denoted $W(\mathbf{q})$. We used a square 17×17 pixel window to process our datasets. We optimized the position of each corner hypothesis by using the Gauss-Newton algorithm on this cost function. While this produces very good results, we noticed that it can still break if specular reflections are visible in the calibration pattern. We tried to avoid these during recording.

4.3. Camera rig calibration

For camera intrinsics and rig calibration we obtained an initial estimate with Kalibr [9]. Kalibr employs only checkerboard detection using OpenCV [3], but not tracking. We therefore refined the calibration afterwards by optimizing the reprojection error of the checkerboard corners that we tracked as described in the previous section.

In this optimization, we found that it was very helpful to optimize not only the camera intrinsics and rig geometry, but also the 3D locations of the checkerboard corner points, analogously to performing Structure-from-Motion. This can correct potential inaccuracies in the pattern geometry and print. It strongly improved the camera-mocap synchronization trajectory RMSE in a later calibration step (*e.g.*, in one instance the error was reduced from $3.3mm$ to $1.0mm$).

Another aspect of an accurate calibration is to use a suitable camera model to represent the camera intrinsic calibration. We first experimented with the camera model used by the ETH3D benchmark [10], but found that it did not fit well to the radial distortion at the image corners. While the error may be small when measured in pixels of reprojection error, it can cause large distortions in depth images computed from these images. The initial ETH3D camera model used four coefficients for radial distortion modeling with a polynomial of up to 8th degree. Reducing the number of radial distortion coefficients worsened the results, suggesting that the issue was not numerical instability, but inability of the model to represent the true distortion. We therefore employed a 'non-parametric' camera model which directly maps from normalized image coordinates (*i.e.*, viewing directions pinhole-projected to the plane at $z = 1$) to pixel coordinates by bicubic interpolation in a dense vector-valued grid (with approximately one grid point each 25 pixels in the image). To optimize this camera model within a reasonable runtime, we wrote a custom Levenberg-Marquardt optimizer. It is essential to make use of the sparsity pattern of the residuals, which may be hard to model within optimization frameworks since it potentially changes during optimization (because a projected point may move to different control points for the bicubic interpolation). Our optimization also makes use of the Schur complement for matrix solving while delegating the largest matrix multiplication involved in its computation to the GPU.

4.4. Trajectory processing

The raw trajectories recorded by the motion capturing system contain (weak) noise and outliers. To address the outliers, we manually inspect the trajectory and delete bad poses. To address the noise, we apply a smoothing step to the trajectory: We fit an SE(3)-valued cubic B-spline [11] to it and re-sample the poses from the spline. Smoothness is controlled by the number of spline knots per second; we used 20 to perform only little smoothing.

4.5. Camera-mocap synchronization

Since the cameras and the mocap system use different clocks, their timestamps need to be aligned. Furthermore, the transformation between the poses returned by the mocap system and one of the cameras' projection centers must be determined, which is a standard hand-eye calibration problem. We optimize a cost function with a pose-based residual for each camera rig pose to determine both unknowns. We denote the global, fixed coordinate frame of the mocap system as G, the coordinate frame of the calibration pattern as P, the coordinate frame of the camera rig[2] which is tracked by the mocap system as M, and the coordinate frame of an arbitrarily chosen reference camera as C. A transformation from a coordinate frame A to frame B is denoted T_A^B. We interchangeably use the term pose for T_A^B and T_B^A since $T_A^B = {T_B^A}^{-1}$.

We estimate transformations $T_C^P(t)$ for each image timestamp t by tracking the pattern in the images recorded at this point in time (*c.f.* Sec. 4.2). The images can then be localized using the previously determined camera calibration based on the detected checkerboard corners. We also optimize for a scaling factor S_C^P which relates the scale of the camera poses (which depends on the estimated checkerboard geometry) and the scale of the mocap poses (which depends on the mocap system's calibration, which involves a calibration object of known size). $\text{scale}(S_C^P, T_C^P)$ denotes

[2]The mocap system tracks markers placed on the rig and not the rig cameras themselves.

