CS231A: Computer Vision, From 3D Reconstruction to Recognition Homework #1 Solution

(Spring 2025)

Name: Van-Thinh Vo **Email**: thinh.vovan@hcmut.edu.vn

1 Projective Geometry Problems [20 points]

In this question, we will examine properties of projective transformations. We define a camera coordinate system, which is only rotated and translated from a world coordinate system (with a rigid transform).

(a) **Prove that parallel lines in the world reference system are still parallel in the camera reference system** (note: this is an example of a rigid transformation). **[4 points]** **Solution**: The rigid transformation remains the length, shape, direction and relative position (a chain of rotations, translations, reflections in any order). Therefore, parallel lines in the world reference system are still parallel in the camera reference system. Indeed, supposed that we have two parralel lines $d_1 = \begin{cases} x = a_1 + u_1 t \\ y = a_2 + u_2 t \\ z = a_3 + u_3 t \end{cases}$ and $d_2 = \begin{cases} x = b_1 + u_1 t' \\ y = b_2 + u_2 t' \\ z = b_3 + u_3 t' \end{cases}$ with the same direction vector $\vec{\mathbf{u}}$. Let $A \in d_1$ and $B \in d_2$ with parameter t and t'. After applying rigid transformation to two points, we see that they also be on the two parralel lines.

(b) Now let us consider affine transformations, which are any transformations that preserve parallelism. Affine transformations include not only rotations and translations, but also scaling and shearing. Given some vector p, an affine transformation is defined as

$$A(p) = Mp + b$$

where M is an invertible matrix. **Prove that under any affine transformation, the ratio of parallel line segments is invariant, but the ratio of non-parallel line segments is not invariant. [6 points]. Solution**: Supposed that we have four distint points A, B, C and D. Let $\vec{a} = \overrightarrow{AB}$ and $\vec{b} = \overrightarrow{CD}$.

- Case 1 : $k\vec{a} = \vec{b} (k \neq 0)$, then

$$\begin{cases} \vec{a'} = \mathbf{A}(B) - \mathbf{A}(A) = \mathbf{M}(B - A) = M\vec{a} \\ \vec{b'} = \mathbf{A}(D) - \mathbf{A}(C) = \mathbf{M}(D - C) = \mathbf{M}\vec{b} = k\mathbf{M}\vec{a} \end{cases} \Rightarrow \frac{\|\vec{a}\|}{\|\vec{b}\|} \text{is preserved} \quad (1)$$

- Case 2: $\nexists k \neq 0, k\vec{a} = \vec{b}$. Because scaling and shearing effects depend on the vector direction, the ratio of non-parralel line segments is not invariant. For instance, let $\vec{a} = (1, 0)$ and $\vec{b} = (0, 1)$ and apply scale transformation with $\mathbf{M} = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{bmatrix}$. Obviously, this transformation has no change in $\vec{b}$ while it normalized $\|\vec{a}\|$ by $\frac{1}{2}$.

(c) Consider a unit square $pqrs$ in the world reference system where p, q, r, and s are points. **Will the same square in the camera reference system always have unit area? Prove or provide a counterexample. Similarly, will the unit square be preserved**

or not under an affine transformation? [4 points]. Solution: Dicussed above, rigid transformation preserves direction, length, etc. Therefore, the square area also is identical with origin. With affine transformation, we can prove that $S' = |det(M)| \times S$, then it will change.

(d) You have explored whether these three properties (rotational and translational invariance, and ratio of parallel line segments) hold for affine transformations. Do these properties hold under any projective transformation? **Justify briefly in one or two sentences** (no proof needed). **[6 points]. Solution**: These hold under affine model because it holds parralel, rotational and translation invariance.

2 Affine Camera Calibration (35 points)

(a) Given correspondences for the calibrating grid, solve for the camera parameters using Eq. 2. Note that each measurement $(x_i, y_i) \leftrightarrow (X_i, Y_i, Z_i)$ yields two linear equations for the 8 unknown camera parameters. Given N corner measurements, we have $2N$ equations and 8 unknowns. Using the given corner correspondences as inputs, complete the method `compute_camera_matrix()`. You will construct a linear system of equations and solve for the camera parameters to minimize the least-squares error. After doing so, you will return the 3×4 affine camera matrix composed of these computed camera parameters. **Explain your approach and include the camera matrix that you compute in the written report. [15 points for code + 5 for write-up] Solution**: I built up $\mathbf{M}_{2,4}$ for camera matrix, $\mathbf{P}_{24,4}$ for homogeneous coordinate in real world and lastly, $\mathbf{p}_{24,2}$ for points in image plane. Afterward, using `numpy.linalg.lstsq` to approximate $\mathbf{M}$ as below:

$$\begin{bmatrix} 0.531276507 & -0.0180886074 & 0.120509667 & 129.720641 \\ 0.0484975447 & 0.536366401 & -0.102675222 & 44.3879607 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(b) After finding the calibrated camera matrix, you will compute the RMS error between the given N image corner coordinates and N corresponding calculated corner locations in `rms_error()`. Recall that

