

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

(Spring 2025)

Due: Friday, April 18

Submitting

Please put together a PDF with your answers for each problem, and submit it to the appropriate assignment on Gradescope. We recommend you to add these answers to the latex template files on our website, but you can also create a PDF in any other way you prefer.

The assignment will require you to complete the provided python files and create a PDF for written answers. The instructions are **bolded** for parts of the problem set you should respond to with written answers. To create your PDF for the written answers, we recommend you add your answers to the latex template files on our website, but you can also create a PDF in any other way you prefer. To implement your code, make sure you modify the provided ".py" files in the code folder.

For the written report, in the case of problems that just involve implementing code, you will only need to include the final output (where it is requested) and in some cases a brief description if requested in the problem. There will be an additional coding assignment on Gradescope that has an autograder that is there to help you double check your code. Make sure you use the provided ".py" files to write your Python code. Submit to both the PDF and code assignment, as we will be grading the PDF submissions and using the coding assignment to check your code if needed.

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]
- (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]

- (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]

- (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]

2 Affine Camera Calibration (35 points)

In this question, we will perform affine camera calibration using two different images of a calibration grid. First, you will find correspondences between the corners of the calibration grids and the 3D scene coordinates. Next, you will solve for the camera parameters.

It was shown in class that a perspective camera can be modeled using a 3×4 matrix:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (1)$$

which means that the image at point (X, Y, Z) in the scene has pixel coordinates $(x/w, y/w)$. The 3×4 matrix can be factorized into intrinsic and extrinsic parameters.

An *affine* camera is a special case of this model in which rays joining a point in the scene to its projection on the image plane are parallel. Examples of affine cameras include orthographic projection and weakly perspective projection. An affine camera can be modeled as:

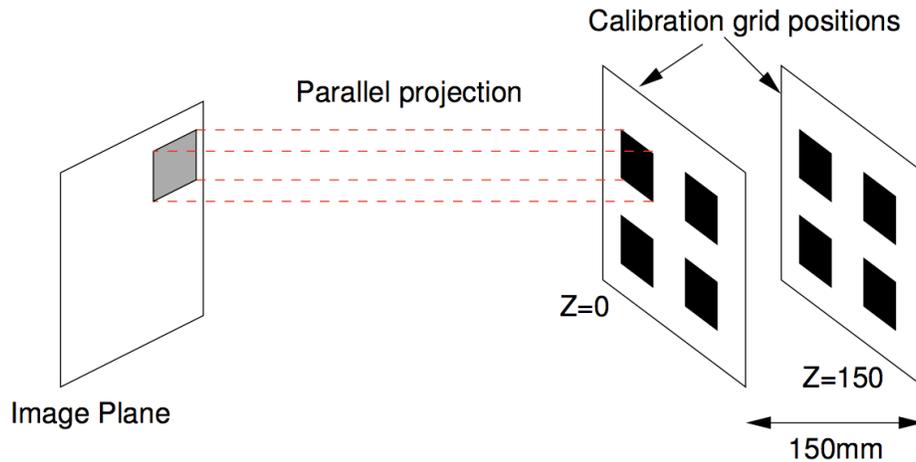
$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (2)$$

which gives the relation between a scene point (X, Y, Z) and its image (x, y) . The difference is that the bottom row of the matrix is $[0 \ 0 \ 0 \ 1]$, so there are fewer parameters we need to calibrate. More importantly, there is no division required (the homogeneous coordinate is 1) which means this is a *linear model*. This makes the affine model much simpler to work with mathematically - at the cost of losing some accuracy. The affine model is used as an approximation of the perspective model when the loss of accuracy can be tolerated, or to reduce the number of parameters being modeled. Calibration of an affine camera involves estimating the 8 unknown entries of the matrix in Eq. 2 (This matrix can also be factorized into intrinsics and extrinsics, but that is outside the scope of this homework). Factorization is accomplished by having the camera observe a calibration pattern with easy-to-detect corners.

