# Report: Calibration & SfM

Computer Vision - Assignment 3

**Ian SCHOLL**
ischoll@student.ethz.ch

## 1 Calibration

*Given the camera models that you saw in the lecture, which part of the full model is not calibrated with our approach?*

Considering the camera models we have seen in class and that are also discussed in [1], we indeed only compute a part of the internal camera calibration in the following paragraphs and code, namely the matrices K and R, as well as the translation t. However, we do not take into account the non-linearities and therefor the distortion of the camera (respectively we are already using an undistorted model/dataset). Therefore we are not calibrating the *distortion part* of the full model, which means the radial correction coefficients (see chapter 7.4 in [1]) in our case don't have to be computed in our case.

### a) Data Normalization

*Briefly explain the potential problems if we skip this step in our algorithm.*

Using the DLT with non-normalized data will only yield exact results for very exact data and infinite precision arithmetics. However, there will always be noise in a practical example, and infinite precision is also not achievable when solving such a task on a computer, both of which will then make the solution diverge – drastically in some cases – from the correct output.
This is because the DLT depends on the coordinate frame in which its data points are expressed (as described in chapter 4.4.4. of [1]). Using normalization allows for the algorithm to be invariant to how the coordinate origin and scale are chosen, which often can be done arbitrarily, such as for the images in our case. It provides us with a canonical coordinate system which then makes the DLT invariant to any similarity transformations in the data.

### b) DLT

*How many independent constraints can you derive from a single 2D-3D correspondence?*

Every 2D-3D correspondence provides *2 constraints*, as we can see from how we build the constraint matrix in the method *BuildProjectionConstraintMatrix*. For each new 2D-3D point pair we can compute two rows of the constraint matrix, which namely are two independent constraints of the form: $[0^T \; -X^T \; x_2X^T] \; P_{vec}{}^T = 0$ & $[X^T \; 0^T \; -x_1X^T] \; P_{vec}{}^T = 0$ (see also chapter 7.2 in [1]). This means, that just 5.5 respectively 6 correspondences in theory would be enough to derive the projection matrix (as P up to scale has 11 independent unknowns).
All the remaining steps in order to recover $P^{hat}$ are pretty straight-forward, only including linear algebra such as matrix multiplications and solving for null-spaces for example employing SVD decomposition.

### c) Optimizing reprojection errors

*How does the reported reprojection error change during optimization?*

In the file *optimization.txt* of my code folder I have listed both $P^{hat}$ before and after optimization, and the evolution of the squared norm of the reprojection error. The following are excerpts of the interesting parts:

```
Reprojection error 0.0006316426059796171  start                Reprojection error 0.0006253538889928932
Reprojection error 0.0006316426169331763  ↓          •         Reprojection error 0.0006253538899283057  ↓
Reprojection error 0.0006316426297633268  ↓          •         Reprojection error 0.0006253538889928334  ↓
Reprojection error 0.0006316426065116134  ↓          •         Reprojection error 0.0006253538899291166  end
```

$P^{hat}$, BEFORE optimization

```
[[-3.422e-01  2.900e-01  5.684e-04 -4.358e-02]
 [ 2.153e-01  2.576e-01 -5.824e-01 -8.058e-02]
 [-1.385e-01 -1.729e-01 -1.218e-01  1.000e+00]]
```

$P^{hat}$, AFTER optimization

```
[[-3.421e-01  2.899e-01  6.869e-04 -4.334e-02]
 [ 2.158e-01  2.577e-01 -5.819e-01 -8.068e-02]
 [-1.401e-01 -1.729e-01 -1.222e-01  1.000e+00]]
```

We can see that the optimization has an effect, namely the squared norm of the reprojection error decreases by ~1%. Also $P^{hat}$ shows slight changes at the order of $10^{-4}$. The error was already considerably low to begin with, however. At the same time we can observe the reprojection output without and with optimizing P as

shown in Fig. 1 & 2. For instance when looking at the center dot, we can indeed recognize a slight improvement in the reprojection, even though the effects are rather minor. However, depending on the noise, such an optimization can be a valuable improvement of the DLT, especially for a more noisy scene.

