RANSAC
COMP 776, Fall 2017
Due: October 16, 2017
Assignment Data: https://drive.google.com/open?id=0BzP2GHNy5xrfSDgyZFZDX3dybU0

Summary
The goals of this assignment are to 1) give you practical experience with the RANdom SAmple Consensus (RANSAC) algorithm and 2) solidify your understanding of fundamental matrix computation. Here, you will implement RANSAC for F-matrix estimation from a set of (noisy) 2D image correspondences; working RANSAC code for line fitting has also been provided for you to start from. Python code (described below) has been provided for this assignment, along with data for you to experiment on.

Reading

- Szeliski 2.1.2, 2.1.3
- Hartley, Richard I. "In defense of the eight-point algorithm." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19, no. 6 (1997): 580-593. [link]
- Raguram, Rahul, Ondrej Chum, Marc Pollefeys, Jiri Matas, and Jan-Michael Frahm. "USAC: a universal framework for random sample consensus." *IEEE transactions on Pattern Analysis and Machine Intelligence* 35, no. 8 (2013): 2022-2038. [link]
    - Required reading: Section 2; Section 3 optional
- (optional, available via library.unc.edu) Hartley & Zisserman
    - 4.1 (Direct Linear Transform)
    - 11.1 (Basic Equations for the F Matrix)
    - 11.2 (Normalized 8-point Algorithm)
    - A5.3 (Least-squares Solution of Homogeneous Equations)

F-Matrix RANSAC – Overview
This assignment asks you to implement the *RANdom SAmple Consensus (RANSAC)* algorithm in order to verify that feature correspondences (*e.g.*, the matches you found in the previous assignment) are correct. As you likely noted, redundant patterns in the image, as well as noise, frequently lead to incorrect correspondences when we only use feature descriptors to match points. Despite this noise, there often are a number of correct correspondences produced from feature matching. The task of this step is to separate the inlier correspondences from the outliers.

To separate inliers and outliers, we leverage the fact that there is a geometric model defining the transformation of points in one image (one view of the scene) to points or lines in the other image (a different view of the scene). As we discussed in class, the two most general transformations between two images are *homographies* and *fundamental matrices*. For this assignment, you will implement a basic RANSAC loop capable of computing the fundamental matrix relating two images and determining *inliers* from a collection of correspondences. F-matrix RANSAC is often used in Structure-from-Motion to verify image correspondences. This operation is known as *geometric verification* of the "raw" 2D feature matches.

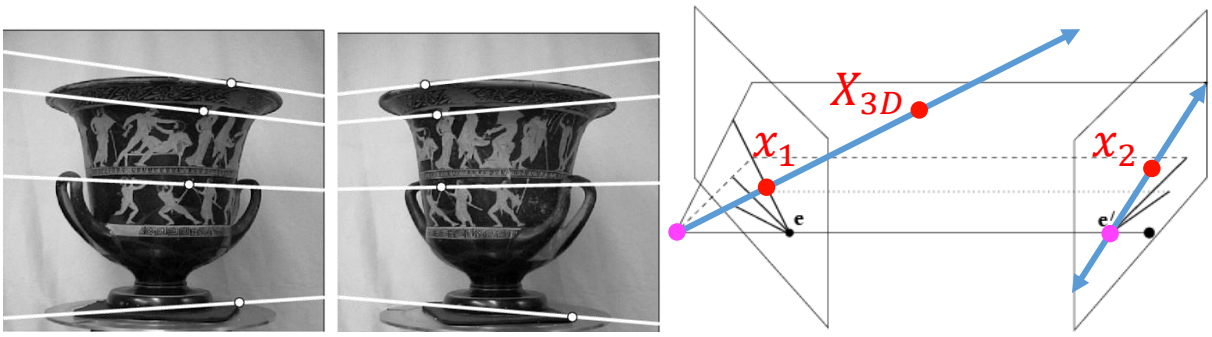


Figure 1. (left) The fundamental matrix maps points in on image into lines in another. The values in the fundamental matrix are determined by the relative rotation and translation of the camera, as well as the camera intrinsics for the two images. (right) Illustration of the point-to-line mapping of the fundamental matrix. The red points represent a point in 3D space and its projection into each of two image planes. We show the *epipole* – the projection of the first image's center of projection into the second image – in magenta. For the left image, we have drawn the 3D ray passing from the camera center through the 3D point (blue line). This ray projects to an *epipolar line* in the right image (blue line with two arrows). All epipolar lines pass through the epipole.

