Superaccurate Camera Calibration via Inverse Rendering

Morten Hannemose[a], Jakob Wilm[b], and Jeppe Revall Frisvad[a]

[a]DTU Compute, Technical University of Denmark
[b]SDU Robotics, University of Southern Denmark

ABSTRACT

The most prevalent routine for camera calibration is based on the detection of well-defined feature points on a purpose-made calibration artifact. These could be checkerboard saddle points, circles, rings or triangles, often printed on a planar structure. The feature points are first detected and then used in a nonlinear optimization to estimate the internal camera parameters. We propose a new method for camera calibration using the principle of inverse rendering. Instead of relying solely on detected feature points, we use an estimate of the internal parameters and the pose of the calibration object to implicitly render a non-photorealistic equivalent of the optical features. This enables us to compute pixel-wise differences in the image domain without interpolation artifacts. We can then improve our estimate of the internal parameters by minimizing pixel-wise least-squares differences. In this way, our model optimizes a meaningful metric in the image space assuming normally distributed noise characteristic for camera sensors. We demonstrate using synthetic and real camera images that our method improves the accuracy of estimated camera parameters as compared with current state-of-the-art calibration routines. Our method also estimates these parameters more robustly in the presence of noise and in situations where the number of calibration images is limited.

Keywords: camera calibration, inverse rendering, camera intrinsics

1. INTRODUCTION

Accurate camera calibration is essential for the success of many optical metrology techniques such as pose estimation, white light scanning, depth from defocus, passive and photometric stereo, and more. To obtain sub-pixel accuracy, it can be necessary to use high-order lens distortion models, but this necessitates a large number of observations to properly constrain the model and avoid local minima during optimization.

A very commonly used camera calibration routine is that of Zhang.[1] This is based on detection of feature points, an approximate analytic solution and a nonlinear optimization of the reprojection error to estimate the internal parameters, including lens distortion. Oftentimes, checkerboard corners are detected using Harris' corner detector,[2] followed by sub-pixel saddle-point detection, such as that of Förstner and Gülch,[3] which is implemented in OpenCV's `cornerSubPix()` routine. This standard technique can be improved for example by more robust and precise sub-pixel corner detectors[4,5] or use of a pattern different from the prevalent checkerboard.[6,7] A different line of work aims at reducing perspective and lens-dependent bias of sub-pixel estimates.[8,9] In the work of Datta,[10] reprojection errors are reduced significantly by iteratively rectifying images to a frontoparallel view and re-estimating saddle points. Nevertheless, such techniques are still dependent on how accurately and unbiased the corners/features were detected in the first place. Perspective and lens-distortion are then not considered directly, as their parameters are known only after calibration. Instead, the common approach is to try to make the detector mostly invariant to such effects. However, for larger features such as circles, it is questionable whether these can be detected in an unbiased way without prior knowledge of lens parameters. In addition, the distribution of the localization error is unknown and least-squares optimization may not be optimal.

In this paper, instead of relying solely on the sub-pixel accuracy of points in the image, we render an image of the calibration object given the current estimate of calibration parameters and the pose of the object. This non-photorealistic rendering of the texture of the calibration object can be compared to the observed image, which lets us compute pixel-wise differences in the image domain without interpolation. Because we are comparing

Further author information: (Send correspondence to M.H.)
M.H. E-mail: mohan@dtu.dk

differences in pixel intensities, we can model the errors as normally distributed which closely resembles the noise characteristics usually seen in camera images. This process is iterated in an optimization routine so that we are able to directly minimize the squared difference between the observed pixels and our rendered equivalent.

To ensure convergence of the optimization, the error must be differentiable with respect to camera parameters, object pose, and image coordinates. We ensure this by rendering slightly smoothed versions of the calibration object features.

2. RELATED WORK

We use a texture for our implicit rendering. This bears some resemblance to the version of texture-based camera calibration[11] where a known pattern is employed. We thus inherit some of the robustness and accuracy benefits that this method earns because it is not relying exclusively on feature extraction. Our optimization strategy is however simpler and more easily applied in practice as compared with their rank minimization problem with nonlinear constraints.

The work by Rehder *et al.*[12] is more closely related to ours. They argue that an initial selection of feature points (like corners) is an inadequate abstraction. As in our work, they use a standard calibration technique for initialization. With this calibration, they implicitly render the calibration target into selected pixels to get a more direct error formulation based on image intensities. This is then used to further refine different calibration parameters through optimization. Their approach results in little difference from the initial calibration values in terms of intrinsic parameters. Instead, they focus on the use of their technique for estimating line delay in rolling shutter cameras and for inferring exposure time from motion blur. Rehder *et al.* select pixels for rendering where they find large image gradients in the calibration image. Our pixel selection scheme is different from theirs: we use all the pixels that the target is projected to, and our objective function is different.