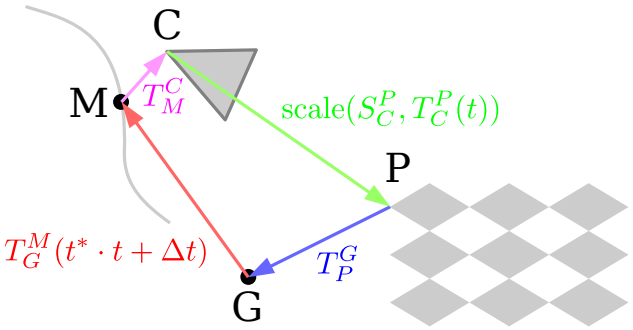

Figure 7. Sketch of the transformations involved in hand-eye calibration and time alignment.

scaling the transformation's translation component with the scaling factor. The transformation between the motion capturing system's origin G and the pose tracked by it, M, is parametrized by the estimated time alignment. It consists of a factor t^* and an offset Δt such that timestamps t of the camera can be converted to mocap timestamps as $t' = t^* \cdot t + \Delta t$. In order to obtain a reasonable parameter scaling for the optimization, we make sure that the origin of the multiplication is in the center of the trajectory's duration by shifting the timestamps accordingly as a pre-processing step. The transformation $T_G^M(t')$ for a residual is then determined by interpolating the mocap trajectory at time t', *i.e.*, the converted timestamp of the rig's image acquisition. Putting the pieces together, the optimization cost is defined as follows, where the colors match those in Fig. 7:

$$\sum_{t \in \Omega} \rho_{\text{Huber}}(\|\log(T_M^C \cdot T_G^M(t^* \cdot t + \Delta t) \cdot T_P^G \cdot \text{scale}(S_C^P, T_C^P(t)))\|_2) \ . \quad (2)$$

Here, Ω is the set of timestamps for which images were recorded by the camera rig, and a corresponding mocap pose has also been recorded. We solve the resulting optimization problem using the Ceres [1] optimization framework. Initial values for the optimization are obtained as follows.

We roughly estimate the initial time alignment as the transformation which brings the first and last timestamps of the camera and mocap poses within the dataset into alignment. For determining T_M^C, we mark the mocap markers, which are fixed on the checkerboard (*c.f.* Fig. 6), in a few images. The camera pose C in the G coordinate frame (T_G^C) for these images can then be determined from the camera calibration and the positions of the marked markers[3]. Using the initial time alignment, we also obtain rough poses of the M frame (T_G^M) by interpolating the mocap trajectory at the timestamps of the images. The initial value of T_M^C can then be determined as the difference between the M and C poses via $T_M^C = T_G^C \cdot {T_G^M}^{-1}$.

Next, we determine an initial estimate for the scaling factor S_C^P by randomly sampling timestamp pairs from within the trajectory and comparing the distance between the two T_G^C poses (*i.e.*, mocap poses concatenated with T_M^C) at these timestamps to the distance between the two T_P^C (localized camera) poses.

Finally, we initialize T_P^G, the pattern's pose within the mocap coordinate frame, by concatenating transforms (for any image): $T_P^G = {T_G^M}^{-1} \cdot T_C^M \cdot T_P^C$. We estimate all initial pose values and the scaling within RANSAC [4] processes to achieve robustness against outliers.

It should be noted that calibrating T_M^C with this approach requires to rotate the camera during the calibration sequence. Otherwise, translational changes to T_M^C would not be distinguishable from translational changes to T_P^G.

Timestamps for the Vicon poses are derived from Vicon frame numbers, assuming a constant frame rate. In an experiment, we optimized a separate time offset for each rig image set, instead of performing the alignment with $\Delta t, t^*$ described above, to determine whether the proposed time alignment is appropriate. The resulting offsets were fairly constant without showing any signs of systematic changes during the dataset, suggesting that the approach is suitable. Furthermore, we always run the synchronization on the combined synchronization dataset parts from before and after the dataset content recording. If anything with the timing goes wrong during the dataset it would thus likely become noticeable since it would likely lead to a bad synchronization result, leaving high residuals.

4.6. Depth estimation

In order to provide aligned RGB-D data, we estimate depth maps from the infrared camera pair and transform them to match the perspective of one of the color cameras. For depth estimation, we use a variant of standard PatchMatch Stereo [2] with a Zero-mean Normalized Cross Correlation (ZNCC) matching cost and a matching window of 11×11 pixels.

