Assignment 3

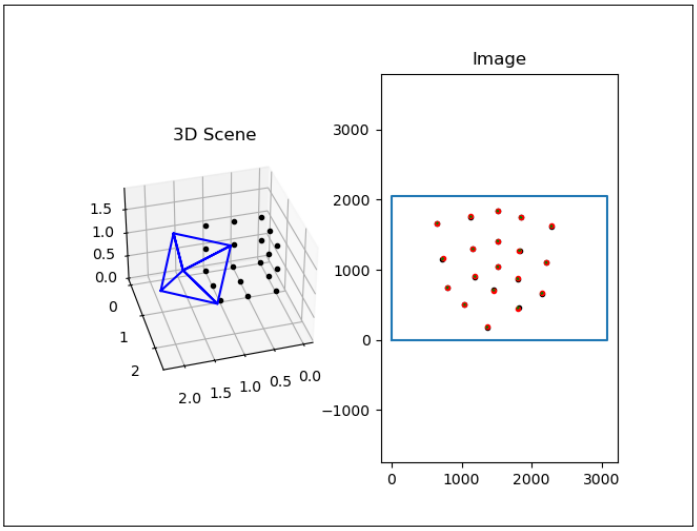
Computer Vision ETH, Fall 2021

Quentin Guignard

19th November, 2021

2 Camera Calibration

Figure 1: Result of the calibration where the camera is the blue shape looking at the points



Approach

The goal of the part is to find the transformation of the camera in space from some 2D key points in an image and their relative corresponding 3D image. In other words, given some 2D and 3D points (knowing their mappings) we need to find K, R and t which defines $P = K[R|t]$. In a first step we will find an estimate of P by solving a linear system by finding its null space. Indeed, we build a system $Cp = 0$ such that C is a constraint matrix. Indeed, we know that the points on the 2D image can be back-projected in space to find the corresponding 3D point. However, the projection transformation from a 3D to a 2D space is not a bijection therefore we have an ambiguity. Indeed we can only relate any 2D points to an image to a line where the 3D image would lie. Therefore, we stacked this constraint in C for any two pairs (x, X) such that $x = PX$. We then solved for p using an SVD. The solution is the singular vector yielding in a null

singular value which corresponds to a zero length axis of the hyper-ellipsoid transformation defined by the diagonal matrix of the SVD. This yields a solution to the null space. From here we have extracted P from p. We then needed to minimized the error $min\|x_i - PX_i\|$ because solving the constraint only optimized the algebraic error and not necessarily the geometric error. After this step, we were able to decompose the optimized P into K, R and t.

2.2.a Data Normalization *To avoid numerical instabilities it is good to normalize the data. Two simple methods are already implemented in NormalizePoints2D, NormalizePoints3D and you just need to apply them. Make sure to save the transformation matrices that were used for normalization to later be able to recover the original projection matrix. Briefly explain the potential problems if we skip this step in our algorithm*

We do need data normalization because we need to find the projection matrix. And in order to find the projection matrix we need to solve a consequent linear system. Our data contains noise and therefore contains an amount of error. Thus, this amount of errors grow up as when performing the height points algorithm as the coefficients of the projected and back-projected points will get multiplied (see the course slides). Therefore, normalizing the data helps for the algorithm to converge faster. Moreover, normalizing helps to improve the accuracy of the results and make them invariant of the choice of initial coordinate frames (p.107 Multiple View and Geometry).

2.2.b DLT *[...] You can interpret this as a vector product a * vec(P) = 0 and assemble A by stacking the as of all constraints. You need at least 11 independent constraints to determine the 11 degrees of freedom (DoF) of P, but for this exercise just use all available correspondences to minimize the least squares error. How many independent constraints can you derive from a single 2D-3D correspondence? [...]*

The constraint is initially expressed as three equations in function of x and X but only two of them are linearly independent. One of the row is up to scale as, given $x = (x_1, x_2)$, the sum of x_1 times one of the row and x_2 times the other. This yields to two linearly independent constraints for each mapping the constraint matrix.

2.2.c Optimizing reprojection errors *How does the reported reprojection error change during optimization? Discuss the difference between algebraic and geometric error and explain the problem with the error measure $e = x \times PX$ in your report.*

The re-projection error decreased during the optimization and went very close to zero which indicates that the minimization step worked properly. We also observe the on Figure 1 that the red dot are well aligned with the black dots on the 2D image.

```
Optimization terminated successfully    (Exit mode 0)
            Current function value: 0.0006253538899291131
            Iterations: 15
            Function evaluations: 209
            Gradient evaluations: 15
Reprojection error after optimization: 0.0006253538899291131
```

The DLT algorithm minimized $\|Cp\|$ and $Cp = e$ where e is the algebraic error vector associated to the points correspondences. The algebraic error distance is defined as $\|e\|$. The geometric distance is the squared euclidean distance of $d(x, \hat{x})$ such that $\hat{x} = PX$.

Finally, the hand-out suggest to discuss the following error measurement $e = x \times PX$. The latter represents a line by its normal vector on the image plane between the image point and its estimate. The problem is that in presence of noise we will never get a zero normal vector. To my understanding, this is why we use $\|Cp\|$ instead of $e = x \times PX$. Indeed, if $\hat{x} = PX$ never exactly falls over x then $\|e\| \neq 0$ considering $e = x \times PX$. More precisely, the error vector is ambiguous with all the points that are on the line $x \times PX$ and that effectively share the same normal vector.

2.2.d Denormalizing the projection matrix *Using the previously saved data normalization matrices, compute the proper projection matrix for the original input data. Hint: Write the normalized points in terms of the original ones, and compare the normalized equation to the non-normalized.*

See code.

