CS231A: Computer Vision, From 3D Reconstruction to Recognition Homework #2

(Winter 2022) Due: **Friday, Februrary 4**

Vasu G. Patel: vgpatel1@stanford.edu

1 Fundamental Matrix Estimation From Point Correspondences (30 points)

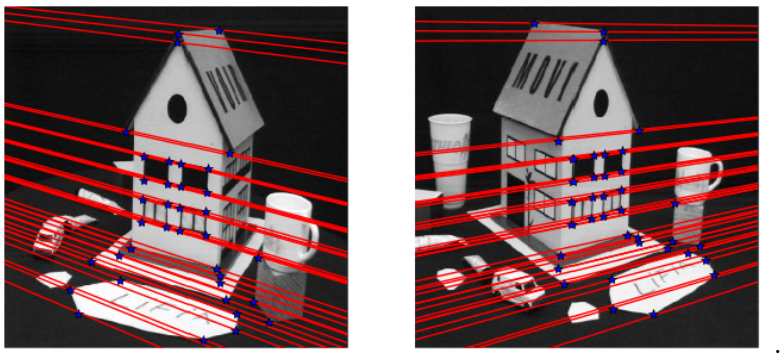


Figure 1: Example illustration, with epipolar lines shown in both images (Images courtesy Forsyth & Ponce)

This problem is concerned with the estimation of the fundamental matrix from point correspondences. In this problem, you will implement both the linear least-squares version of the eight-point algorithm and its normalized version to estimate the fundamental matrices. You will implement the methods in `p1.py` and complete the following:

(a) Implement the linear least-squares eight point algorithm in `lls_eight_point_alg()` and report the returned fundamental matrix. Remember to enforce the rank-two constraint for the fundamental matrix via singular value decomposition. Briefly describe your implementation in your written report.
[20 points]

Answer 1(a): Fundamental Matrix from LLS 8-point algorithm:

$$\begin{bmatrix} 1.55218081e-06 & -8.18161523e-06 & -1.50440111e-03 \\ -5.86997052e-06 & -3.02892219e-07 & -1.13607605e-02 \\ -3.52312036e-03 & 1.41453881e-02 & 9.99828068e-01 \end{bmatrix}$$

Step 1: Build Matrix W
Step 2: Compute SVD of W and obtain $\widehat{F}$ = last column of V matrix (from SVD of W)
Step 3: Impose rank constraint of 2 by zeroing out last singular value
that is F = U.diag(D[0], D[1], 0).V^T

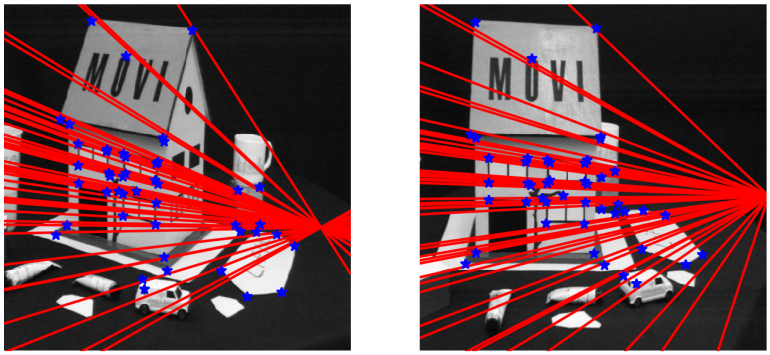


Figure 2: Dataset 1 Linear Least Square method

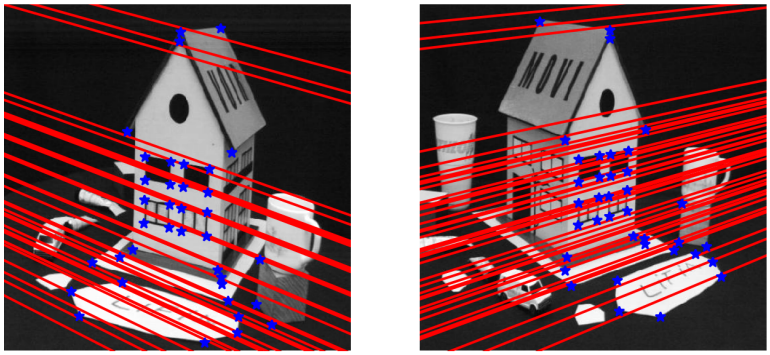


Figure 3: Dataset 2 Linear Least Square method

(b) Implement the normalized eight point algorithm in `normalized_eight_point_alg()` and report the returned fundamental matrix. Remember to enforce the rank-two constraint for the fundamental matrix via singular value decomposition. Briefly describe your implementation

in your written report.
[5 points]