Despite the use of continuous depth estimates for each pixel, we noticed that the results showed a significant preference towards quantized depth values. We believe that this might arise from image intensity peaks, such as the maxima of the speckles produced by the infrared emitter. When bilinearly interpolating within an image, peak values of the interpolation result are always at pixel centers. If the peak in the stereo image is lower than the peak in the reference image, it is thus best aligned with an integer pixel disparity that exactly aligns the reference pixel center with the stereo pixel center. We mitigated this to some extent by modifying the sampling of the stereo matching window: In local

[3] This is assuming that the camera in which the markers were marked is chosen as the coordinate frame C. Otherwise, the rig geometry known from prior calibration as well as an estimate of the scaling between the mocap measurements and the rig geometry must be taken into account to get from the camera's coordinate frame to frame C.

Figure 8. Example frame with left infrared image (top left), right infrared image (top right), right RGB image (bottom left), and estimated depth image matching the RGB image (bottom right).

stereo methods, usually pixel centers within a small window around a pixel in the reference image are used for computing the stereo matching metric for this pixel. Instead, we randomly sample subpixel locations within the window to get a more continuous distribution of samples and thus reduce the tendency towards integer disparity alignment of the window as a whole.

The rest of our stereo pipeline follows standard practice, *c.f.* the post-processing in [12]. We refrain from using regularization since we expect getting a good data term from the active infrared illumination in most cases, and want to avoid making uncertain guesses in areas of homogeneous image content. We post-process the depth image with a median filter to remove some outliers. We then perform outlier filtering by left/right consistency checking, discarding pixels for which the normal points too far away from the camera, and discarding small connected components (where pixels can only count as connected if they have similar depth) in the depth image. We also discard pixels for which the co-visible range of the two cameras is small (such that the probability of the observed surface being within this range is small). Afterwards, we smooth the depth image with a bilateral filter and fill small holes.

Finally, we transform the depth image to the viewpoint of the RGB camera out of the color camera stereo pair which is closer to the infrared camera used as the stereo reference image. We create a triangle mesh from the depth map and render it in the color camera's viewpoint to perform the reprojection. This performs correct subpixel handling and avoids any hole artifacts which pixel-wise forward warping would create. Fig. 8 shows an example of the stereo inputs and resulting depth map together with the corresponding color image.

Since the source and destination camera for the reprojection are close together, only very few occlusion artifacts should be created in this process, which is a common processing step for RGB-D cameras. For example, the Asus Xtion uses this step as well, while the involved cameras are farther apart than the ones in our camera rig.

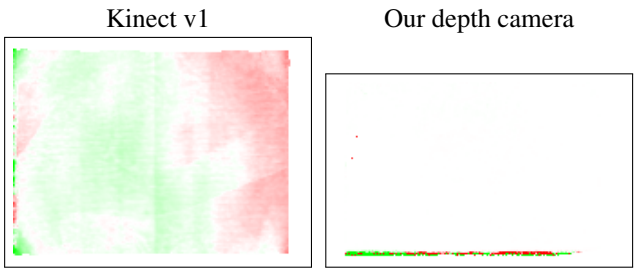

Figure 9. Visualizations of the depth deformation calibrated by BAD SLAM for the TUM RGB-D benchmark dataset long_office_household (left) and for a dataset from our benchmark (right). Saturated red (resp. green) means that the depth is distorted by 1 cm at 1 m depth towards (resp. away from) the camera. White indicates no distortion. While a distortion pattern is present for the Kinect v1 used by the TUM RGB-D benchmark, there is no visible distortion for our custom camera. The horizontal line arises from self-calibration outliers in an area where very few depth measurements are due to little stereo camera overlap. The same occurs for the Kinect on the left image border.

4.7. Calibration verification

To make sure that all components of the system are well-calibrated, we imposed thresholds on several calibration residuals at different steps of the processing pipeline. The choice of thresholds is based on examining the calibration residuals for our datasets. We choose the thresholds to prevent avoidable inaccuracies.