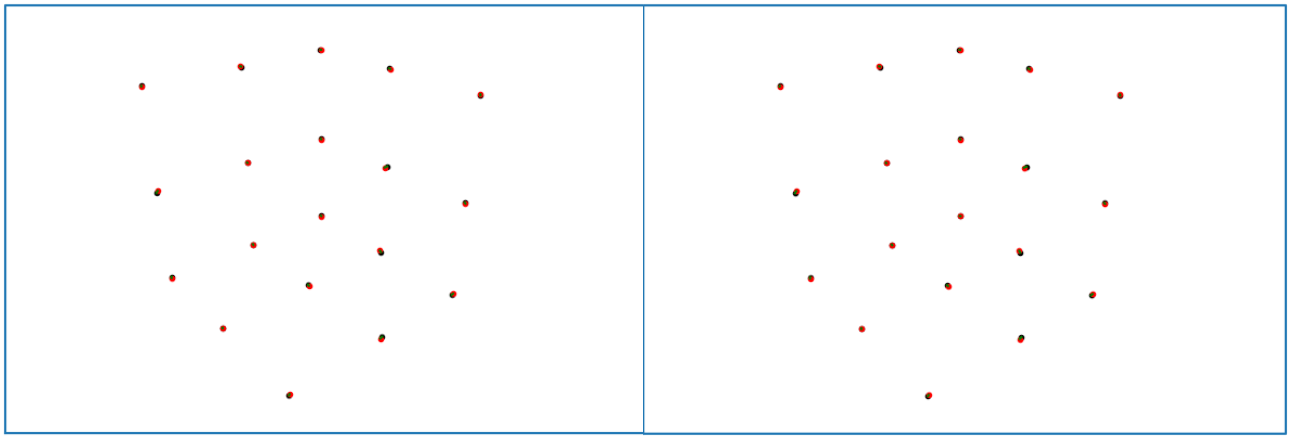

*Figure 1: Result without optimizing P* *Figure2: Result with optimizing P*

*Discuss the difference between algebraic and geometric error and explain the problem with the error measure $e = x \times PX$ in your report.*

The algebraic error is computed and directly minimized right when we try to solve for the null-space of the constraint matrix in order to get P. This means that it is the product of the constraint matrix with the projection matrix (A*P), which we try to minimize to a value of zero as good as possible (cf. Chapter 7.1 in [1]). The algebraic error can be seen as the distance between a 3D point and its back-projected ray, on a plane parallel to the principal camera plane. However, this measure of $e = x \times PX$ is neither geometrically nor statistically meaningful, and therefore minimizing this might not actually have the effect we want (namely an ideal projection and reprojection between 2D and 3D points, see also chapter 4.2.1 of [1]). A good normalization however can lead to good results even only after minimizing the algebraic error, as is the case in our example when above in this section good results have been presented even without optimizing P (and therefore minimizing the geometric error). It is also a computationally cheap and linear way of minimizing the error.

The geometric error, and more specifically the reprojection error, on the other hand looks at the 2D-distance between the initial 2D point and its corresponding back-projection through a 3D point. This indeed is a geometrically meaningful measure that will directly improve our model when used in the wished-for manner for minimization. It is however more costly to compute as usually several iterations are needed.

A good approach is a combined one such as we apply it in our problem, when you initially use the algebraic techniques to get a first minimization, complemented with further geometric refinement iterations.

## d) Denormalizing the projection matrix

Again, this is a straight-forward step where we only have to apply the formula $P = T^{-1}P^{hat}U$

## e) Decomposing the projection matrix

*Report your computed K, R, and t and discuss the reported re-projection errors before and after nonlinear optimization. Do your estimated values seem reasonable?*

K=
[[2.713e+03 3.313e+00 1.481e+03]
 [0.000e+00 2.710e+03 9.654e+02]
 [0.000e+00 0.000e+00 1.000e+00]]

R =
[[-0.774  0.633 -0.007]
 [ 0.309  0.369 -0.877]
 [-0.552 -0.681 -0.481]]

t =
[[0.047 0.054 3.441]]

For all three matrices, the numbers and orders of magnitude seem reasonable, as R clearly seems to be orthogonal (which is a property of a rotation matrix), and t shows a translation of the camera with respect to the world frame in the order of meters and centimeters. This again seems logical when considering the z axis of the camera frame seems to be roughly pointing at the world origin, so with the setup we are looking at here, a distance of ~3.5m in z-direction and slight translations in x- and y-direction indeed make sense. Also $\alpha_x$ and $\alpha_y$ of the K-matrix are practically equal, and it is upper triangular, which is again an expected and desired result. All of this seems to yield good visual results as seen in *f)*.

