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

(Winter 2022)

Vasu G. Patel: vgpatel1@stanford.edu

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.

(a) Prove that parallel lines in the world reference system are still parallel in the camera reference system. [**4 points**]

Answer 1(a): Let k, l be parallel lines in world reference system. k1, k2 points lie on line k and l1, l2 points lie on line l. From the property of parallel lines: `k X l = 0`
⇒ `(k2 - k1) X (l2 - l1) = 0`

Applying Translation (t) to line k and l
⇒ `(k2 + t - k1 - t) X (l2 + t - l1 - t)` ⇒ `(k2 - k1) X (l2 - l1) = 0`

Similarly, applying Rotation (R) to k and l
⇒ `(Rk2 - Rk1) X (Rl2 - Rl1)` ⇒ `R(k2 - k1) X R(l2 - l1) = 0`

Hence, it is proved that applying rotation and translation (Isometric transformation), the lines remain parallel.

(b) 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. [**4 points**]

Answer 1(b): Transformation from world reference system to camera reference system only involves Rotation (R) and Translation (t) which is similar to isometries transformation. Since, in Isometric transformation preserve parallel lines and angles, $pqrs$ when projected to $p\prime q\prime r\prime s\prime$ will have unit area in the projected plane.

(c) Now let's 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**]

Answer 1(c): Let line k || l. Points k1, k2, l1, l2 lie on line k & respectively, such that ratio of line segment `k:l = m` (constant).
⇒ `(k2 - k1) = m (l2 - l1)`

Applying affine transformation to line 'k':
`A(k2 - k1) = M(k2 - k1) + b` $\Rightarrow$ `M(m(l2 - l1)) + b` $\Rightarrow$ `m * M(l2 - l1) + b`
`A(k2 - k1) = m * (M(l2 - l1) + c)`, where c = b/m
`A(k2 - k1) = m * A(l2 - l1)`
Hence, the ratio of parallel line segment is invariant while applying affine transformation.

To its contrary, for non-parallel lines this won't be true as affine tranformation preserves parallelism.

(d) You have explored whether these three properties hold for affine transformations. Do these properties hold under any projective transformation? Justify briefly in one or two sentences (no proof needed). **[6 points]**

Answer 1(d): Projective transformation does not not preserve parallelism and henceforth project shapes will not have same area as in real world. In addition, the ratio of parallel line segments will not remain invariant when project into image plane.

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:

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

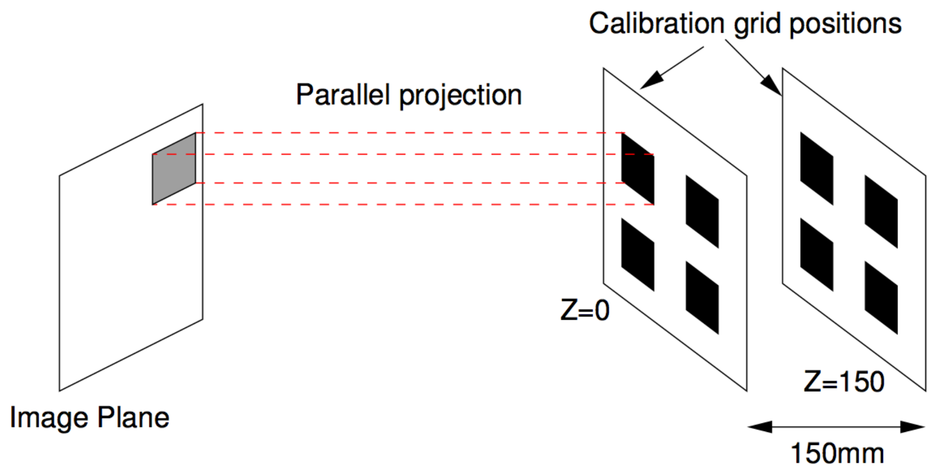
which means that the image at point (X, Y, Z) in the scene has pixel coordinates $(x/w, y/w)$. The 3×4 matrix can be factorized into intrinsic and extrinsic parameters.

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:

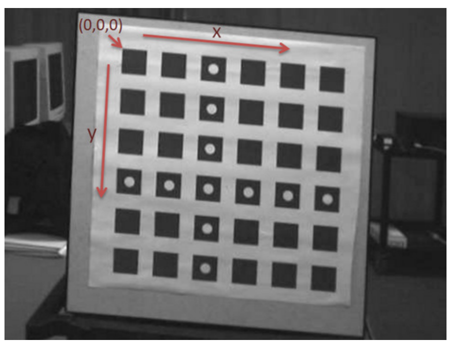
$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2}$$

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.