2.2.e Decomposing the projection matrix *[...] Report your computed K, R, and t and discuss the reported reprojection errors before and after nonlinear optimization. Do your estimated values seem reasonable? [...]*

The reported values are as follows:

```
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]]
```

As seen in the course, K must have a positive diagonal and be a upper triangular matrix. We can see that this is the case. Moreover, we can see that the two first coefficient of the diagonal are the same which endorse our intuition that the unit of distance per pixel is the same as nothing is deformed in the 2D image result. Concerning the rotation matrix all the columns vectors have length 1 and the matrix is orthonormal which is what we expected. Also, by looking at the picture on Figure 1 the rotation, the translation vector and the camera looks fine in regards to the 2D image point of view.

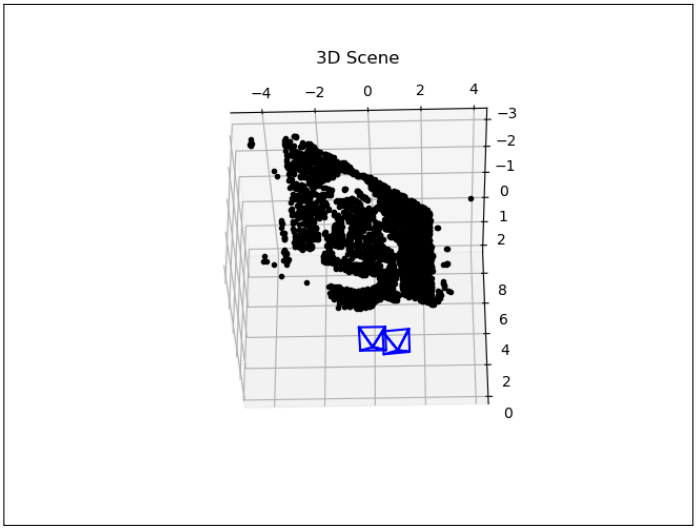
3 Structure from motion

Approach

The goal of this part is to reconstruct a scene from a sequence of images from different points of view. We want to be able to place the cameras in space relatively to the real point of view in the real scene where the picture was taken. This way we will also reconstruct the scene by a set of 3D points triangulated from all the images by looking at thier correspondences. The algorithm works as follows.

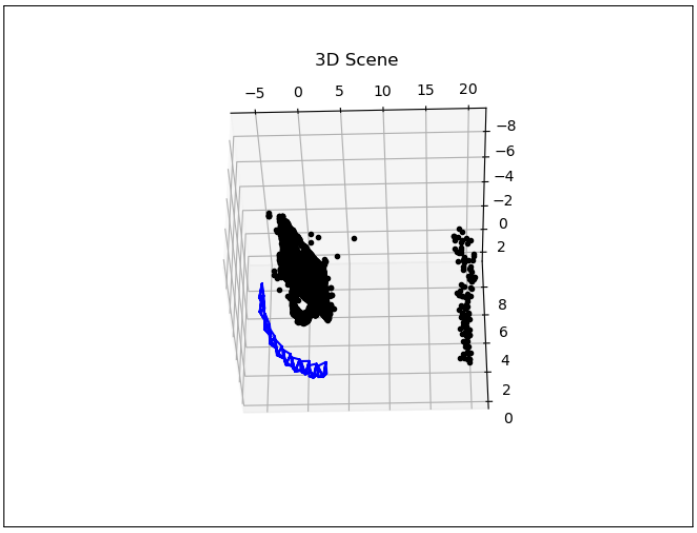
First, we need to find a starting set of 3D points triangulated from two randomly selected images. This way, for the rest of the images, we will be able to just use this starting 3D points set and constraint each key point on the image to be as much as possible aligned with them (intuitively). Therefore, give I_1 and I_2 we find the essential matrix E such that for $x_1 \in I_1$ and $x_2 \in I_2$ we have $x_1 E x_2 = 0$. The matrix E defines a correspondence mapping between I_1 and I_2 in the same way as F with the only exception that the cameras are calibrated and normalized meaning that $P_1 = [R|t]$ and $P_2 = [I|0]$. After having found E by having solved a linear system with the height points algorithm (we used a constraint matrix and a SVD), we can decompose E in four different poses that we will need to filter. Now that we have our poses we want to actually know which one is correct and we want to get the starting 3D point set. Therefore, the next step is to triangulate the points, again by solving a linear system of constraint (see course). We find constraints on each normalized mapping $x_1 = P_1 X_1$ and $x_2 = P_2 X_2$ and solve $AX = 0$ with an SVD to find a 3D point X_{12} that is projected on two corresponding points x_1 and x_2. We do this process for all four poses that we have found and take the one with the most points front of the camera which means taking points with component $Z > 0$ (in camera space) as the Z axis is the principal axis in camera space and is pointing forward. Once we have found the best pose we can simply keep track of our correspondences for the two first camera. We now know how to setup the first camera in the space and we have already some points that make our scene.

Figure 2: Reconstructed scene with two cameras



As we can see on figure two we can distinguish the fountain and both images 1 and 2 are next to each other which is what we expected. Now, we just have to process all images and increment our 3D points starting set by the new ones that arises from new point of views. Therefore, for each new image not yet registered, we find all the correspondences between our actual bank of 3D points and if there is enough points, we register this image by triangulating all possible new points between the currently considered image (not yet registered) and all the registered images. We then increment our 3D point set by those point. We proceed like this until we have registered all the images.

Figure 3: Reconstructed scene with all the cameras



As a side note, the result looked very bad with the images 3 and 4. After having seen a moodle post, we tried with images 1 and 2 and the result looked way better. We still cannot really explain the points on the side of the final result. They maybe are part of the wall?