Killian Kaempf

Computer Vision hw3

Given the camera models that you saw in the lecture, which part of the full model is not calibrated with our approach?
We estimate here only the intrinsic linear parameters, i.e R and t, but we don't estimate non linear parameters such as the lens distortion but it cannot be included here because of the method we use.

a) Data normalization

Already implemented. It is useful because we don't depend on the input scale. It allow us to be protected against overflows if we have huge values. There is also the fact that sometime a single coordinate may have more weight than the others because the values are much bigger and we avoid it with the normalization. Otherwise we may have shifted or distorted results when we look at intermediary plots depending on the code if we don't adapt the parameters. Normalization also allow us to avoid the management of the focal length during the computations. It makes it also easier to compare the losses.

b) DLT

For the DLT, we have the equation $[x1\ x2\ x3]\ P\ [X1\ X2\ X3]^T = 0$ and this gives us 3 equations but two linearly independents. So for each pair (x,X) we have 2 constraints and we can concatenate all these constraints into single matrix. Then we can obtain the null space of that matrix and it correspond to P. P is the projection matrix to go from a 3D point to a 2d point up to scale. Because here we have 19 points it means that we have 38 constraints which is more than enough.

How many independent constraints can you derive from a single 2D-3D correspondence?
For each data point that we have, we get **2 linearly independent equations** from the $[x]_xPX = 0$ equality. So we can derive 2 constraints. Let's also note that we have 11 df because P34 =1 by definition and the 11 other elements are unknown. So we need 11/2 = 5,5 points –> 6 points at least and they need to be non-degenerate.

c) Optimizing reprojection errors

Here we want to find the P that minimize the norm of the errors. We use the definition of the error: e = x – PX. To do this, we compute the error of each point by projecting the 3d point using P onto the 2d image and look at the error between the point and the 2d projection. We also need to note that we square the norm and that is this value we want to minimize.

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 × PX in your report.
It is minimized thanks to the scipy minimize function that we use and we keep the $P_{34} = 1$ during the whole optimization. Note that the error is not minimized at each step in a convex fashion but can increase at some time steps before decreasing again.
One of the key problem here is that the error is not a scalar but a vector. If we want to minimize the error, we need a scalar otherwise we have to make assumptions of the dimensions. There is also the fact that if x and PX are co-linear then the algebraic error is 0 even though they may be different points on the projected plan.

Killian Kaempf

d) Denormalizing the projection matrix

Here we do the inverse operations to unnormalize the Projection optimized P matrix. We use the matrices obtained during the 2d and 3d normalizations. We do that because we want to apply our new matrix P on unnormalized points. We use the relationship P= inv(T2D) * P_hat *(T3D)

e) Decomposing the projection matrix

Finally, we decompose the P matrix to obtain the intrinsic parameters (K) and rotation (R) matrices and the translation (t) vector. We find inv(R) and inv(K) using the qr decomposition and we make sure that the rotation is correct by having det(R) =1 and that K has positive diagonals (it is the focal length in pixel for x and y for the first two elements of the diagonal)! To do this we multiply K with T and M with inv(T) to have the positive diagonal for K. This allows us to compute the matrices K and R and the vector t=-RC with C as the null space of P obtained using svd method

Report your computed K, R, and t and discuss the reported reprojection 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]] intrinsic params

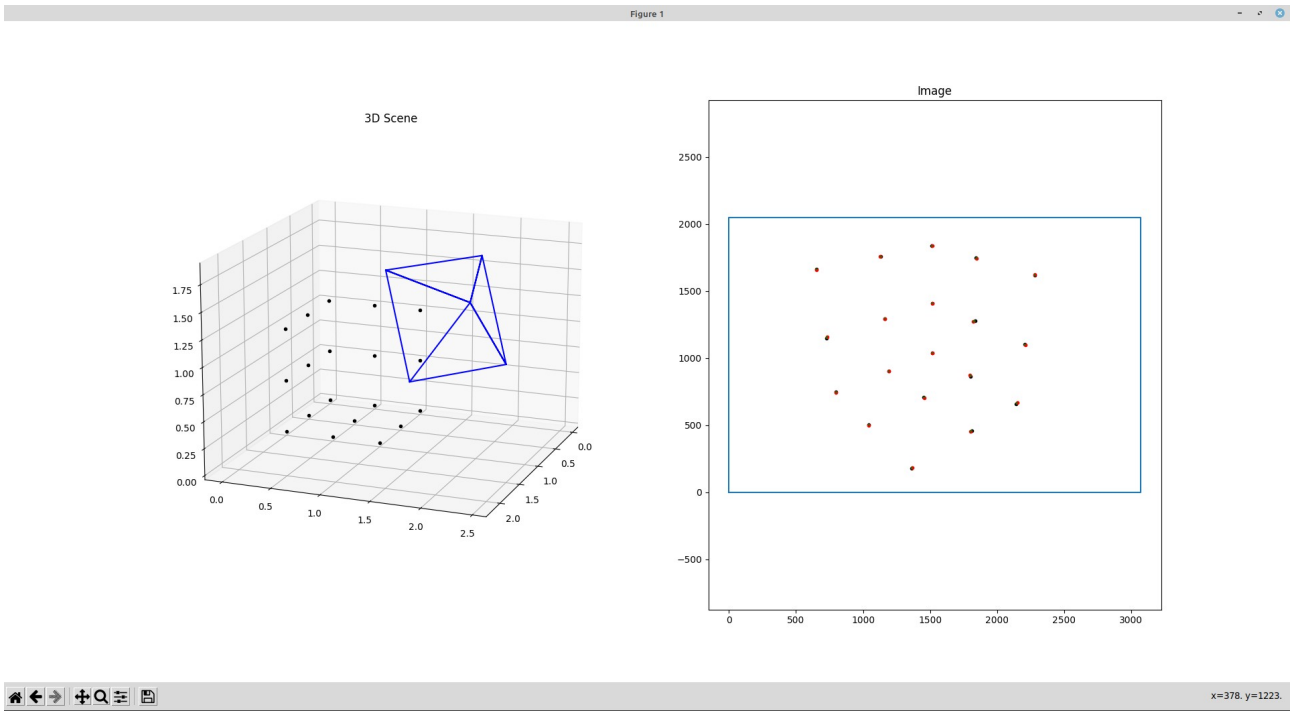
R =[[-0.774 0.633 -0.007]
 [0.309 0.369 -0.877]
 [-0.552 -0.681 -0.481]] Rotation matrix

