ETH Zürich
Computer Science Department

Computer Vision

Camera calibration and Structure from Motion

Objective:

This exercise consists of two parts. First, you will calibrate a camera based on correspondences between the image and points on a calibration target. Second, you will produce a reconstruction of a small scene using Structure from Motion (SfM) methods. We will rely mostly on the Direct Linear Transform (DLT) method to compute the scene geometry from correspondences. If you get stuck during the exercise you can check the exercise session slides for some additional hints.

2.1 Setup

We recommend to use a virtual environment to avoid cluttering your Python installation. To set this up using the Python `venv` module on Linux just use

```
cd code
python3 -m venv venv
source venv/bin/activate
pip install --upgrade pip
pip install -r requirements.txt
```

In case you want to use a different setup, just install the dependencies in `requirements.txt` manually.

2.2 Calibration

For calibration we usually rely on calibration targets with known geometry and easy to identify visual features. Traditionally, checkerboard patterns as shown in fig. 1 are a popular choice, but other targets with known 3D points can work just as well if you have the correct correspondences. The exercise code is provided in `calibration.py`.

For your convenience we provide you with correspondences in the framework, but you can try to replace them if you want to calibrate your own camera. (In that case you need to make sure that the 3D points are not all coplanar, otherwise our used method will fail.) The code framework provides some visualization of this data. Given the correspondences, the (simplified) process of calibrating the camera consists of data normalization, estimating the projection matrix using DLT, optimizing based on the reprojection errors and then decomposing it into camera intrinsics and pose. You will need to fill in some parts of the code for each step. Give a brief explanation of the approach in your own words in the report

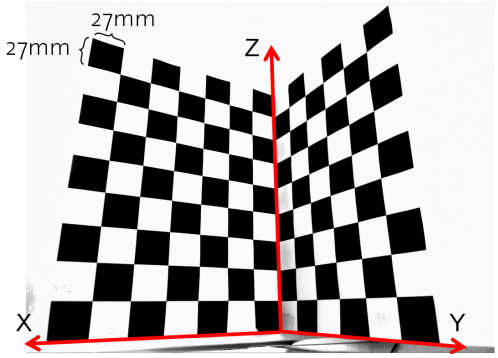

Figure 1: Typical calibration object.

and address the questions given in the individual tasks. Given the camera models that you saw in the lecture, which part of the full model is not calibrated with our approach?

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.

b) DLT

The basic projection constraint for a 2D point x and 3D point X is $x \propto PX$ where $\propto$ means equality up to scale. We can get rid of the scale factor by writing

$$x \propto PX \rightarrow x \times PX = [x]_\times PX = 0$$

where $\times$ and $[\cdot]_\times$ mean the cross product and cross product matrix, respectively. With DLT we reformulate the constraint again to obtain a constraint matrix A so that $A\ vec(P) = 0$. Here, $vec(P)$ is a vectorized version of the projection matrix P,

$$vec(P) = [P_{11}, P_{12}, \ldots, P_{34}]^T \ .$$

Assembling the constraint matrix

To obtain A you can explicitly write the equations of the linear system and rearrange them to the form

$$P_{11} \cdot (\ldots) + P_{12} \cdot (\ldots) + \cdots = 0$$

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?

Solving for the nullspace of A

As you can see from the equations, $vec(P)$ is just the nullspace of A. We can find this

efficiently using singluar value decomposition (SVD). This is already implemented in the framework and you don't need to do it yourself.

c) Optimizing reprojection errors
DLT minimizes an *algebraic* error which is 0 if the constraint is fulfilled perfectly, but is not necessarily a meaningful error measurement otherwise. Given the estimate of P we can refine it using *geometric* errors. For this you need to implement the reprojection error $e = x - PX$. Make sure to normalize the homogeneous coordinates, otherwise you will get wrong error measurements. The rest of the optimization is already implemented. 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.

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.*

e) Decomposing the projection matrix
While we can use the estimated projection matrix to project points into the image, we are actually in the more general camera intrinsics and pose. For this we need to decompose P into its different parts. Remember that

$$P = K[R|t] = K[R| - R^T C] = [KR| - KR^T C] = [M| - KR^T C]$$

The left 3×3 submatrix is a product of a upper triangular and an orthogonal matrix. We use QR decomposition to get

$$K^{-1}, R^{-1} = qr((KR)^{-1}) = qr(M^{-1}) \ .$$

We want the intrinsic matrix K to have a positive diagonal (since the focal length is always positive). We therefore set

$$KR = (\hat{K}T)(T^{-1}\hat{R}) = \hat{K}TT^{-1}\hat{R}, \ \ T = diag(sign(diag(\hat{K})))$$

Also, the orthogonal matrix from QR decomposition does not necessarily have determinant 1 (which is a requirement for a proper rotation matrix). If the determinant is -1, we can simply set $R = -R'$ since P is up to scale.

The position of the camera center C is the nullspace of P and can be used to compute the translation t. Report your computed K, R, and t and discuss the reported reprojection errors before and after nonlinear optimization. Do your estimated values seem reasonable?

2.3 Structure from Motion

We now combine reative pose estimation, absolute pose estimation and point triangulation to a simplified Structure from Motion pipeline. The code framework for this is provided in the file `sfm.py`. We provide you again with extracted keypoints and feature

matches. These matches are already geometrically verified and do not contain outliers since we would need extra steps to handle outliers in the pipeline.

Before you start, use the visualization in the framework to verify that the data is loaded correctly and to get an idea of the scene. This can help you judge if your results are correct later on. You can comment out the visualization after this to save time when testing your code.

To initialize the scene we will need to estimate the relative pose between two images. Afterwards we can triangulate the first points in the scene and then iteratively register more images and triangulate new points.

a) Essential matrix estimation
We provide the camera intrinsics matrix K, so we can compute the relative pose between the first images from the essential matrix E. We use DLT as in the calibration exercise to reformulate the constraint

$$\hat{x}_1 E \hat{x}_2 = 0 \ .$$

Remember to lift the keypoints to the normalized image plane first, i.e., $\hat{x} = K^{-1}x$. When you obtained an estimate from DLT you need it to fulfill the essential matrix constraints (i.e., to project it to the manifold of essential matrices). For a proper essential matrix, the first singular values need to be equal (and > 0), and the third one is 0. You can do this by using SVD on your estimate and set the singular values accordingly. Since E is up to scale, you can just set the first singular values to 1.

b) Point triangulation
The DLT for point triangulation is already implemented in the framework. However, some points that fulfill the mathematical constraints lie behind one or both cameras. Since these points are clearly wrong, implement a filtering step in the point triangulation.

c) Finding the correct decomposition
The decomposition of the essential matrix into a relative pose is already implemented. However, this gives four different poses that all fulfill the requirements of the essential matrix. To find the actually correct one, try to triangulate points with each one and use the one with the most points in front of both cameras. *Hint: Set one of the camera poses to identity and be careful with the direction of the transformation.*

d) Absolute pose estimation
Given a point cloud and correspondences to the image keypoints we can estimate the absolute pose of a new image with respect to the scene. This is a simpler version of the calibration task (and actually uses some of the code that you wrote there), so you don't need to implement anything additional here.

e) Map extension
Once we know the new images pose we can extend the map by triangulating new points from all possible pairs of registered images. Implement the triangulation with all pairs and make sure to correctly add the new 2D-3D correspondences to each image.

2.4 Hand-in

The deadline for this assignment is Friday, November 19, 23:59. Write a short report explaining the approaches to calibration and SfM and address the questions mentioned in the tasks. You can add a visualization of your results as well. Upload it (in PDF format) together with your code in a single zip file to moodle.