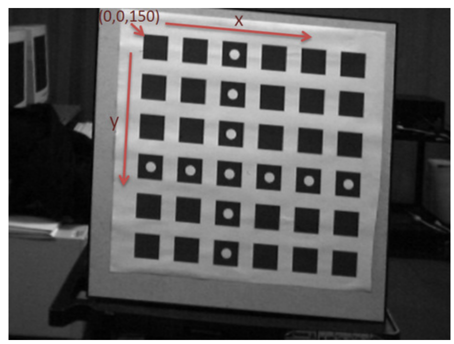
Scene Coordinate System



(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.

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 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. In your written report, submit your code as well as the camera matrix that you compute. [**15 points**]

Answer 2(a):

$$\begin{bmatrix} 5.31276507e-01 & -1.80886074e-02 & 1.20509667e-01 & 1.29720641e+02 \\ 4.84975447e-02 & 5.36366401e-01 & -1.02675222e-01 & 4.43879607e+01 \\ 0.00000000e+00 & 0.00000000e+00 & 0.00000000e+00 & 1.00000000e+00 \end{bmatrix}$$

(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

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

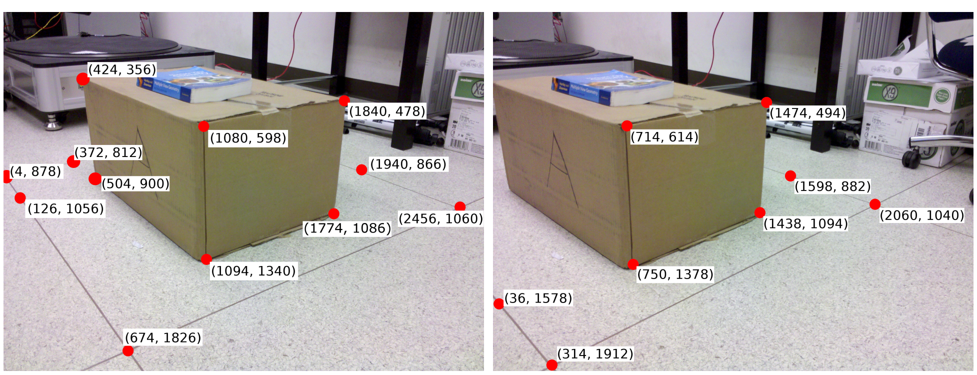
Please submit your code and the RMS error for the camera matrix that you found in part (a). [**15 points**]

Answer 2(b):
RMS Error: 0.993830483279845

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

Answer 2(c):
No it is not possible to calibrate a camera using one checker board results into degenerate solutions.

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**]

Answer 3(a): Computed set of vanishing points are:
v1: [6517.222176639512, -685.7329895094649]

v2: [-721.3165273806906, -134.57249285243017]
v3: [1190.6088650754994, 6460.269849001461]
v1b: [4400.303020985471, -128.81113075563073]
v2b: [-1395.373604870271, -141.70785621104508]
v3b: [1045.1436950146629, 7641.605083088954]

(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]**

Answer 3(b): Computed intrinsic camera matrix (K):

$$\begin{bmatrix} 2.59416989e+03 & 0.00000000e+00 & 7.73289532e+02 \\ 0.00000000e+00 & 2.59416989e+03 & 9.79503297e+02 \\ 0.00000000e+00 & 0.00000000e+00 & 1.00000000e+00 \end{bmatrix}$$

(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 3(c): We won't be able to obtain camera matrix for degenerate cases like:
1. When two or more vanishing points are co-linear along the line of sight, it would be a degenerate case and we won't be able to get K matrix.
2. If the points are lying on non-orthogonal planes, it won't be mathematically possible to retrieve K matrix.

(d) The method used to obtain vanishing points is approximate and prone to noise. Discuss approaches to refine this process. **[5 points]**

Answer 3(d): Process of obtaining vanishing points can be refined using following techniques:
1. Using multiple parallel lines would help generate several vanishing points (since, one vanishing point is obtained from every pair of parallel line) which are fairly close to one another. Averaging those vanishing points would help reduce the noise as it gets closer to true (x,y) coordinates of vanishing point.
2. By using corner detection detection algorithm and retrieving the coordinates of those detected corner would be more accurate than manually identifying the corner and coordinates

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

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

Identify a sufficient set of vanishing lines on the ground plane and the plane on which the letter A exists, written on the side of the cardboard box, (plane-A). Use these vanishing lines to verify numerically that the ground plane is orthogonal to the plane-A. Fill out the method `compute_angle_between_planes()` and submit your code and the computed angle. **[10 points]**

Answer 3(e):
Angle between floor and box: 90.027361241031 degrees

(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 your code and your results. [**10 points**]

Answer 3(f): Rotation between two cameras:

$$\begin{bmatrix} 0.96154157 & 0.04924778 & -0.15783349 \\ -0.01044314 & 1.00703585 & 0.04571333 \\ 0.18940319 & -0.06891607 & 1.00470583 \end{bmatrix}$$

Angle around z-axis (pointing out of camera): -2.931986 degrees
Angle around y-axis (pointing vertically): -8.918793 degrees
Angle around x-axis (pointing horizontally): -2.605117 degrees