Assignment03-2025

February 7, 2025

In this homework, we will program from scratch the linear calibration algorithm we have seen in the course, in the first part of the camera calibration session.

1. **[1pt]** Write a code to generate the images $\mathbf{p}_k^i$ of the vertices $\mathbf{p}_k^m$ of **two cubes in the 3D space (16 points)**, where k is the index of the vertice. You may choose any combination of intrinsic/extrinsic parameters $\mathbf{K}$, $\mathbf{R}$, $\mathbf{t}$. The only constraint is that **we want the 16 points to fit in the image** (tune the parameters so that it is the case).

2. **[1pt]** Use a random generator (see example below) to add a **perturbation** to the coordinates of all the **projected points**. This will simulate the noise in the detection process. We will initially use a normal distribution with $\sigma = 1.0$.

[3]:
```
import numpy as np
m, sigma = 0.0, 1.0
samples = np.random.normal(m, sigma, 32)
print(samples)
```

```
[-0.94925475  1.07124176  0.24855815  0.83956255 -0.10045812  0.78351213
  2.16589818 -0.03636466 -0.34650842  1.77882959 -0.10455337  1.53174065
 -0.00786807  0.33130893 -0.55080361  0.74114442  1.37965138 -1.04885226
  1.21399729  1.9199281  -0.0225208  -1.82064203  0.35074009  0.2578335
 -0.55775224 -0.531642    1.26838593  2.12207699  0.05880186 -0.83657963
 -0.33871929 -0.07695651]
```

3. **[1.5pts]** Use the 16 correspondences $(\mathbf{p}_k^i, \mathbf{p}_k^m)$ and the DLT algorithm seen in the class to obtain an estimate $\mathbf{P}^*$ of the projection matrix. This will imply constructing the data matrix $\mathbf{D}$ and apply an eigen-decomposition of $\mathbf{D}^\top\mathbf{D}$. You can use the version of the eigendecomposition present in numpy, an example is given below.

[5]:
```
a =  np.random.randn(12, 12)
evalues, evectors = np.linalg.eig(a)
print(evalues)
```

```
[ 4.23687578+0.j         -2.65484742+1.89341037j -2.65484742-1.89341037j
  0.80631998+2.33205632j  0.80631998-2.33205632j  1.32589474+0.36766811j
  1.32589474-0.36766811j -0.66142846+2.10510303j -0.66142846-2.10510303j
 -0.27128018+0.j         -1.19086334+0.j         -2.33237124+0.j        ]
```

4. **[1pt]** Deduce estimates for the intrinsic/extrinsic parameters $\hat{\mathbf{K}}$, $\hat{\mathbf{R}}$, $\hat{\mathbf{t}}$. Explain how you have raised the ambiguity over the extrinsic parameters.

5. **[1pt]** Write a function that computes the **average reprojection error** $D(\psi)$ of your points (i.e. the average distance between their detected position and their projection in pixels

$$D(\psi) = \sqrt{\frac{1}{16}\sum_{k=1}^{16}(u_k^i - \hat{u}^i(\psi, \mathbf{p}_k^m))^2 + (v_k^i - \hat{v}^i(\psi, \mathbf{p}_k^m))^2}$$

by a candidate projection matrix parametrized by 11 parameters, ψ.

6. **[1.5pt]** Study experimentally the influence of the number of points used for the estimation in question 3 on the obtained average reprojection error: To do so, repeat the estimation done before with random subsets of the points with increasing cardinal and evaluate $D(\psi)$ in each case. You can use several subsets for one cardinal value and give the averaged average reprojection error.

7. **[1pt]** Study experimentally the influence of the noise level (σ) over the obtained reprojection error.

8. **[2pt]** Use the least_squares function from the scipy.optimize package to implement a non-linear least-square parameters refinement, over the intrinsic and extrinsic parameters, starting from the estimates from question 4.

https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html