In more recent work, Rehder and Siegwart[13] extend their direct formulation of camera calibration[12] to include calibration of inertial measurement units (IMUs). In this work, the authors introduce blurring into their renderings to simulate imperfect focusing and motion blur. We also use blurring, and their objective function is more similar to ours in this work. However, they still only select a subset of pixels for rendering based on image gradients, and they, again, did well in estimating exposure time from motion blur but did not otherwise improve results over the baseline approach.

In terms of improved image features, Ha *et al.*[7] proposed replacing the traditional checkerboard with a triangular tiling of the plane (a deltille grid). They describe a method for detecting this pattern and checkerboards in an image and introduce a method for computing the sub-pixel location of corner points for deltille grids or checkerboards. This is based on resampling of pixel intensities around a saddle point and fitting a polynomial surface to these. We consider this approach state-of-the-art in camera calibration based on detection of interest points, and we therefore use it for performance comparison.

3. METHOD

Our method builds on top of an existing camera calibration method. This is used as a starting guess for the camera matrix $\mathbf{K}_0$, the distortion coefficients $\mathbf{d}_0$ and the poses of each calibration object $\mathbf{R}_{i0}$, $\mathbf{t}_{i0}$. We use this to render images of calibration objects, which we compare with images captured by the camera. Based on this comparison, the optimizer updates the camera calibration until the result is satisfactory. An outline of our method is in Figure 1.

The rendering is based on sampling a smooth function $\mathrm{T_G}$ that describes the texture of the calibration object. The initial calibration with a standard technique reveals which pixels the calibration target covers. Each of these pixels is undistorted and projected onto the surface of the calibration object to get the coordinates for where to sample $\mathrm{T_G}$. This inverse mapping from pixel coordinates to calibration object coordinates is advantageous with respect to calculating the gradient used in optimization. Each sampled $\mathrm{T_G}$ value is compared with the intensity of its corresponding pixel, and the sum of squared errors is the objective function that we minimize.

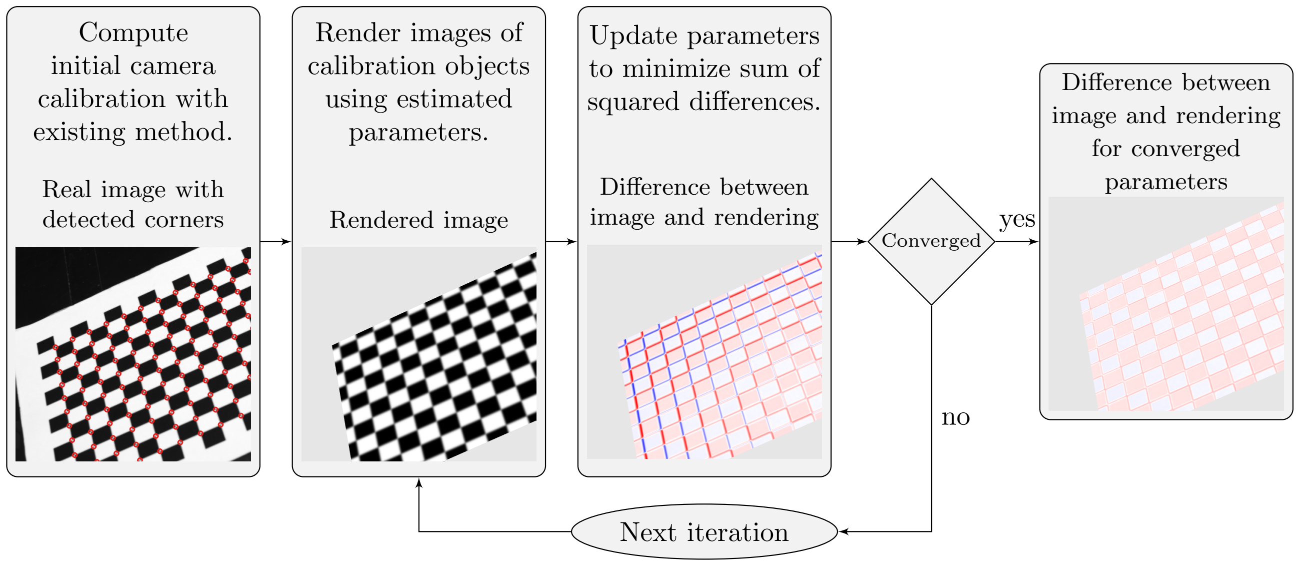

Compute initial camera calibration with existing method.
Real image with detected corners
Render images of calibration objects using estimated parameters.
Rendered image
Update parameters to minimize sum of squared differences.
Difference between image and rendering
Converged
yes
no
Difference between image and rendering for converged parameters
Next iteration