We require the initial camera calibration reprojection error to be below 0.07 pixels on average. The reprojection error during rig geometry refinement for a dataset must be below 0.1 pixels on average. We allow for a slightly higher error here since the infrared lighting is not continuous, *c.f.* Sec. 3.2 and Sec. 3.3. The reprojection error for the localization step in the camera-Vicon calibration must be below 0.085 pixels on average. This threshold is chosen between the first two since this reprojection error depends on both previous steps, while the lighting is continuous here since only the color cameras are used. In addition, we compute the Absolute Trajectory Error (ATE) RMSE, as defined in the paper, between the mocap trajectory and the final localized camera poses within the camera-Vicon calibration step, which we require to be at most one millimeter. This error is influenced by both the camera calibration and the camera-Vicon calibration. We rejected datasets which do not meet all of the quality criteria.

We also verified the camera calibration quality with BAD SLAM's optional depth deformation self-calibration step (*c.f.* Sec. 3.3, paragraph "Camera intrinsics optimization", in the paper). Fig. 9 visualizes example results of this cali-

bration for the Kinect v1 and for our depth camera. While a clear distortion pattern is visible for the Kinect, no distortion is visible for our camera, verifying that our calibration is good.

References

[1] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. http://ceres-solver.org. 6

[2] Michael Bleyer, Christoph Rhemann, and Carsten Rother. PatchMatch stereo - stereo matching with slanted support windows. In *BMVC*, 2011. 6

[3] Gary Bradski and Adrian Kaehler. OpenCV. *Dr. Dobbs journal of software tools*, 3, 2000. 5

[4] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 6

[5] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The KITTI dataset. *IJRR*, 32(11):1231–1237, 2013. 1

[6] Pascal Gohl, Dominik Honegger, Sammy Omari, Markus Achtelik, Marc Pollefeys, and Roland Siegwart. Omnidirectional visual obstacle detection using embedded FPGA. In *IROS*, 2015. 2

[7] Bahadir K. Gunturk, John Glotzbach, Yucel Altunbasak, Ronald W. Schafer, and Russel M. Mersereau. Demosaicking: color filter array interpolation. *IEEE Signal processing magazine*, 22(1):44–54, 2005. 4

[8] Keigo Hirakawa and Thomas W. Parks. Adaptive homogeneity-directed demosaicing algorithm. *IEEE Transactions on Image Processing*, 14(3):360–369, 2005. 4

[9] Jérôme Maye, Paul Furgale, and Roland Siegwart. Self-supervised calibration for robotic systems. In *Intelligent Vehicles Symposium (IV)*, 2013. 5

[10] Thomas Schöps, Johannes L. Schönberger, Silvano Galliani, Torsten Sattler, Konrad Schindler, Marc Pollefeys, and Andreas Geiger. A multi-view stereo benchmark with high-resolution images and multi-camera videos. In *CVPR*, 2017. 5

[11] Hannes Sommer, James Richard Forbes, Roland Siegwart, and Paul Furgale. Continuous-time estimation of attitude using B-splines on Lie groups. *Journal of Guidance, Control, and Dynamics*, 39(2):242–261, 2015. 5

[12] Jure Zbontar and Yann LeCun. Stereo matching by training a convolutional neural network to compare image patches. *Journal of Machine Learning Research*, 17(1-32):2, 2016. 7

