Computer Vision/Visual Perception
ESIREM Le Creusot & Master in Computer Vision
Universite de Bourgogne, February 2022

Assignment 3 - Camera Calibration with DLT

The goal of this exercise is to understand and implement the Direct Linear Transformation (DLT) method for camera calibration. The DLT method, proposed by Abdel-Aziz and Karara [1], uses a set of control points whose scene and image coordinates are already known. The control points are normally fixed to a rigid frame (calibration frame for you). The problem is essentially to calculate the mapping between the 2D image space coordinates ($\mathbf{p}$) and the 3D object space coordinates ($\mathbf{P}$). For the case of 3D to 2D correspondences, the mapping takes the form of a 3x4 projection matrix ($\mathbf{P}$) such that $\mathbf{p} = \mathbf{MP}$ as discussed during the lectures. In this exercise you will implement the DLT algorithm for calibration of a camera using 2D-3D correspondences. You will also need to simulate a 3D calibration pattern and a camera to test your implementation. Please provide concrete examples/discussions to support your explanations for the questions when suitable.

Part I - Environment Setup

Please create a notebook on Colab or follow these steps to configure your conda environment if you are worning in your local machine:

```
>> conda create -n calibration_dlt
>> conda activate calibration_dlt
>> conda install jupyter
# Go your workspace directory of the course and run jupyter
>> jupyter-notebook
```

We will greatly use Numpy arrays and definitions during this exercise. So do not hesitate to go back in the slides of Class 2 (OpenCV/Python) to check some definitions or in this tutorial on Numpy[1]. There are also some helpful "Cheat Sheets" of basic commands for Numpy and Scipy (Linear algebra in Python)[2].

Part II - Simulating 3D Calibration Pattern and Camera

Please follow the steps below to simulate a 3D calibration pattern observed by a camera, and visualize the 3D points and camera as well as the 2D image projections by plotting them in python as you go along.

1. Generate 3D scene points: There are several ways to make this. You can start by generating just a grid of 3D points on a 3D plane in front of your world coordinate system. You will generate and add more planes for the full calibration pattern later in step 4 once the rest of the steps are clear.
   Tip: Numpy has the function *np.meshgrid* that can be useful.

[1] `https://numpy.org/devdocs/user/quickstart.html`
[2] Scipy: `http://datacamp-community-prod.s3.amazonaws.com/dfdb6d58-e044-4b38-bab3-5de0b825909b`
Numpy: `http://datacamp-community-prod.s3.amazonaws.com/ba1fe95a-8b70-4d2f-95b0-bc954e9071b0`

2. Visualize your generated points with matplotlib in 3D.

3. Generate a camera:

    - Recall the different parameters of the pinhole camera model;
    - You can start by using the intrinsic parameters from the **Exercise 4 (a)** of the Class 2 exercise assignments.
    - Form the projection matrix representing the camera.

4. Image formation: project the 3D scene point(s) to the image plane of the camera using the perspective projection matrix defined in step 2.

    - Are all image points within the image? You can modify the camera parameters to ensure so.

5. Now that you are familiar with simulating a camera and 3D scene points, it is time to generate a calibration pattern and its image projections for the DLT calibration.

    - A conventional 3D calibration pattern consists of three orthogonal planes with regularly spaced points, some possible examples are shown in Figure 1.
    - Recall the minimum number of points you will need for DLT-based camera calibration.

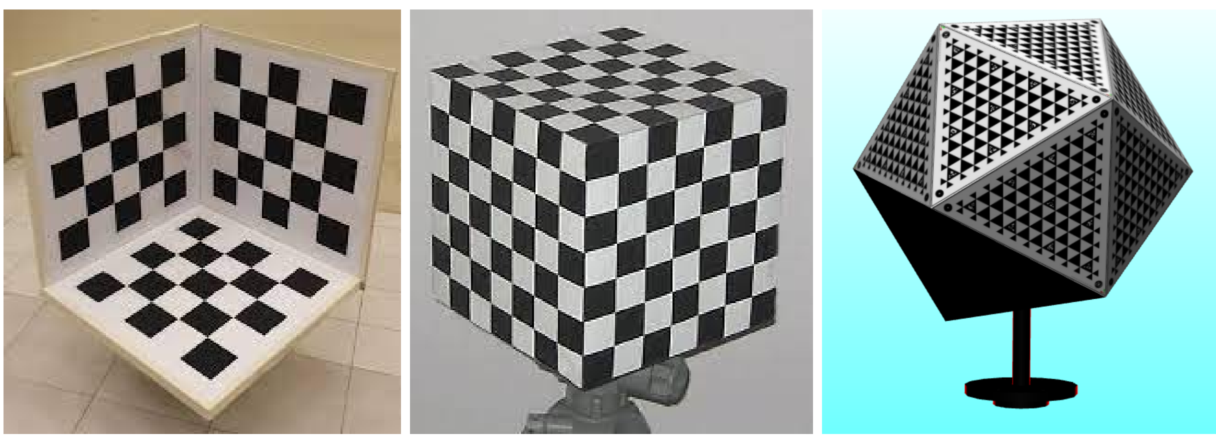

Figure 1: Examples of 3D calibration objects/patterns: corner, cube (similar to corner) and icosahedron.