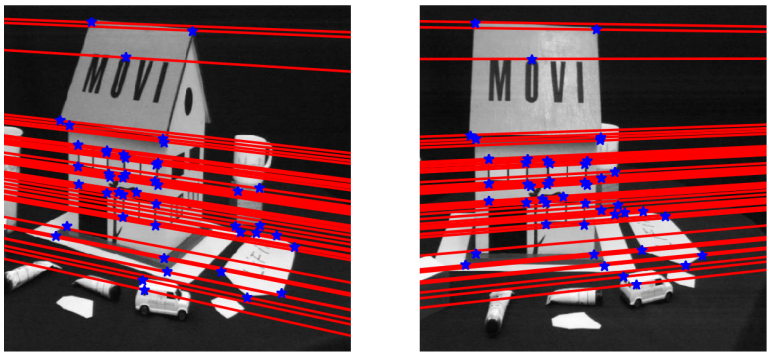
Answer 1(b): Fundamental Matrix from normalized 8-point algorithm:

$$\begin{bmatrix} 6.52113484e-07 & -5.33615067e-06 & 8.80860210e-05 \\ -4.91237449e-06 & -3.40420428e-07 & -6.43807393e-03 \\ -8.56136054e-04 & 8.84208000e-03 & 1.45953063e-01 \end{bmatrix}$$

Step 1: Normalize input image co-ordinates by computing mean and similarity transformation

Step 2: Compute transformation matrix as follows:

$$\begin{bmatrix} S & 0 & -S*(\text{mean}_x) \\ 0 & S & -S*(\text{mean}_y) \\ 0 & 0 & 1 \end{bmatrix}$$



.

Figure 4: Dataset 1 Normalized 8-point algorithm

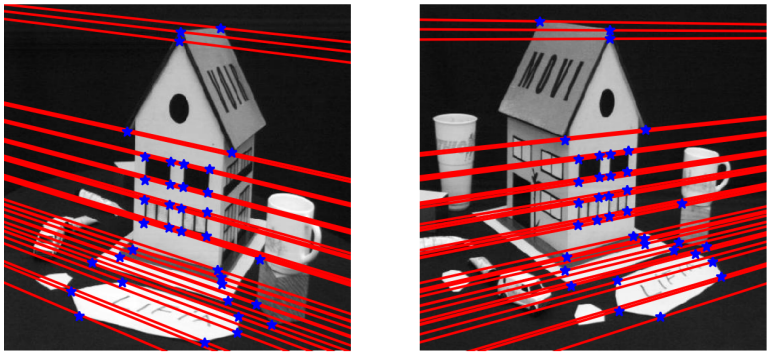


Figure 5: Dataset 2 Normalized 8-point algorithm

(c) After implementing methods to determine the Fundamental matrix, we can now determine epipolar lines. Specifically to determine the accuracy of our Fundamental matrix, we will compute the average distance between a point and its corresponding epipolar line in `compute_distance_to_epipolar_lines()`. Briefly describe your implementation in your written report.
[5 points]

Answer 1(c):
Distance to lines in image 1 for LLS: 9.701438829441459
Distance to lines in image 2 for LLS: 14.568227190508505

Distance to lines in image 1 for normalized: 0.8895134540568641
Distance to lines in image 2 for normalized: 0.8917343723800074

Distance between line ax + by = c and point (x_0, y_0) is given as
$\|ax_0 + by_0 + c\| \frac{}{\sqrt{a^2+b^2}}$

2 Matching Homographies for Image Rectification (20 points)

Building off of the previous problem, this problem seeks to rectify a pair of images given a few matching points. The main task in image rectification is generating two homographies H_1, H_2 that transform the images in a way that the epipolar lines are parallel to the horizontal axis of the image. A detailed description of the standard homography generation for image rectification can be found in Course Notes 3. You will implement the methods in `p2.py` as follows:

(a) The first step in rectifying an image is to determine the epipoles. Complete the function `compute_epipole()`. Include the epipoles you calculated in your written report.
[2 points]

Answer 2(a):
epipole 1: [-1.30071143e+03 -1.42448272e+02 1.00000000e+00]
epipole 2: [1.65412463e+03 4.53021078e+01 1.00000000e+00]

(b) Finally, we can determine the homographies H_1 and H_2. We first compute H_2 as the homography that takes the second epipole e_2 to a point at infinity $(f, 0, 0)$. The matching homography H_1 is computed by solving a least-squares problem. Complete the function `compute_matching_homographies()`. In your written report include a description of your implementation, the homographies computed for the sample image pair, and your rectified images plotted using `plot_epipolar_lines_on_images()`.
[18 points]

Answer 2(b):
Compute H2:

Step 1: compute translation matrix T = $\begin{bmatrix} 1 & 0 & -(image\_width)/2 \\ 0 & 1 & -(image\_height)/2 \\ 0 & 0 & 1 \end{bmatrix}$

Compute translated epipole 2 = T.e2
Step 2: Compute rotation matrix R
Step 3: find 'f' = R.(e2)
Step 4: Compute G matrix = $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1/f & 0 & 1 \end{bmatrix}$

Step 5: Apply sanity check to make sure epipole is located at infinity along horizontal line
H2 homography:
$\begin{bmatrix} 0.80979813 & -0.12203687 & 79.93311835 \\ -0.03001867 & 1.01581538 & 3.63604348 \\ -0.00069936 & 0.00010539 & 1.15205554 \end{bmatrix}$

Compute H1:
Step 1: Compute M = $[e]_x$.F
Step 2: Compute H_a
Step 3: Compute H1 = H_a.H2.M

H1 homography:

$\begin{bmatrix} -12.00063161 & -4.15501447 & -123.47688144 \\ 1.41006481 & -14.87041473 & -284.17746873 \\ -0.00921889 & -0.00219185 & -12.303344 \end{bmatrix}$

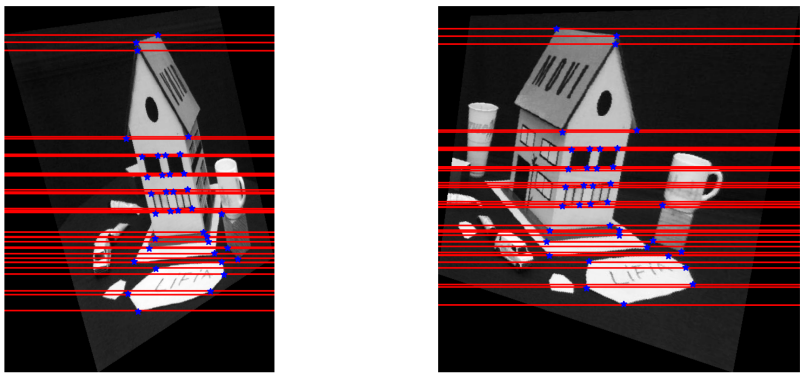


Figure 6: Applying Homography H1 & H2 to image 1, 2 respectively

3 The Factorization Method (15 points)

In this question, you will explore the factorization method, initially presented by Tomasi and Kanade, for solving the affine structure from motion problem. You will implement the methods in `p3.py` and complete the following:

(a) Implement the factorization method as described in lecture and in the course notes. Complete the function `factorization_method()`. Briefly describe your implementation in your written report.
[5 points]

Answer 3(a):
Step 1: compute x_bar for image 1 and image 2
Step 2: Compute x1_hat and x2_hat for image 1 and image 2
Step 3: Compute D matrix by concatenating x1_hat and x2_hat along axis = 0
Step 4: Compute SVD of D having 3 non-singular values
Step 5: Compute structure and motion using SVD decomposition of D:
5(a): Structure = diagonal matrix S(with 3 non-singular values) * last 3 columns of V matrix
5(b): Motion = First three columns of U matrix

(b) Run the provided code that plots the resulting 3D points. Compare your result to the ground truth provided. The results should look identical, except for a scaling and rotation. Explain why this occurs.
[3 points]

Answer 3(b):
Because of similarity transformation, the scaling and rotation ambiguity persist even after carrying out affine structure from motion.