For the reported re-projection error before and after nonlinear optimization, see the discussion above in *c).*

### f) Visualization of results

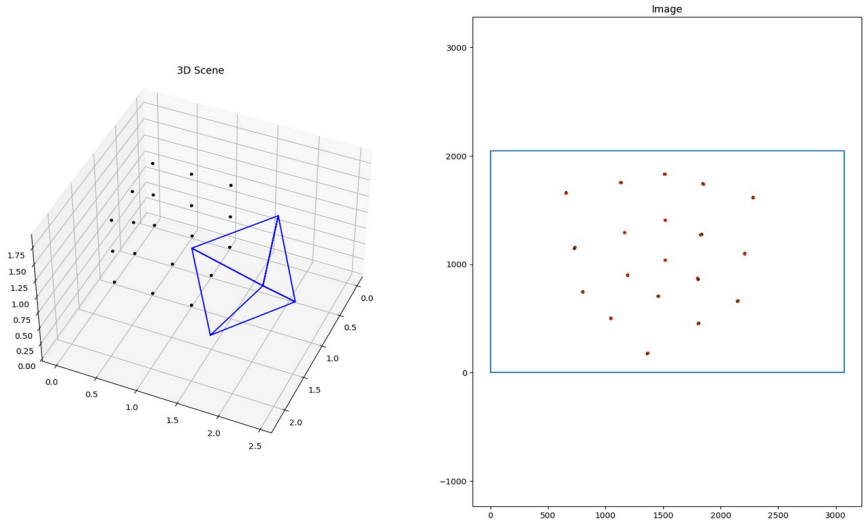


*Figure 3: Visual results of the calibration exercise: camera pose (left), and reprojected 3D points on the image plane (right)*

The visual results look very good: the camera pose seems very reasonable both in orientation and distance from the grid. And the reprojected red points seem to almost perfectly overlap the original points in black on the image plane, the green lines between them basically do not show at all.

# 2 Structure from Motion

## a) Essential matrix estimation

In order to get the keypoints to the normalized image plane, we can use the matrix multiplication with $K^{-1}$. I have also decided to *switch my constraint to* $\hat{x_2}^T E \hat{x_1} = 0$ (and therefore also the corresponding sanity check), as this saves me from having to switch around all the transformations at a later stage; this way I'm already computing the correct transformation from image 1 to 2 expressed in the correct frame. Each keypoint correspondence provides us with one constraint, which can be added to the constraint matrix. In theory, to calculate the 9 elements of E, which we want to know up to scale, 8 point correspondences would be enough (therefore called 8-point algorithm). Calculating the vector-matrix product and written out in elements, each of these constraints between two 2D points $\hat{x_1} = [x_1 \;\; y_1 \;\; 1]^T$ and $\hat{x_2} = [x_2 \;\; y_2 \;\; 1]^T$ then reads:

$$[\, x_2 x_1 \quad x_2 y_1 \quad x_2 \quad y_2 x_1 \quad y_2 y_1 \quad y_2 \quad x_1 \quad y_1 \quad 1 \,]\, E_{vec}^{hat} = 0$$

This means, the null-space projection of the constraint matrix reveals $E^{hat}$ in vectorized form (up to scale). Setting the wished for singular values then reveals E.

## b) Point triangulation

To filter all wrong 3D points, a simple check of the Z coordinate (should be <0) of the points projected into each camera frame and eliminating the corresponding points and correspondences does the job.

## c) Finding the correct decomposition

This now ready to implement *TriangulatePoints* method can also be implemented directly to check which of the four R-t combinations (always set directly as is for image 2; because image 1 is set to pose [I|0], the relative also directly are the global poses for image 2) yields the most correct points. The best one I then set as the R-t for the second image. The 3D points are later tied with indices to both images as correspondences.

## d) Absolute pose estimation

Uses the same normalization as *a)*, and the same constraint matrix as derived in *b)* of chapter *1 Calibration*.

## e) Map extension

Every further image can now be used to estimate its pose using the 2D-3D correspondences it has with already existing points. Then, new 3D points can be triangulated with already registered image, which finally are saved globally to extend the point cloud as well as indexed as new correspondences to the images.