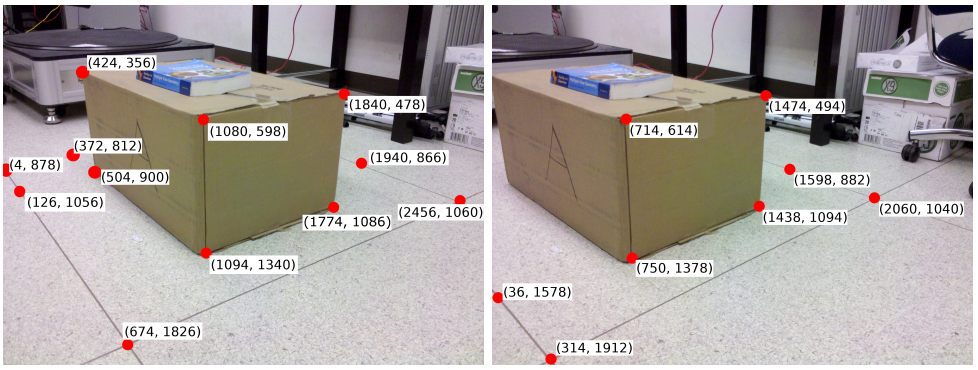
$$\text{RMS}_{\text{total}} = \sqrt{\sum((x - x')^2 + (y - y')^2)/N}$$

Compute the RMS error for the camera matrix that you found in part (a). **What are some possible sources of error when computing the error between the given corners and calculated points using the camera matrix. [5 points for code + 5 points for write-up] Answer**: Errors come from the noisy, radial distortion and the affine camera (low accuracy).

(c) Could you calibrate the matrix with only one checkerboard image? **Explain briefly in one or two sentences. [5 points] Answer**: Absolutely because whether to solve or not depends on the number of chosen points.

3 Single View Geometry (45 points)

In this question, we will estimate camera parameters from a single view and leverage the projective nature of cameras to find both the camera center and focal length from vanishing points present in the scene above.



(a) Image 1 (1.jpg) with marked pixels (b) Image 2 (2.jpg) with marked pixels

Figure 1: Marked pixels in images taken from different viewpoints.

(a) In Figure 2, we have identified a set of pixels to compute vanishing points in each image. Please complete `compute_vanishing_point()`, which takes in these two pairs of points on parallel lines to find the vanishing point. You can assume that the camera has zero skew and square pixels, with no distortion. **[5 points]**

(b) Using three vanishing points, we can compute the intrinsic camera matrix used to take the image. Do so in `compute_K_from_vanishing_points()`. **[10 points]**

(c) Is it possible to compute the camera intrinsic matrix for any set of vanishing points? Similarly, is three vanishing points the minimum required to compute the intrinsic camera matrix? **Justify your answer. [5 points] Answer**: We can compute K with any set of vanishing points greater or equal than the number of parameters in ω only if we know cosin each pair of vanishing points. Therefore, having three mutually orthogonal vanishing points is the minimum and best choice.

(d) The method used to obtain vanishing points is approximate and prone to noise. **Discuss what possible sources of noise could be, and select a pair of points from the image that are a good example of this source of error. [5 points] Answer**:

- Manually set up leading to noise.
- Radial distortion (with point near the boundary).
- Pixel can not actually represent the corner and edge.
- For example, $(126, 10256)$ and $(4, 878)$ in image 1.

(e) This process gives the camera internal matrix under the specified constraints. For the remainder of the computations, use the following internal camera matrix:

$$K = \begin{bmatrix} 2448 & 0 & 1253 \\ 0 & 2438 & 986 \\ 0 & 0 & 1 \end{bmatrix}$$

Use the vanishing lines we provide in the starter code to verify numerically that the ground plane is orthogonal to the plane front face of the box in the image. Fill out the method `compute_angle_between_planes()` and **include a brief description of your solution and your computed angle in your report. [8 points for code + 2 points for write-up] Answer**: I followed p1_review.

(f) Assume the camera rotates but no translation takes place. Assume the internal camera parameters remain unchanged. An Image 2 of the same scene is taken. Use vanishing points to estimate the rotation matrix between when the camera took Image 1 and Image 2. Fill out the method `compute_rotation_matrix_between_cameras()` and **submit a brief description of your approach and your results in the written report. [8 points for code + 2 points for write-up] Answer**: I followed instruction.