Part III - Direct Linear Transformation: DLT

Now that you have generated the 3D calibration pattern and have the image projections corresponding to the 3D points, next implement the DLT algorithm to calibrate the simulated camera.

1. Assume all five camera intrinsic parameters are unknown.

2. Implement the DLT algorithm following the notes from Projective Geometry (D. Fofi) and the steps (i) and (iii) shown in Figure 2 (from Hartley & Zisserman [2]).

3. From the projection matrix estimated using DLT, extract the camera intrinsic and extrinsic parameters.

Objective

Given $n \geq 6$ world to image point correspondences $\{\mathbf{X}_i \leftrightarrow \mathbf{x}_i\}$, determine the Maximum Likelihood estimate of the camera projection matrix P, i.e. the P which minimizes $\sum_i d(\mathbf{x}_i, \mathtt{P}\mathbf{X}_i)^2$.

Algorithm

(i) **Linear solution.** Compute an initial estimate of P using a linear method such as algorithm 4.2(*p*109):

(a) **Normalization:** Use a similarity transformation T to normalize the image points, and a second similarity transformation U to normalize the space points. Suppose the normalized image points are $\tilde{\mathbf{x}}_i = \mathtt{T}\mathbf{x}_i$, and the normalized space points are $\tilde{\mathbf{X}}_i = \mathtt{U}\mathbf{X}_i$.

(b) **DLT:** Form the $2n \times 12$ matrix A by stacking the equations (7.2) generated by each correspondence $\tilde{\mathbf{X}}_i \leftrightarrow \tilde{\mathbf{x}}_i$. Write $\mathbf{p}$ for the vector containing the entries of the matrix $\tilde{\mathtt{P}}$. A solution of $\mathtt{A}\mathbf{p} = \mathbf{0}$, subject to $\|\mathbf{p}\| = 1$, is obtained from the unit singular vector of A corresponding to the smallest singular value.

(ii) **Minimize geometric error.** Using the linear estimate as a starting point minimize the geometric error (7.4):

$$\sum_i d(\tilde{\mathbf{x}}_i, \tilde{\mathtt{P}}\tilde{\mathbf{X}}_i)^2$$

over $\tilde{\mathtt{P}}$, using an iterative algorithm such as Levenberg–Marquardt.

(iii) **Denormalization.** The camera matrix for the original (unnormalized) coordinates is obtained from $\tilde{\mathtt{P}}$ as

$$\mathtt{P} = \mathtt{T}^{-1}\tilde{\mathtt{P}}\mathtt{U}.$$

Algorithm 7.1. *The Gold Standard algorithm for estimating* P *from world to image point correspondences in the case that the world points are very accurately known.*

Figure 2: DLT algorithm recipe steps from Chapter 7 of Hartley & Zisserman [2].

4. Compare your results with the ground truth, if they do not match, there's probably a bug (or numerical issues).

5. Add noise on the image points - typically the detected 2D points are noisy (real life, believe it or not, is not a perfect simulation). Test the behavior of the algorithm with increasing noise.

   - You can model the noise on the 2D points as a zero-mean Gaussian distribution and test for increasing values of the standard deviation (for example, 0.25, 0.5, 0.75, 1 pixel and so on).

6. How would you evaluate the performance of the algorithm? Which error measures you can compute?

Part IV: (Bonus Questions) Sensitivity & Conditioning

- Why the considering 3D points should not belong to a single planar surface? (justify looking at the system of equations when the points belong to a place e.g. $Z = k$).

- Please make different configurations (distances of the pattern to the camera and angle between the planes of the pattern, number of point correspondences) and evaluate the results obtained results.
- Increasing the number of points results in better accuracy in the estimation? Make the test without the coordinate normalization step and comment the obtained results.

Submission

Please return a concise PDF report explaining your reasoning and equation developments. You should also submit your python notebook (commented) with the corresponding implementations, together inside a [name]_list_3.zip file (replacing [name] with your name ;-)). The submission should be done via the teams channel of the course using teams assignment.
Deadline: 11/02/2022 at 23:59pm.

Notes:

- Plagiarism copying the work from another source (student, internet, etc.) will be awarded a 0 mark. In case there are multiple submissions with the same work, each one will receive a 0.
- **Credits:** This assignment was extensively based on exercises kindly provided by Devesh Adlakha.

References

[1] Abdel-Aziz, Y.I., & Karara, H.M.. "Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry". Proceedings of the Symposium on Close-Range Photogrammetry, 1971.

[2] Hartley, Richard, and Andrew Zisserman. "Multiple view geometry in computer vision." (2003). Chapter 7 (7.1)

[3] Fusiello, Andrea. "A matter of notation: Several uses of the Kronecker product in 3D computer vision." Pattern Recognit. Lett. 28 (2007): 2127-2132.