Figure 1: Overview of our method with a checkerboard as an example. All images are crops of a larger image. Difference images: Red represents positive values and blue represents negative values. For clear visualization, we have multiplied the focal length by 1.01 in our initial guess. Note that the converged difference image still resembles a checkerboard pattern because we do not compensate for the board's albedo.

3.1 Projection of points

Let us introduce a function that projects points in $\mathbb{R}^3$ to the image plane of a camera, including distortion

$$\mathrm{P}\left(\mathbf{K}, \mathbf{d}, \mathbf{p}\right) = \begin{bmatrix} x \\ y \end{bmatrix}. \tag{1}$$

Here, $\mathbf{K}$ is a camera matrix, $\mathbf{d}$ is a vector of distortion coefficients, and $\mathbf{p}$ is the point we are projecting, where the elements are

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 \end{bmatrix}, \quad \mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}. \tag{2}$$

P is then implemented as follows. First, the points are projected to normalized image coordinates:

$$\begin{bmatrix} x_\mathrm{n} \\ y_\mathrm{n} \end{bmatrix} = \frac{1}{p_z} \begin{bmatrix} p_x \\ p_y \end{bmatrix}, \quad r^2 = x_\mathrm{n}^2 + y_\mathrm{n}^2. \tag{3}$$

The normalized points are distorted using the distortion model of Brown-Conrady.[14,15]

$$\begin{bmatrix} x_\mathrm{nd} \\ y_\mathrm{nd} \end{bmatrix} = \begin{bmatrix} x_\mathrm{n} \\ y_\mathrm{n} \end{bmatrix} \left(1 + k_1 r^2 + k_2 r^4\right) + \begin{bmatrix} 2p_1 x_\mathrm{n} y_\mathrm{n} + p_2\left(r^2 + 2x_\mathrm{n}^2\right) \\ 2p_2 x_\mathrm{n} y_\mathrm{n} + p_1\left(r^2 + 2y_\mathrm{n}^2\right) \end{bmatrix}, \tag{4}$$

and the distorted points are converted to pixel coordinates

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f_x x_\mathrm{nd} + x_0 \\ f_y y_\mathrm{nd} + y_0 \end{bmatrix}. \tag{5}$$

3.2 Rendering

Let each calibration board have its own u, v coordinate system, and let $\mathbf{R}_i$ and $\mathbf{t}_i$ describe the pose of the ith board. Let $\mathbf{r}_{i_j}$ denote the ith column of $\mathbf{R}_i$. The 3D position of a point on a board is then

$$\mathbf{p}(i,u,v) = \begin{bmatrix}\mathbf{R}_i & \mathbf{t}_i\end{bmatrix}\begin{bmatrix}u\\v\\0\\1\end{bmatrix} = \begin{bmatrix}\mathbf{r}_{i_1} & \mathbf{r}_{i_2} & \mathbf{t}_i\end{bmatrix}\begin{bmatrix}u\\v\\1\end{bmatrix}. \tag{6}$$

Using a camera matrix $\mathbf{K}$ and distortion coefficients $\mathbf{d}$, we can project this point to the image plane

$$\begin{bmatrix}x\\y\end{bmatrix} = \mathrm{P}(\mathbf{K}, \mathbf{d}, \mathbf{p}(i,u,v)). \tag{7}$$

We can solve for u, v in terms of x, y in the above expression and obtain a new function:

$$\begin{bmatrix}u\\v\end{bmatrix} = \mathrm{P}^{-1}(\mathbf{K}, \mathbf{d}, i, x, y). \tag{8}$$

As mentioned, we use a function to define the texture of the calibration board (checkerboard, circles, deltille grid, or likewise). Let us call this texture function $\mathrm{T}(\mathbf{p}_{uv})$. Because natural images are not always sharp, we introduce a blurry version of the texture map by convolving it with a Gaussian kernel in u and v. This has the additional advantage that the texture map becomes smooth, which makes the objective function differentiable.

$$\mathrm{T_G}(\mathbf{p}_{uv}, \sigma_u, \sigma_v) = (G_{\sigma_u} * G_{\sigma_v} * \mathrm{T})(\mathbf{p}_{uv}). \tag{9}$$