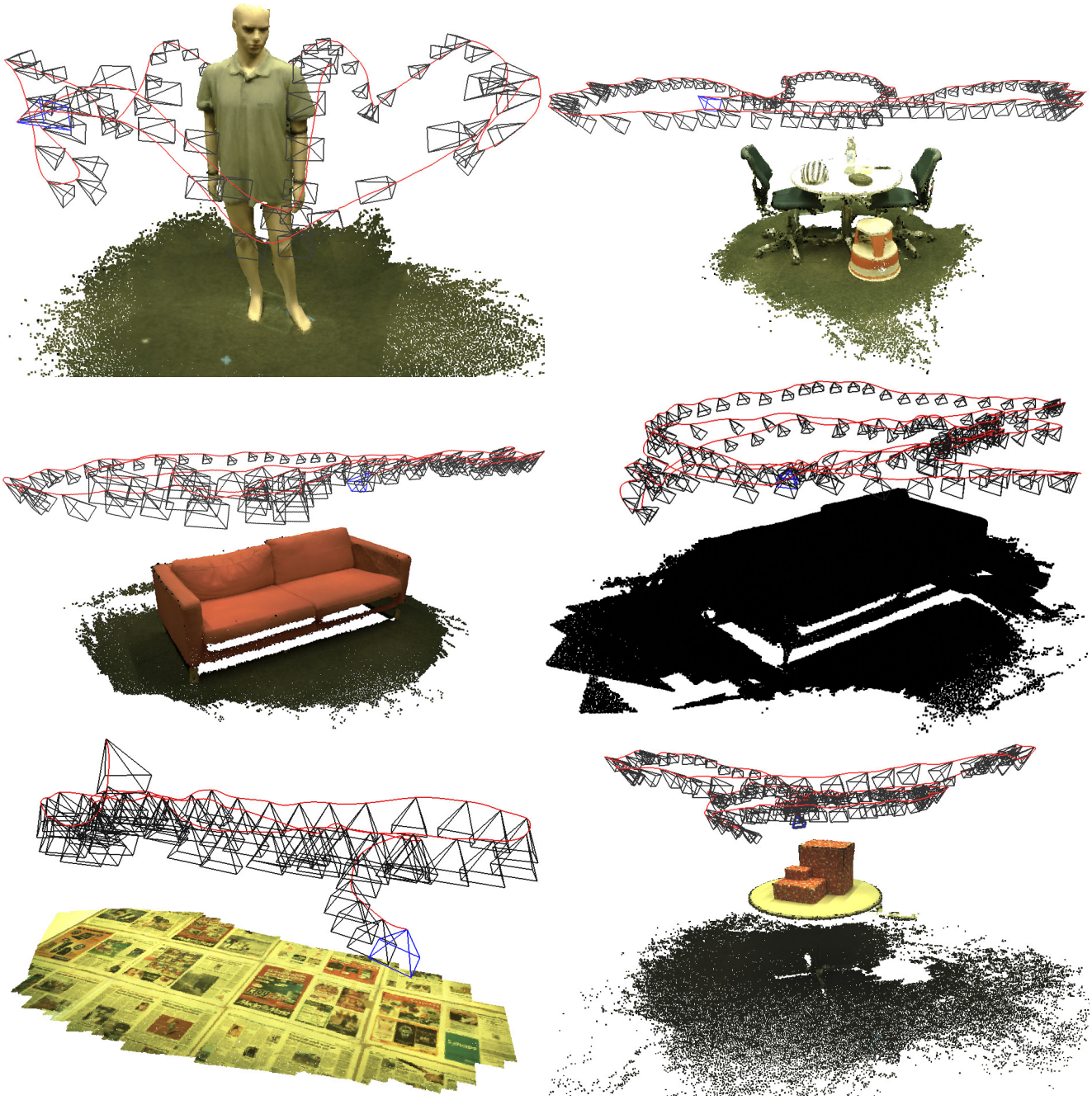

Figure 10. Example models and trajectories (with keyframes) reconstructed in real-time by BAD SLAM on datasets from our benchmark, showing the variety of datasets and reconstruction quality. Row-wise, left to right: Mannequin (643 frames, ca. 120'000 surfels). Table with objects (1180 frames, ca. 106'000 surfels). Sofa (976 frames, ca. 118'000 surfels). Sofa in darkness (1605 frames, ca. 210'000 surfels). Textured plane (630 frames, ca. 85'000 surfels). Presents (1554 frames, ca. 104'000 surfels).