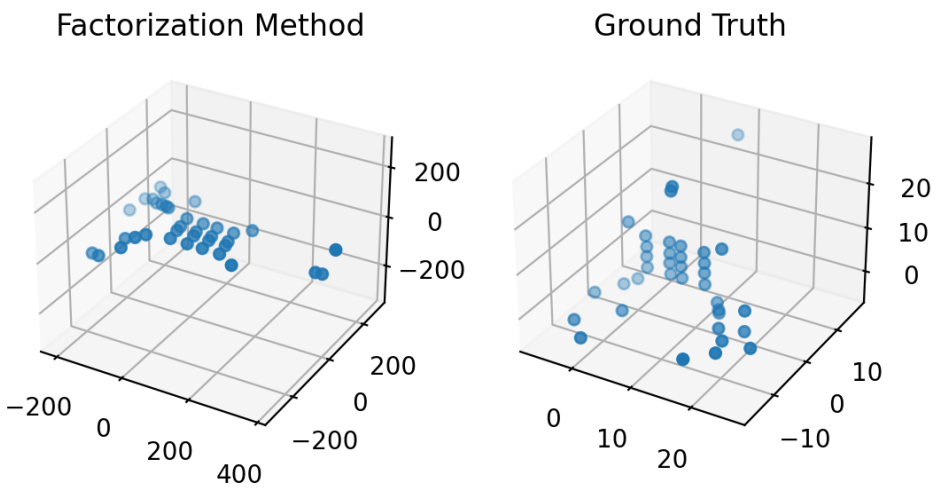


Figure 7: Set 1 image factorization method and ground truth

(c) Report the 4 singular values from the SVD decomposition. Why are there 4 non-zero singular values? How many non-zero singular values would you expect to get in the idealized version of the method, and why?
[2 points]

Answer 3(c): There are four non-singular values observed because of affine approximation of projecting 3D points to a single plane before projecting them to image plane.
S (set1): [959.5852216, 540.47613178, 184.43174791, 27.9151956]

In the idealized situation, we would expect 3 non-singular values, because of structure matrix consists of 3D points in (x, y, z) coordinate. Hence decomposition of D matrix should have atmost 3 non-singular values.

(d) The next part of the code will now only load a subset of the original correspondences. Compare your new results to the ground truth, and explain why they no longer appear similar.
[3 points]

Answer 3(d): Below image shows the results of choosing 3 non-singular values: They do no appear similar because the singular matrix with 3 non-singular values can be merged with either motion (U) or structure (V) matrix. Given this ambiguity, ideally square root of sigma matrix should be taken and then multiply each of them with U and V matrix to obtain correct structure and motion matrix.

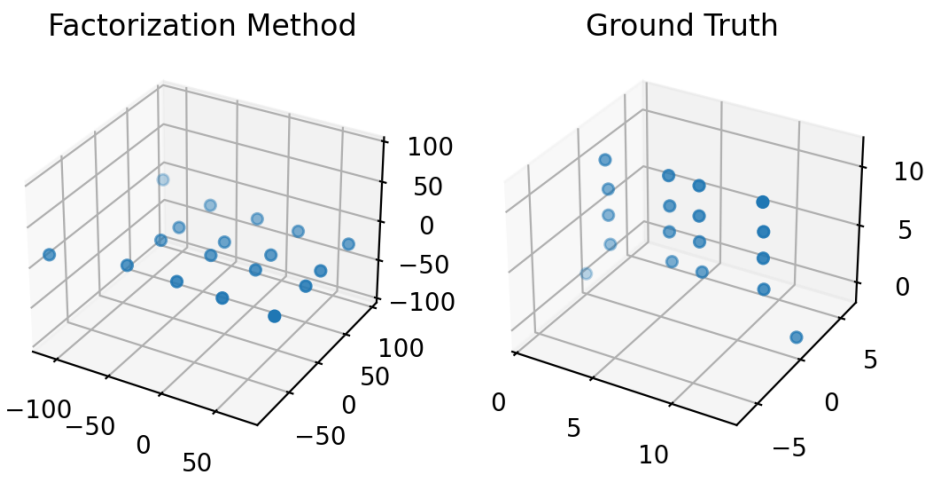


Figure 8: Sub-Set 1 image factorization method and ground truth

(e) Report the new singular values, and compare them to the singular values that you found previously. Explain any major changes.
[2 points]