This blur is applied in texture space, but we actually want it to be uniform around the interest point in image space. Thus, the standard deviations σ_u and σ_v, need to be corrected according to the length of the positional differential vector of the projection to the image plane. We introduce this quantity as

$$M_u = \left\|\frac{\partial \mathrm{P}(\mathbf{K}, \mathbf{d}, \mathbf{p}(i,u,v))}{\partial u}\right\|_2. \tag{10}$$

Inserting Equations (8) and (10) in Equation (9), we obtain a function that enables the rendering of an image of the calibration object:

$$\mathrm{C}_i(x, y, \sigma_{i,j}) = \mathrm{T_G}\left(\mathrm{P}^{-1}\left(\mathbf{K}, \mathbf{d}, i, x, y\right), \sigma_{i,j}/M_u, \sigma_{i,j}/M_v\right), \tag{11}$$

where M_v is the same as M_u but with respect to v and $\sigma_{i,j}$ is a measure of how blurry the image is around the jth interest point on the ith calibration board. This implies that the formula is only valid in the neighborhood of this point, and therefore we introduce

$$\mathcal{N}_{i,j} \tag{12}$$

to describe the set of pixel coordinates where the rendering is accurate. We choose $\mathcal{N}_{i,j}$ to be the pixels where the corresponding u, v coordinate is no further away than one half of the interest point spacing in Manhattan distance given by the initial camera calibration. For convenience of notation, let us define a set containing all $\mathbf{R}_i$, $\mathbf{t}_i$, and $\sigma_{i,j}$

$$\beta = \left\{\mathbf{R}_i, \mathbf{t}_i : i \in \{1, \ldots, n_i\}\right\} \cup \left\{\sigma_{i,j} : i \in \{1, \ldots, n_i\}, j \in \{1, \ldots, n_j\}\right\}, \tag{13}$$

where n_i is the number of calibration boards and n_j is the number of interest points on each calibration board. Using Equations (11) to (13), our optimization problem is then

$$\hat{\mathbf{K}}, \hat{\mathbf{d}}, \hat{\beta} = \underset{\mathbf{K},\mathbf{d},\beta}{\arg\min} \sum_{i=1}^{n_i} \sum_{j=1}^{n_j} \sum_{x,y \in \mathcal{N}_{i,j}} \Big(\mathrm{C}_i(x, y, \sigma_{i,j}) - I_i(x,y)\Big)^2, \tag{14}$$

where $I_i(x, y)$ is the intensity of the pixel at x, y in the image containing the ith calibration board. We parameterize $\mathbf{R}_i$ as quaternions and solve Equation (14) using the Levenberg-Marquardt algorithm.[16,17] Because Equation (9) is defined to give values between 0 and 1, in the case where $\sigma = 0$, our optimization problem is equivalent to maximizing the sum of pixels on white parts of T while minimizing the sum of pixels corresponding to black parts of T.

3.3 Computation of P^{-1}

Recall that P^{-1} is the function that, given a camera calibration and the pose of a calibration board, transforms from x, y in pixel space to u, v coordinates on the board. The first step in computing this is to invert Equation (5) by normalizing the pixel coordinates

$$\begin{bmatrix} x_{\mathrm{nd}} \\ y_{\mathrm{nd}} \end{bmatrix} = \begin{bmatrix} \frac{x-x_0}{f_x} \\ \frac{y-y_0}{f_y} \end{bmatrix}. \tag{15}$$

Inverting Equation (4) is not possible to do analytically, so we use an iterative numerical approach.[18] Note however that we can compute analytical derivatives of the inverse of Equation (4) by applying the inverse function theorem. To map the undistorted normalized coordinates to the calibration object, we combine Equations (3) and (6):

$$s \begin{bmatrix} x_{\mathrm{n}} \\ y_{\mathrm{n}} \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{r}_{i_1} & \mathbf{r}_{i_2} & \mathbf{t}_i \end{bmatrix}}_{\mathbf{H}_i} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}. \tag{16}$$

From this, it is clear that $\mathbf{H}_i$ is a homography transforming from the space of the ith calibration board to the normalized image plane. We invert the homography to perform the mapping

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{H}_i^{-1} \begin{bmatrix} x_{\mathrm{n}} \\ y_{\mathrm{n}} \\ 1 \end{bmatrix}. \tag{17}$$

Because the Levenberg-Marquardt algorithm is gradient-based, we need derivatives. We designed our texture function T_{G} to be smooth and differentiable, and fortunately the function P^{-1} is also differentiable, which implies that C_i is differentiable. Our implementation uses dual numbers for computing analytical derivatives.