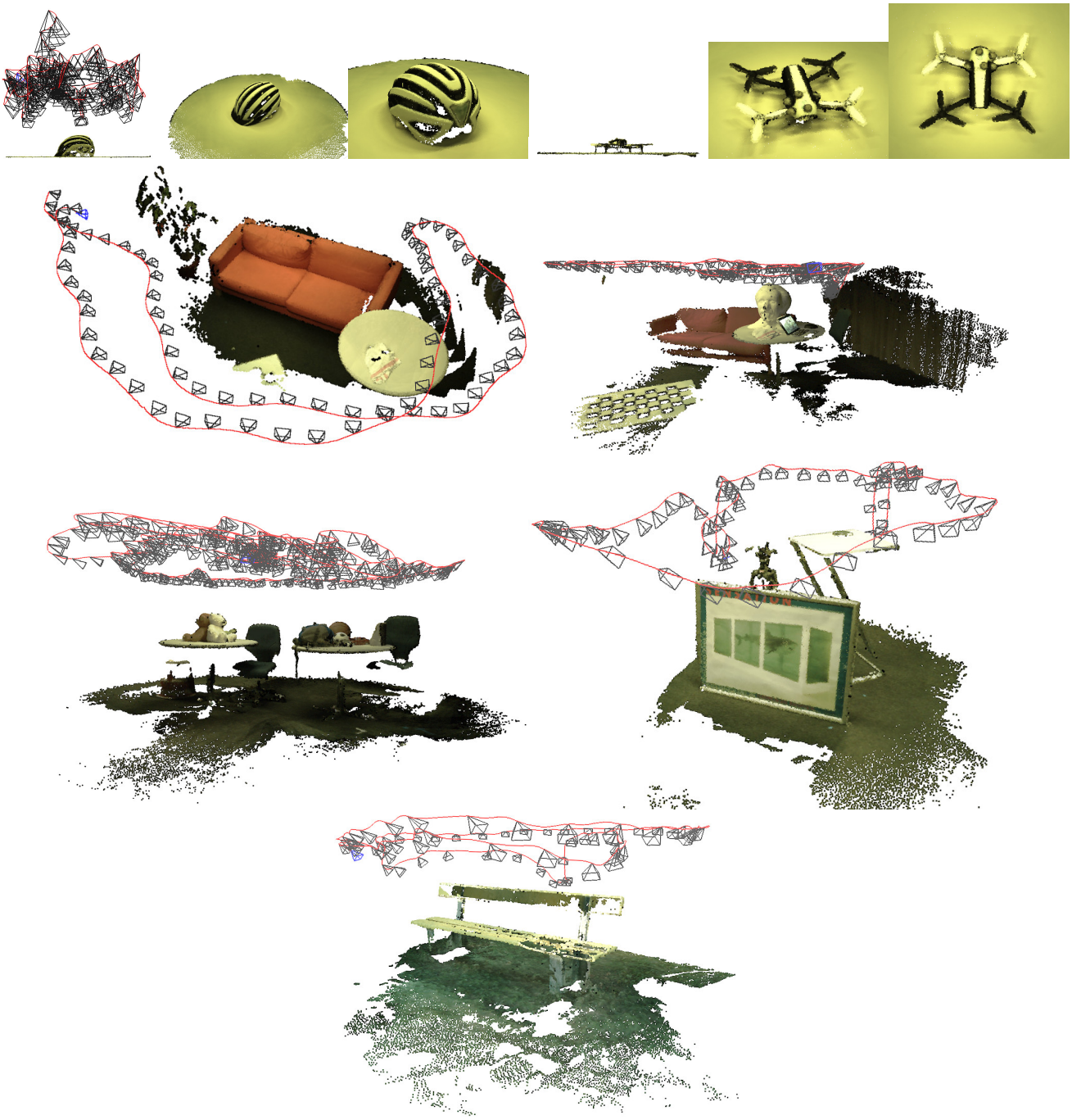

Figure 11. Continuation of Fig. 10. Row-wise, left to right: Three views of a reconstructed bike helmet (1704 frames, ca. 82'000 surfels; The first view is a side view which also shows that the table surface on which the helmet is placed is reconstructed as close to planar. This demonstrates both a good reconstruction and a high dataset quality.) Three views of a reconstructed drone with markers (1412 frames, ca. 63'000 surfels). Scene with table, sofa, and a plant (740 frames, ca. 81'000 surfels). Scene with checkerboard, sofa, and an Einstein head figure (1530 frames, ca. 131'000 surfels). Scene with two tables, in which the room lights are turned on and off during the sequence (2556 frames, ca. 189'000 surfels). Several objects (771 frames, ca. 121'000 surfels). Outdoors scan of a bench (660 frames, ca. 100'000 surfels).