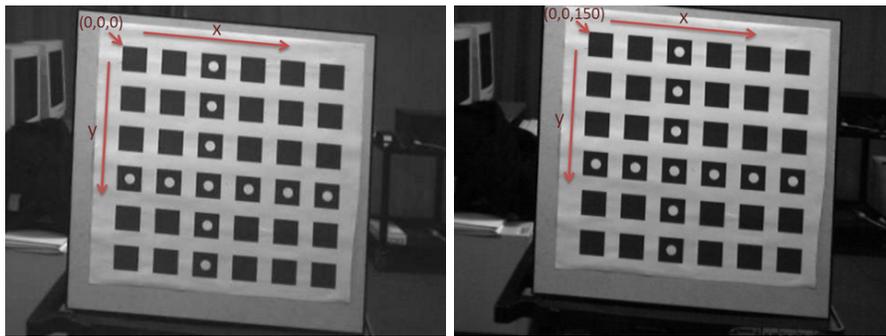
Scene Coordinate System

The calibration pattern used is shown in Figure 1, which has a 6×6 grid of squares. Each square is $50mm \times 50mm$. The separation between adjacent squares is $30mm$, so the entire grid is $450mm \times 450mm$. For calibration, images of the pattern at two different positions were captured. These images are shown in Fig. 1 and can be downloaded from the course website. For the second image, the calibration pattern has been moved back (along its normal) from the rest position by $150mm$.

We will choose the origin of our 3D coordinate system to be the top left corner of the calibration pattern in the rest position. The X -axis runs left to right parallel to the rows of squares. The Y -axis runs top to bottom parallel to the columns of squares. We will work in units of millimeters. All the square corners from the first position corresponds to $Z = 0$. The second position of the calibration



(a) Image formation in an affine camera. Points are projected via parallel rays onto the image plane



(b) Image of calibration grid at $Z=0$ (c) Image of calibration grid at $Z=150$

Figure 1: Affine camera: image formation and calibration images.

grid corresponds to $Z = 150$. The top left corner in the first image has 3D scene coordinates $(0, 0, 0)$ and the bottom right corner in the second image has 3D scene coordinates $(450, 450, 150)$. This scene coordinate system is labeled in Fig. 1.

- (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]
- (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

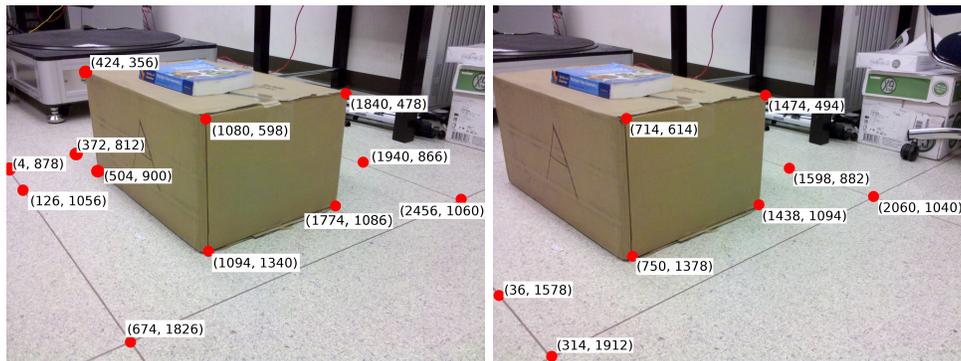
$$\text{RMS}_{\text{total}} = \sqrt{\sum ((x - x')^2 + (y - y')^2) / N}$$

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]

- (c) Could you calibrate the matrix with only one checkerboard image? **Explain briefly in one or two sentences.** [5 points]

3 Single View Geometry (45 points)



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

Figure 2: Marked pixels in images taken from different viewpoints.

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) 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]
- (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]
- (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]

- (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]