4. RESULTS

When comparing a camera calibration to the ground truth, one could measure errors of each parameter individually,[11] but this is difficult to interpret, especially for distortion parameters as they can counteract each other. Motivated by this, we introduce *per-pixel reprojection error*, which measures the root mean squared distance in pixels between points projected with the true and estimated camera intrinsics. For each pixel, the image plane coordinates x, y define a line in $\mathbb{R}^3$ along which we select a point $\mathbf{q}_{xy}$ that projects to this pixel:

$$\mathrm{P}(\mathbf{K}, \mathbf{d}, \mathbf{q}_{xy}) = \begin{bmatrix} x \\ y \end{bmatrix}, \tag{18}$$

where $\mathbf{K}$ is the true camera matrix and $\mathbf{d}$ are the true distortion coefficients. We can now compute the per-pixel reprojection error E by using the estimated parameters to project the same points. Computing the differences, we have

$$E = \sqrt{\sum_{x,y} \left\| \mathrm{P}(\hat{\mathbf{K}}, \hat{\mathbf{d}}, \mathbf{q}_{xy}) - \begin{bmatrix} x \\ y \end{bmatrix} \right\|_2^2}, \tag{19}$$

where $\hat{\mathbf{K}}$ is the estimated camera matrix, $\hat{\mathbf{d}}$ are the estimated distortion coefficients and x, y sum over all possible pixel locations.

4.1 Synthetic data

We generate a set of 500 images of size 1920×1080 with a virtual camera with focal length $f = 1000$ each containing a single 17×24 checkerboard. The images are rendered so that the pixel intensities lie in the range $[0.1; 0.9]$. Figure 2 shows examples of these images. Each image is blurred by filtering it with a Gaussian kernel with zero mean and standard deviation σ. After this we add normally distributed noise to each pixel with zero mean and standard deviation σ_n, examples of this can be seen in Figure 3.

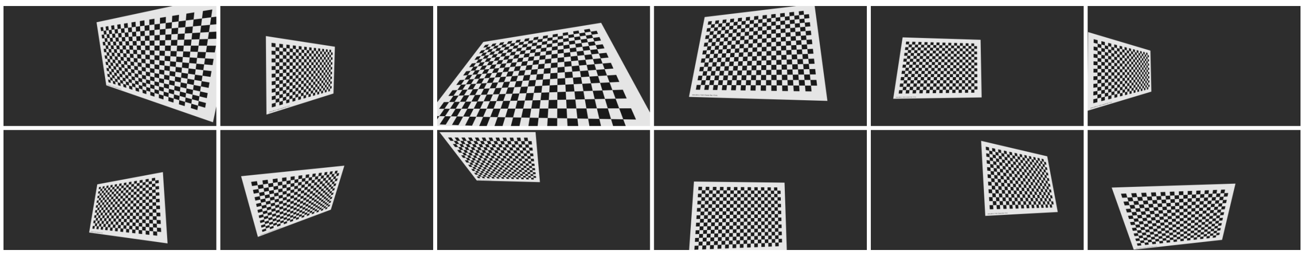
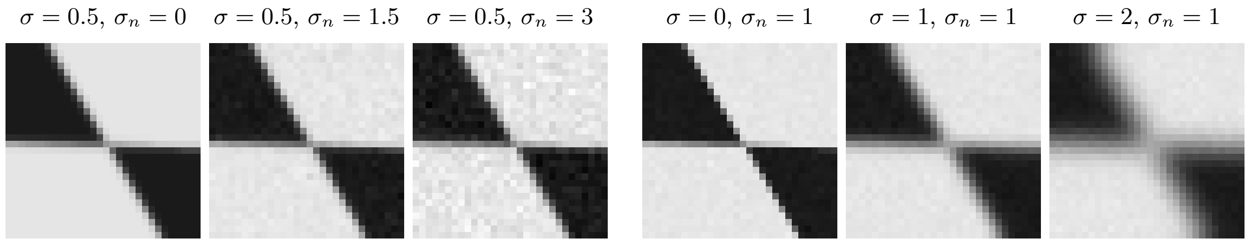


Figure 2: Sample images from our synthetic image dataset.


$\sigma = 0.5$, $\sigma_n = 0$
$\sigma = 0.5$, $\sigma_n = 1.5$
$\sigma = 0.5$, $\sigma_n = 3$
$\sigma = 0$, $\sigma_n = 1$
$\sigma = 1$, $\sigma_n = 1$
$\sigma = 2$, $\sigma_n = 1$
Figure 3: A corner from a checkerboard in a synthetic image with various levels of blur and noise added.

