LUT Engineering Science
Data-Centric Engineering
Tuomas Eerola, Joona Kareinen, Samuel Repka
Spring term 2025

BM40A0902 3D Computer Vision
Exercise 7
Camera calibration.

1. Estimating the projection matrix (3 point).

   Consider a 3-D cube with one corner at the origin of the world coordinate system, and an edge length of 5 cm. The cube was projected onto a 2D image. The goal of this task is to estimate the projection matrix when given 3D points and their projections.

   (a) Load data from `cube_points.mat`. Coordinates of cube corners in 3D are given as 3×8 matrix `points3d`. The respective projected points are given in 2×8 matrix `points2d`. Plot original 3D points on the first plot, and then plot projected 2D points on the second plot. You can also use `connecting_indices` vector to connect the points into a cube.

   (b) Implement function `calibrate(points3d, points2d)` that performs direct-linear-transform (DLT) and finds a suitable projection matrix that would project `points3d` to `points2d`. Apply that function to the given points to find projection matrix `M`.

   (c) Project given 3D points using found matrix `M`. Plot them along with given 2D projected points. Are they close?
   Calculate reprojection error

   $$E(p, \hat{p}) = \frac{1}{N} \sum_{i=1}^{N} ||p_i - \hat{p}_i||_2, \tag{1}$$

   where p are the given 2D points, $\hat{p}$ are the projected points using matrix `M` and $||p_i - \hat{p}_i||_2$ is the Euclidean distance between them.

   (d) Enhance your code by adding normalization. Either write a separate function for calibration with normalization, or add it as an option to already implemented `calibrate` function. The new function must find the projection matrix of the perspective projection with point normalization. Remember to perform both normalization before the DLT and then denormalization afterwards!

   (e) In real-life scenario, exact coordinates are quite difficult to acquire, the locations are not exact and noisy. Compare results with and without normalization using data from `cube_points_noisy.mat`. Are they improved with normalization? Are the new reprojected points visually closer to the given 2D points? Is reprojection error smaller?

2. Decomposition of projection matrix (1 point).

   In this task the projection matrix obtained in the previous task will be decomposed.

   Write a function `decompose_projection(M)` which decomposes the projection matrix into intrinsic part K, rotation R, and camera location C. Check your result by calculating the projection matrix from the parts, i.e., $M = KR[I \ \ -C]$ and checking that this corresponds to the projection matrix that was used as the input.

Decompose projection matrix that you have found in Task 1. Plot camera extrinsic properties (rotation and position) using `plot_frame` function along with given 3D points.

Note: RQ decomposition for MATLAB is provided in the template file. For Python, you can use `rq` from `scipy` (`from scipy.linalg import rq`).