t = [0.047 0.054 3.441] position of the origin of the world coordinate system expressed in coordinates of the camera-centered coordinate system

Reprojection error:
Before: 0.00063164
After: 0.00062535

We have an error reduction of 1% which is reasonable for a calibration for the P optimization .

The figure also look reasonable because on the left we see that the camera seems to be pointing at the correct direction if we play with the 3d plot and on the right we can see that the 1% seems reasonable when we look at the red and green dots. That means that the system was not too noisy.



Ex 2 Incremental SFM

First we load all the images and we pick two images that we will use as the two references to do the first triangulation. In my case I picked images 5 and 6 to have a good result. I noticed that some other choices of pairs were not that stable and it was giving not very good results

a) Essential matrix estimation

First, we normalize all the pixels of both images. Then we select the pairs of pixels based on the matches given. Now, instead of having one 2d point and one 3d point like in task 1, we have two 2d points so we have the equation x_1'Ex_2 where x1 and x2 come from a pair from matches. This gives us only one constraint per pair instead of two with the calibration. Then we compute the null space of this matrix of constrain matrix to find E_hat. We adapt so that the first two singular values are equal and that the last one is 0 and we obtain E.

Then using E we can find the 4 possible relative poses. These relative poses have to applied to the first image and not the second one because we want to move the 3d points into the camera space.

The pose of the second image to the identity for the rotation (no rotation) and 0 for the translation

b) Finding the correct decomposition

Nothing fancy here, we set the pose of the first image and triangulate to compute the number of points visible. We keep the one that has the more points visible.

c) Point triangulation

Most of the function is already implemented here. We add the 3d points for the corresponding points that are not in any of the two images. We use the Matrix A using the book formula to compute these 3d points based on the two 2D points. Once we have that list of 3d points, we

compute the matrices T1 and T2 to convert the 3d points into the two camera spaces and then filter to keep only the points that are in front of both camera (T1 and T2 are used here instead of P1 and P2 because we don't want to compute projections but we want the coordinate of the points based on the respective camera). The origin of the usage of T1 and T2 comes from this document: http://mi.eng.cam.ac.uk/~cipolla/publications/contributionToEditedBook/2008-SFM-chapters.pdf . That means that the Z index must be > 0. We return the corresponding points (3d coordinate and the 2d pixel index for each of the two images).

d) Absolute pose estimation

Already implemented. Compute the pose for new images that we want to add to the system. This correspond to the direction and position of the newly added camera based on the image that we want to add.

e) Map extension

This is a function called each time we want add an image to the already triangulated images (except for the first two). We triangulate each image with all already registered images and add the 3 points to the saved points of the two images used during the triangulation. We create a dictionary to keep track of all the indices of the pixels in the 2d images and which 3d points are tracked. We then only have to update the reconstruction state and update the list of all the 3D points. For each image registered and the one that we triangulate, we add all the 3d correspondences between the 2d indices of the images and the global index of the 3d pixel.
It has to be done in two steps because we need to have the global index of the 3d points and this lists keeps increasing as long as we add new images. When we run the triangulation we use the local indices for the 3d points so we need to add an offset to point to the same point but at his emplacement in the other list.
We triangulate all images one after the other (computing the pose before the triangulations for each new image) and once it is done, we only have to plot the cameras and the 3d points and the task is over.

Result:

Killian Kaempf