Answer 3(e): Following singular values are obtained:
S (set1_subset): [264.54396508, 210.06072009, 7.21921783, 5.12857709]

Major changes are observed in the reduced value of singular values for 3rd and 4th singular values, these values have significantly reduced as compared to result from 3(c).

4 Triangulation in Structure From Motion (35 points)

Figure 9: The set of images used in this structure from motion reconstruction.

Structure from motion is inspired by our ability to learn about the 3D structure in the surrounding environment by moving through it. Given a sequence of images, we are able to simultaneously

estimate both the 3D structure and the path the camera took. In this problem, you will implement significant parts of a structure from motion framework, estimating both R and T of the cameras, as well as generating the locations of points in 3D space. Recall that in the previous problem we triangulated points assuming affine transformations. However, in the actual structure from motion problem, we assume projective transformations. By doing this problem, you will learn how to solve this type of triangulation. In Course Notes 4, we go into further detail about this process. You will implement the methods in `p4.py` and complete the following:

(a) Given correspondences between pairs of images, we compute the respective Fundamental and Essential matrices. Given the Essential matrix, we must now compute the R and T between the two cameras. However, recall that there are four possible R, T pairings. In this part, we seek to find these four possible pairings, which we will later be able to decide between.

Answer 4(a): Four possible values of RT are as follows:

```
[[[0.98305251 −0.11787055 −0.14040758  0.99941228]
 [−0.11925737 −0.99286228 −0.00147453 −0.00886961]
 [−0.13923158  0.01819418 −0.99009269  0.03311219]]

 [[0.98305251 −0.11787055 −0.14040758 −0.99941228]
 [−0.11925737 −0.99286228 −0.00147453  0.00886961]
 [−0.13923158  0.01819418 −0.99009269 −0.03311219]]

 [[0.97364135 −0.09878708 −0.20558119  0.99941228]
  [0.10189204  0.99478508  0.00454512 −0.00886961]
  [0.2040601  −0.02537241  0.97862951  0.03311219]]

 [[0.97364135 −0.09878708 −0.20558119 −0.99941228]
  [0.10189204  0.99478508  0.00454512  0.00886961]
  [0.2040601  −0.02537241  0.97862951 −0.03311219]]]
```

(b) In order to distinguish the correct R, T pair, we must first know how to find the 3D point given matching correspondences in different images. The course notes explain in detail how to compute a linear estimate (DLT) of this 3D point: Implement the linear estimate of this 3D point in `linear_estimate_3d_point()`. Like before, we print out a way to verify that your code is working. Include the output generated by this part of the code in your written report.
[5 points]

Answer 4(b): Linear estimate of the 3D point
[0.67671686 −1.10241883 4.66034]
Difference: 0.0029243053036863698

(c) However, we can do better than linear estimates, but usually this falls under some iterative nonlinear optimization. To do this kind of optimization, we need some residual. A simple one is the reprojection error of the correspondences, which is computed as follows:
For each image i, given camera matrix M_i, the 3D point P, we compute $y = M_i P$, and find the image coordinates

$$p_i' = \frac{1}{y_3} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

Given the ground truth image coordinates p_i, the reprojection error e_i for image i is

$$e_i = p_i' - p_i$$

The Jacobian is written as follows:

$$J = \begin{bmatrix} \frac{\partial e_1}{\partial P_1} & \frac{\partial e_1}{\partial P_2} & \frac{\partial e_1}{\partial P_3} \\ \vdots & \vdots & \vdots \\ \frac{\partial e_m}{\partial P_1} & \frac{\partial e_m}{\partial P_2} & \frac{\partial e_m}{\partial P_3} \end{bmatrix}$$

