Problem Set 4: Geometry

Description

In this project you will explore *calibrating* a camera with respect to 3D world coordinates as well as estimating the relationship between two camera views.

What to submit

Download and unzip: ps4.zip

`ps4/`

- input/ - input images, videos or other data supplied with the problem set
- output/ - directory containing output images and other files your code generates
- ps4.py - code for completing each part, esp. function calls
- *.py - Python modules, any utility code
- ps4_report.pdf - a PDF file with all output images and text responses

Zip it back up as `ps4.zip`, and submit on T-Square.

Guidelines

1. Include all the required images in the report to avoid penalty.
2. Include all the textual responses, outputs and data structure values (if asked) in the report.
3. Make sure you submit the correct (and working) version of the code.
4. Include your name and GTID in the report.
5. Even if the code is not working, submit the code as the instructors can read through the algorithms to give partial credit. Comment your code appropriately.
6. Late submissions should be emailed to the TAs to be graded for partial credit.

Questions

1. **Calibration**

   The files pts2d-pic_a.txt and pts3d.txt both contain a list of twenty 2D and 3D points of the image pic_a.jpg. The goal is to compute the projection matrix that goes from world 3D coordinates to 2D image coordinates. Recall that, using homogeneous coordinates, the equation is:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \simeq \begin{bmatrix} s*u \\ s*v \\ s \end{bmatrix} = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & m_{1,4} \\ m_{2,1} & m_{2,2} & m_{2,3} & m_{2,4} \\ m_{3,1} & m_{3,2} & m_{3,3} & m_{3,4} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

   Recall: You solve for the 3x4 matrix M using either SVD to solve the homogeneous version of the equations or by setting $m_{3,4}$ to 1 and then using the normal Least-Squares method. Remember that M is only known up to a scale factor.

To make sure that your code is correct, we are going to give you a set of "normalized points" in the files pts2d-norm-pic_a.txt and pts3d-norm.txt. If you solve for M using all the points you should get a matrix that is a scaled equivalent of the following:

$$M_{\text{normA}} = \begin{bmatrix} -0.4583 & 0.2947 & 0.0139 & -0.0040 \\ 0.0509 & 0.0546 & 0.5410 & 0.0524 \\ -0.1090 & -0.1784 & 0.0443 & -0.5968 \end{bmatrix}$$

For example, this matrix will take the last normalized 3D point, which is < 1.2323, 1.4421, 0.4506, 1.0 > and will project it to the < u, v > of < 0.1419, −0.4518 > where we converted the homogeneous 2D point $< u_s, v_s, s >$ to its inhomogeneous version by dividing by s (i.e., the real transformed pixel in the image).

a. Given the normalized 2D and 3D lists, namely pts2d-norm-pic_a.txt and pts3d-norm.txt, write a function **`solve_least_squares(pts3d, pts2d)`** that solves for the 3x4 transformation matrix M_{normA} using the Least Squares method. It should return the M matrix as well as an error metric (sum of squared residual errors of all points).

   Test it on the normalized 3D points by multiplying those points by your M matrix and comparing the resulting normalized 2D points to the normalized 2D points given in the file. Write a function **`project_points(pts3d, M)`** that transforms each 3D point to 2D. Remember to divide by the homogeneous value to get an inhomogeneous point.

   You can do the comparison by checking the *residual* between the predicted location of each test point using your equation and the actual location given by the 2D input data. Compute residuals for each point by writing a function **`get_residuals(pts2d, pts2d_projected)`**. The residual is the L2 distance (square root of the sum of squared differences) in u and v.

   Functions:
   - `solve_least_squares(pts3d, pts2d) -> M, error`
   - `project_points(pts3d, M) -> pts2d_projected`
   - `get_residuals(pts2d, pts2d_projected) -> residuals`

   Output:
   - The matrix M you recovered from the normalized points (3x4) [text response]
   - The < u, v > projection of the last point given your M matrix [text response]
   - The residual between that projected location and the actual one given [text response]

b. Now you are ready to calibrate the camera. Using the 3D and 2D point lists for the image, we're going to compute the camera projection matrix. To understand the effects of over constraining the system, you're going to try using sets of 8, 12, and 16 points and then look at the residual errors. To debug your code, you can use the normalized set from above, but for the actual question, you'll need to use: pts2d-pic_b.txt and pts3d.txt

For the three point set sizes k of 8, 12, and 16, repeat 10 times, respectively:
- Randomly choose k points from the 2D list and their corresponding points in the 3D list.
- Compute the projection matrix M on the chosen points.
- Pick 4 points **not** in your set of k, and compute the average residual.
- Return the M that gives the lowest residual.

Function: `calibrate_camera(pts3d, pts2d)`
Output:
- Average residual for each trial of each k (10x3) [text response]
- Explain any difference you see between the results for the different k's [text response]
- The best M matrix (3x4) [text response]

c. Finally we can solve for the camera center in the world. Let us define M as being made up of a 3x3 matrix that we'll call Q and a 4th column we'll call m_4:

$$M = [\mathbf{Q}|\mathbf{m_4}]$$

From class we said that the center of the camera C could be found by:

$$C = -Q^{-1}\mathbf{m_4}$$

To debug your code: If you use you the normalized 3D points to get the M_{normA} given above, you would get a camera center of:

$$C_{\text{normA}} =< -1.5125, -2.3515, 0.2826 >$$

Given the best M from the last part, compute C.