We select n random images from these and use the checkerboard detector from Ha *et al.*[7] with default parameters to detect points. After this, we use the standard method by Zhang[1] to compute the camera calibration. This is a calibration we compare with (Ha *et al.*), but also our initial guess for Equation (14). We do this for $n \in \{3, 20, 50\}$ and for varying values of σ and σ_n. For each n, σ and σ_n we perform 25 trials with randomly sampled images. The results of these experiments are in Figure 4. For comparison with OpenCV,[18] we use the detected points as initialization for `cornerSubPix`,[3] with a 5×5 window. As the images are rendered without distortion, we do the calibration without distortion as well.

We observe that our method performs better than Ha *et al.*[7] and OpenCV[3,18] for each n across various levels of noise and blur, except for $n = 3$ in cases with much blur. We also observe that our method is consistently better in noisy situations.

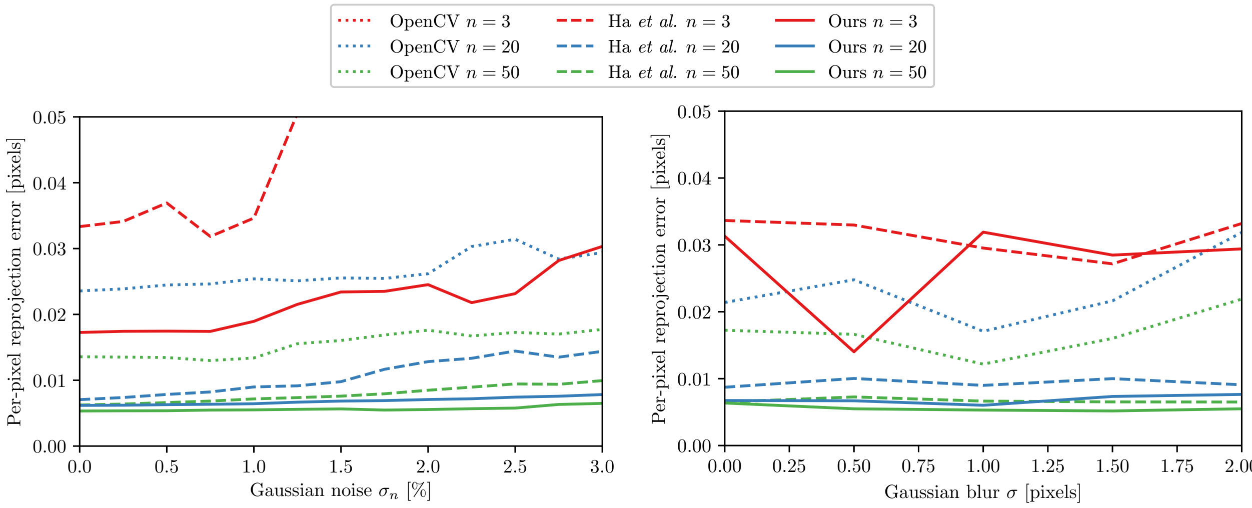


Figure 4: Comparison of our method with Ha *et al.*[7] and OpenCV[3,18] for varying number of images used in calibration (n). Left: Varying σ_n with fixed $\sigma = 0.5$. Right: Varying σ with fixed $\sigma_n = 1\%$. OpenCV $n = 3$ lies beyond the plotted area.

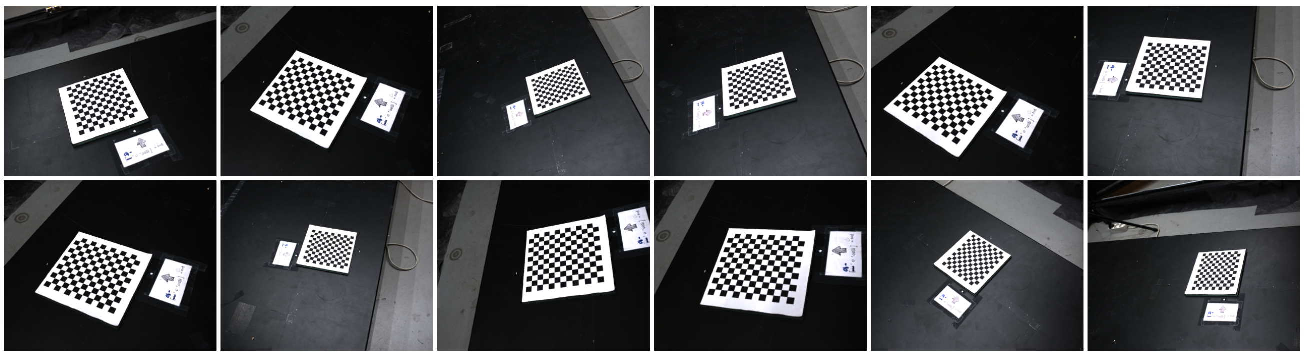


Figure 5: Sample images from our real image dataset.