## f) Visualization of results and general discussion

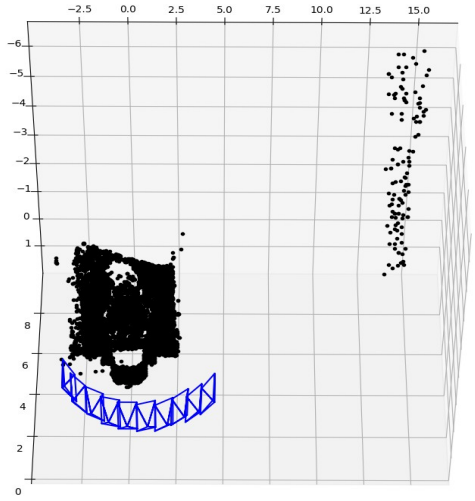

*Figure 4: Resulting pointcloud with SfM (initial images 0005 & 0006)*

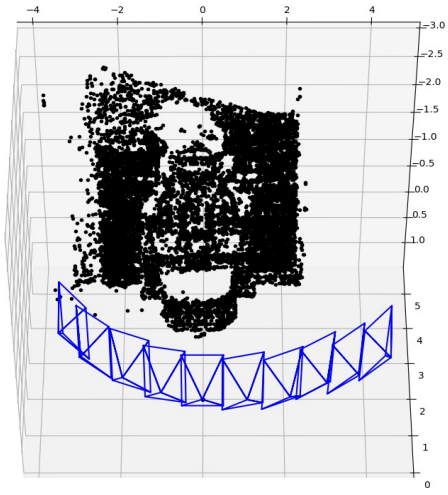

*Figure 5: Resulting pointcloud with SfM, filtered out remote wall*

In the final point cloud visualization (Fig. 4) we can clearly recognize the 3D silhouette of the fountain from the images, and even some remote wall is modeled on the back right, due to some feature match there on images 0008 and 0009 (can be observed when scatter plotting the features and matches on each image). The way the cameras are positioned around the fountain also seems correct. There is only very few outlier 3D points that don't seem to be computed correctly (likely due to noise).
We can filter out the remote wall from the point cloud in order to observe the fountain a bit better as seen in Fig. 5.

With my code I have found the initial images 0005 and 0006 to show some of the most stable and correct results. When for example initializing with images 0006 and 0007 however, we can see that with only two views everything seems fine (see Fig. 6). However, as soon as we introduce the other images as well, the camera poses are unstable and not computed correctly, and therefore the resulting point cloud is also not proper (see Fig. 7 & 8). This could possibly be explained with how we compute all the further image poses: in order to not complicate things, we use just the DLT without any optimization there. For some choices in initial images, this seems to be a good and stable approximation, however for other initializations the noise might just be too strong and is enough to disturb the correct functioning of the process. Therefore, the remaining image poses don't get computed correctly and the DLT without optimization fails.

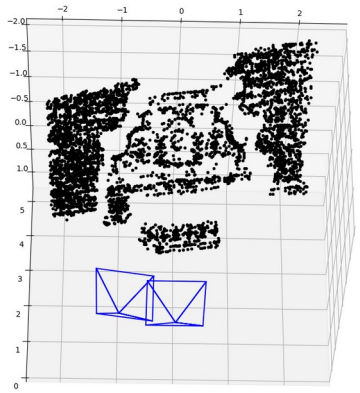

*Figure 6: Result using just images 0006 & 0007*

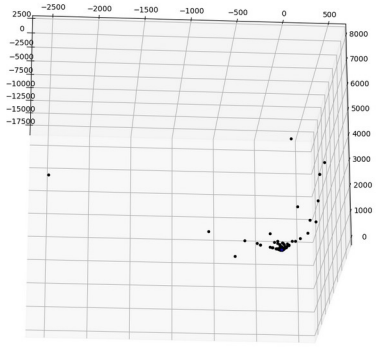

*Figure 7: Point cloud when initially using 0006 & 0007*

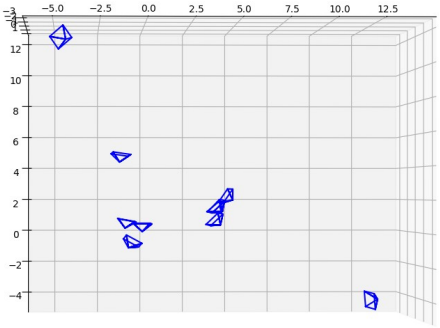

*Figure 8: Camera poses when initially using 0006 & 0007*

## Sources

[1] Hartley, R., & Zisserman, A. (2004). *Multiple View Geometry in Computer Vision* (2nd ed.). Cambridge: Cambridge University Press.