Output:
- The location of the camera in real 3D world coordinates [text response]

2. Fundamental Matrix Estimation

We now wish to estimate the mapping of points in one image to lines in another by means of the fundamental matrix. This will require you to use similar methods to those in question 1. We will make use of the corresponding point locations listed in pts2d-pic_a.txt and pts2d-pic_b.txt.

Recall that the definition of the Fundamental Matrix is:

$$\begin{bmatrix} u' & v' & 1 \end{bmatrix} \begin{bmatrix} f_{1,1} & f_{1,2} & f_{1,3} \\ f_{2,1} & f_{2,2} & f_{2,3} \\ f_{3,1} & f_{3,2} & f_{3,3} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = 0$$

Given corresponding points you get one equation per point pair. With 8 or more points you can solve this. With more points (such as the 20 in the files) you solve using the the same least squares method as

in question 1 above.

a. Create the Least Squares function that will solve for the 3x3 matrix $\tilde{F}$ that satisfies the epipolar constraints defined by the sets of corresponding points. Solve this function to create your Least Squares estimate of the 3x3 transform $\tilde{F}$.

Function: `compute_fundamental_matrix(pts2d_a, pts2d_b) -> F`
Output:
- The matrix $\tilde{F}$ generated from your Least Squares function [text response]

b. The linear squares estimate of $\tilde{F}$ is full rank; however, the fundamental matrix is a rank 2 matrix. As such, we must reduce its rank. In order to do this, we can decompose $\tilde{F}$ using Singular Value Decomposition into the matrices $U\Sigma V^T = \tilde{F}$. We can then estimate a rank 2 matrix by setting the smallest singular value in Σ to zero, thus generating Σ'. The fundamental matrix is then easily calculated as $F = U\Sigma' V^T$. Use the SVD function to do, well, the SVD.

Output:
- Fundamental matrix F [text response]

c. Now you can use your matrix F to estimate an epipolar line l_b in Image 'B' corresponding to point p_a in Image 'A': $l_b = F p_a$

Similarly, epipolar lines in Image A corresponding to points in Image B are related by the transpose of F.

Draw these epipolar lines. Below is one way; the key is to be able to take the projective geometry form of the line and draw it in the image:
- The resulting lines l_i defined in homogeneous coordinates can not be drawn directly using standard line functions, which take two points as input. In order to use such functions, we can find the intersection of a given line l_i with the image boundaries.
- If we define l_L to be the line corresponding to the left hand side of the image and l_R to be the line corresponding to the right hand side of the image, we can find the points $P_{i,L} = l_i \times l_L$ and $P_{i,R} = l_i \times l_R$. We can now plot the line running through the points $P_{i,L}$, $P_{i,R}$.
- However, we must first have the equations for l_L and l_R. Making use of the point-line duality, we know, e.g., $l_L = P_{UL} \times P_{BL}$. Where P_{UL} is the point defining the upper left-hand corner of the image and P_{BL} is the bottom left-hand corner of the image.

- An example of such an image is:

Output:
- Images with the estimated epipolar lines drawn on them (to be included in report as well): ps3-2-c-1.png (pic_a with lines), ps3-2-c-2.png (pic_b with lines)

EXTRA CREDIT PROBLEMS (2-d and 2-e)

If you look at the results of the last section the epipolar lines are close, but not perfect. The problem is that the offset and scale of the points is large and biased compared to some of the constants. To fix this, we are going to normalize the points.

In the 2D case, this normalization is really easy. We want to construct transformations that make the mean of the points zero, and, optionally, scale them so that their magnitude is about 1.0 or some other not-too-large number:

$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -c_u \\ 0 & 1 & -c_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

The transform matrix T is the product of the scale and offset matrices. The c_u, c_v is just the mean. To compute a scale *s*, you could estimate the standard deviation after subtracting the means. Or you could find the maximum absolute value. Then, the scale factor *s* would be the reciprocal of whatever estimate of the scale you are using.

d. Create two matrices T_a and T_b for the set of points defined in the files pts2d-pic_a.txt and pts2d-pic_b.txt respectively. Use these matrices to transform the two sets of points. Then, use these normalized points to create a new Fundamental matrix $\widehat{F}$. Compute it as above, including making the smaller singular value zero.

Output:
- The matrices T_a, T_b and $\widehat{F}$ [text response]

e. Finally you can create a better F by: $F = T_b^T \widehat{F} T_a$

Using this new F, redraw the epipolar lines of 2-c. They should be better.

Output:
- The new F [text response]

- Images with the "better" epipolar lines drawn (to be included in report as well): ps3-2-e-1.png (pic_a), ps3-2-e-2.png (pic_b)