4.2 Real data

When comparing camera calibrations on real data, an often reported measure is the reprojection error of all points. That is however not something we are able to do as our method incorporates the constraint of the calibration object geometry, and the reprojection error will thus per definition always be zero. Based on Figure 4, the points detected by the detector from Ha *et al.*[7] are clearly quite accurate, which motivates us to use it as a pseudo-ground truth.

We use a dataset of 128 images at a resolution of 3376×2704, each containing a 12×13 checkerboard. We randomly select 64 images to use as our test set, and detect points in them with Ha *et al.*[7] which we use as pseudo-ground truth. Then we select n of the images not in the test set and use them to compute the camera intrinsics. For each image in the test set, we use the already detected points together with our camera calibration, to compute the pose of the checkerboard, which in turn allows us to project points to the camera, and thereby measure a reprojection error. For each n we partition the 64 images in the training set into non-overlapping sets of size n and do the camera calibration for each of them. Figure 6 and Table 1 show the performance of our method compared to Ha *et al.*[7] and OpenCV.[3] For $n < 5$ our method performs better and has a lower standard deviation. For large n, Ha *et al.*[7] achieve an extremely similar reprojection error, but the points we are computing the reprojection error against are also detected by their method. It can also be seen that the reprojection error of their method on the training data approaches the same values from below, so the best achievable test set reprojection error is limited either by the camera model or the accuracy of the detected points.

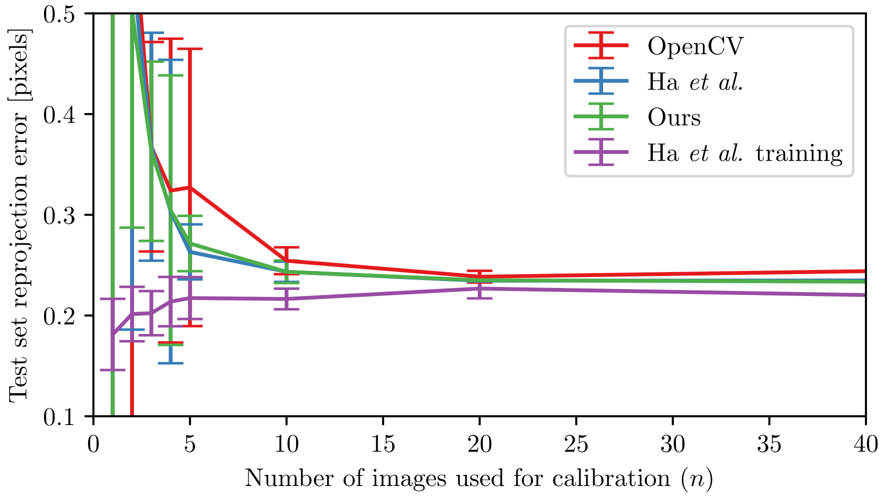


Figure 6: Reprojection error for images in the test set of our real dataset as a function of n. Bars on each point show ± 1 standard deviation.

Table 1: Data from Figure 6. $\pm$ indicates the standard deviation of the reprojection error.

n	2	3	4	5
OpenCV	0.61 ± 0.52	0.37 ± 0.10	0.32 ± 0.15	0.33 ± 0.14
Ha *et al.*	0.55 ± 0.36	0.37 ± 0.11	0.30 ± 0.15	0.26 ± 0.03
Ours	0.50 ± 0.22	0.36 ± 0.09	0.30 ± 0.13	0.27 ± 0.03

5. DISCUSSION

Although we have only used this method to compute intrinsics of a single camera in this paper, it is straightforward to extend to intrinsics of multiple cameras and their extrinsics. The homography $\mathbf{H}_i$ can easily incorporate the pose of the camera, and then all one needs is a separate set of parameters per camera.

Even though we have chosen to use the Brown-Conrady[14,15] distortion model in our work, this is a choice mostly motivated by being able to fairly compare with OpenCV.[18] Our method is not tailored to this distortion model, and one could replace it with another, such as the division model.[19]

We do not attempt to match the scaling of the image intensities in the rendering as in the work of Rehder *et al.*[12] We experimented with scaling the image intensities to match the rendering or including the local intensity of the rendering as a parameter in the optimization as well, but we did not observe any increase in accuracy when doing this.