Recall that each e_i is a vector of length two, so the whole Jacobian is a $2K \times 3$ matrix, where K is the number of cameras. Fill in the methods `reprojection_error()` and `jacobian()`, which computes the reprojection error and Jacobian for a 3D point and its list of images. Like before, we print out a way to verify that your code is working. Include the output generated by this part of the code in your written report.
[5 points]

Answer 4(c): Jacobian matrix is as follows

$$\begin{bmatrix} 154.33943931 & 0. & -22.42541691 \\ 0. & 154.33943931 & 36.51165089 \\ 141.87950588 & -14.27738422 & -56.20341644 \\ 21.9792766 & 149.50628901 & 32.23425643 \end{bmatrix}$$

Error Difference: 8.301300130674275e-07
Jacobian Difference: 1.817115702351657e-08

(d) Implement the Gauss-Newton algorithm, which finds an approximation to the 3D point that minimizes this reprojection error. Recall that this algorithm needs a good initialization, which we have from our linear estimate in part (b). Also recall that the Gauss-Newton algorithm is not guaranteed to converge, so, in this implementation, you should update the estimate of the point $\hat{P}$ for 10 iterations (for this problem, you do not have to worry about convergence criteria for early termination):

$$\hat{P} = \hat{P} - (J^T J)^{-1} J^T e$$

where J and e are the Jacobian and error computed from the previous part. Implement the Gauss-Newton algorithm to find an improved estimate of the 3D point in the `nonlinear_estimate_3d_point()` function. Like before, we print out a way to verify that your code is working. Include the output generated by this part of the code in your written report.
[5 points]

Answer 4(d): Non-linear estimate of 3D point:
$\begin{bmatrix} -0.92055794 & -4.63993556 & 4.47305116 \end{bmatrix}$
Linear method error: 98.73542356894195
Nonlinear method error: 95.59481784846034

(e) Now finally, go back and distinguish the correct R, T pair from part (a) by implementing the method `estimate_RT_from_E()`. You will do so by:

1. First, compute the location of the 3D point of each pair of correspondences given each R, T pair
2. Given each R, T you will have to find the 3D point's location in that R, T frame. The correct R, T pair is the one for which the most 3D points have positive depth (z-coordinate) with respect to both camera frames. When testing depth for the second camera, we must transform our computed point (which is the frame of the first camera) to the frame of the second camera.

Submit the output generated by this part of the code.
[5 points]

Answer 4(e): Estimate RT:

$$\begin{bmatrix} 0.97364135 & -0.09878708 & -0.20558119 & 0.99941228 \\ 0.10189204 & 0.99478508 & 0.00454512 & -0.00886961 \\ 0.2040601 & -0.02537241 & 0.97862951 & 0.03311219 \end{bmatrix}$$

(f) Congratulations! You have implemented a significant portion of a structure from motion pipeline. Your code is able to compute the rotation and translations between different cameras, which provides the motion of the camera. Additionally, you have implemented a robust method to triangulate 3D points, which enable us to reconstruct the structure of the scene. In order to run the full structure from motion pipeline, please change the variable `run_pipeline` at the top of the main function to `True`. Submit the final plot of the reconstructed statue. Hopefully, you can see a point cloud that looks like the frontal part of the statue in the above sequence of images.
[10 points]

Answer 4(f):

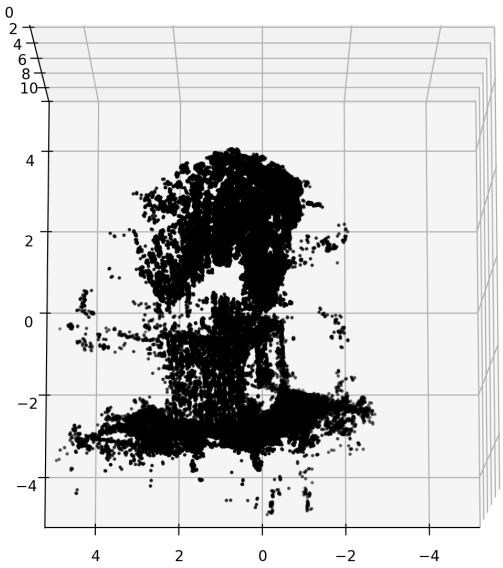


Figure 10: Point cloud for 3D reconstruction of statue