Our method takes around three minutes to solve the optimization problem for 40 images from our real dataset, where each image is 9 Megapixels. We find this to be an acceptable computation time, especially given that even one such problem contains around 30 million residuals.

6. CONCLUSION

We have introduced a method for improving camera calibration based on minimizing the sum of squared differences between real and rendered images of textured flat calibration objects. Our rendering pipeline consists purely of analytically differentiable functions, which allows for exact gradients to be computed making the convergence of the optimization more robust and fast, while still allowing us to blur the image in the image space as would naturally occur. On synthetic data, our method outperforms state-of-the-art camera calibration based on point detection, for images distorted by Gaussian blur and noise.

On real data, our method exhibits a clear advantage when only a few images are available for calibration, and performs at least as well for a larger number of images, but we have not been able to verify whether our method outperforms the existing methods in this case, due to the difficulty of evaluating which of two estimated camera intrinsics is better.

REFERENCES

[1] Zhang, Z., "A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence* **22** (2000).

[2] Harris, C. and Stephens, M., "A combined corner and edge detector," in [*Proceedings of the Alvey Vision Conference*], Taylor, C. J., ed., 23.1–23.6 (1988).

[3] Förstner, W. and Gülch, E., "A fast operator for detection and precise location of distinct points, corners and centres of circular features," in [*Proc. ISPRS intercommission conference on fast processing of photogrammetric data*], 281–305, Interlaken (1987).

[4] Geiger, A., Moosmann, F., Car, Ö., and Schuster, B., "Automatic camera and range sensor calibration using a single shot," in [*International Conference on Robotics and Automation (ICRA)*], 3936–3943, IEEE (2012).

[5] Chen, B., Xiong, C., and Zhang, Q., "CCDN: Checkerboard corner detection network for robust camera calibration," in [*International Conference on Intelligent Robotics and Applications (ICIRA)*], 324–334, Springer (2018).

[6] Ding, W., Liu, X., Xu, D., Zhang, D., and Zhang, Z., "A robust detection method of control points for calibration and measurement with defocused images," *IEEE Transactions on Instrumentation and Measurement* **66**(10), 2725–2735 (2017).

[7] Ha, H., Perdoch, M., Alismail, H., So Kweon, I., and Sheikh, Y., "Deltille grids for geometric camera calibration," in [*The IEEE International Conference on Computer Vision (ICCV)*], 5344–5352 (Oct 2017).

[8] Lucchese, L. and Mitra, S. K., "Using Saddle Points for Subpixel Feature Detection in Camera Calibration Targets," *Computer Engineering* , 191–195 (2002).

[9] Placht, S., Fürsattel, P., Mengue, E. A., Hofmann, H., Schaller, C., Balda, M., and Angelopoulou, E., "Rochade: Robust checkerboard advanced detection for camera calibration," in [*European Conference on Computer Vision (ECCV)*], 766–779, Springer (2014).

[10] Datta, A., Kim, J. S., and Kanade, T., "Accurate camera calibration using iterative refinement of control points," *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops 2009* , 1201–1208 (2009).

[11] Zhang, Z., Matsushita, Y., and Ma, Y., "Camera calibration with lens distortion from low-rank textures," in [*Conference on Computer Vision and Pattern Recognition (CVPR)*], 2321–2328, IEEE (2011).

[12] Rehder, J., Nikolic, J., Schneider, T., and Siegwart, R., "A direct formulation for camera calibration," in [*International Conference on Robotics and Automation (ICRA)*], 6479–6486, IEEE (2017).

[13] Rehder, J. and Siegwart, R., "Camera/IMU calibration revisited," *IEEE Sensors Journal* **17**(11), 3257–3268 (2017).

[14] Brown, D. C., "Decentering distortion of lenses," *Photogrammetric Engineering and Remote Sensing* (1966).

[15] Conrady, A. E., "Decentred lens-systems," *Monthly notices of the royal astronomical society* **79**(5), 384–390 (1919).

[16] Levenberg, K., "A method for the solution of certain non-linear problems in least squares," *Quarterly of applied mathematics* **2**(2), 164–168 (1944).

[17] Marquardt, D. W., "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the society for Industrial and Applied Mathematics* **11**(2), 431–441 (1963).

[18] Bradski, G., "The OpenCV Library," *Dr. Dobb's Journal of Software Tools* (2000).

[19] Fitzgibbon, A. W., "Simultaneous linear estimation of multiple view geometry and lens distortion," in [*Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*], I–125–I–132, IEEE (2001).