Fundamental Matrix Computation

Recall that the fundamental matrix F maps a point $\boldsymbol{x_1}$ in one image into a line $\boldsymbol{l} = F\boldsymbol{x_1}$ in a second image. (See Fig. 1.) We call $\boldsymbol{l}$ an *epipolar line* because it passes through the *epipole* (the projection of the first image's center of projection into the second image's image plane). If $\boldsymbol{x_2}$ is the corresponding point in the second image, then $\boldsymbol{x_2}$ falls on the line $\boldsymbol{l}$, *i.e.*, $\boldsymbol{x_2^T F x_1} = 0$. We say that F encodes the *epipolar geometry* (a function of camera motion) of the image pair.

Hartley (see assigned reading above) gives a good overview of how to compute the fundamental matrix given a set of 2D point correspondences in a pair of images. Here's a brief outline of the steps involved. Given a set of correspondences $\{(\boldsymbol{x_1^1}, \boldsymbol{x_2^1}), (\boldsymbol{x_1^2}, \boldsymbol{x_2^2}), \ldots, (\boldsymbol{x_1^M}, \boldsymbol{x_2^M})\}$:

1) Normalize the points in for each image independently such that their mean is $(0,0)^T$ and their average distance to the origin is $\sqrt{2}$. This normalization is necessary to provide good conditioning on the matrix A in the following DLT computation.
2) Compute the fundamental matrix using the direct linear transform (DLT; see next page).
3) Enforce the rank-2 constraint on F (see next page).
4) Re-apply Hartley's normalization (step 1) to the left and right sides of F. That is, if T_1 and T_2 are 3×3 matrices that normalize the first and second point sets, respectively, then our final matrix is $F' = T_2^T F T_1$, such any general pixel coordinate $(x, y, 1)$ can be mapped by F'. (Check your knowledge (no need to include in your write-up): What kind of transformation are T_1 and T_2?)

The Direct Linear Transform (DLT) and F-matrix Estimation

Each pair of normalized points $(\tilde{\boldsymbol{x}}_1^{\boldsymbol{i}}, \tilde{\boldsymbol{x}}_2^{\boldsymbol{i}})$ give us one epipolar constraint: $\left(\tilde{\boldsymbol{x}}_2^{\boldsymbol{i}}\right)^T F \tilde{\boldsymbol{x}}_1^{\boldsymbol{i}} = 0$. As discussed by Hartley, this can equivalently be re-written as $(\boldsymbol{a}^{\boldsymbol{i}})^T \boldsymbol{f} = 0$, where $\boldsymbol{f}$ is a flattened version of F and $\boldsymbol{a}^{\boldsymbol{i}}$ is a vector derived from $(\tilde{\boldsymbol{x}}_1^{\boldsymbol{i}}, \tilde{\boldsymbol{x}}_2^{\boldsymbol{i}})$. If we have multiple point pairs, we can build up a *linear system of equations*:

$$A\boldsymbol{f} = \begin{bmatrix} (\boldsymbol{a}^{\boldsymbol{1}})^T \\ (\boldsymbol{a}^{\boldsymbol{2}})^T \\ \vdots \\ (\boldsymbol{a}^{\boldsymbol{m}})^T \end{bmatrix} \boldsymbol{f} = 0.$$

As you can see, this system of equations is unique only up to scale – any multiplication by a constant still gives equality to zero. It turns out [Hartley & Zisserman, A5.3] that the solution for $\boldsymbol{f}$ is the eigenvector of $A^T A$ with the smallest associated eigenvalue. Long story short, we have the following steps for performing the DLT:

1) Take the singular value decomposition (SVD) of A, *i.e.*, $A = USV^T$. S is a diagonal matrix whose entries are the singular values of $A^T A$. By convention, the singular values are ordered greatest-to-least along the diagonal.
2) It follows that the last column of V (last row of V^T) contains the (unit-length) eigenvector corresponding to the smallest eigenvalue.[1] This gives us our solution for $\boldsymbol{f}$, which can be reshaped into the 3×3 matrix F.

Enforcing the Rank-2 Constraint

One important property of the fundamental matrix is that it has rank 2, but the solution F provided by the DLT will generally not have this property [Hartley & Zimmerman, 11.1]. To address this issue, we can do the following:

- Take the SVD of F: $F = USV^T$. (Note, this is a second SVD applied to the DLT solution.) Replace the smallest singular value in S with 0 and call the resulting matrix S'. Then, our rank-2 constrained matrix is $F' = US'V^T$.

F-Matrix RANSAC

Hartley's algorithm gives us a way to estimate a fundamental matrix for a given set of 2D point correspondences. We can embed this in a RANSAC framework as follows:

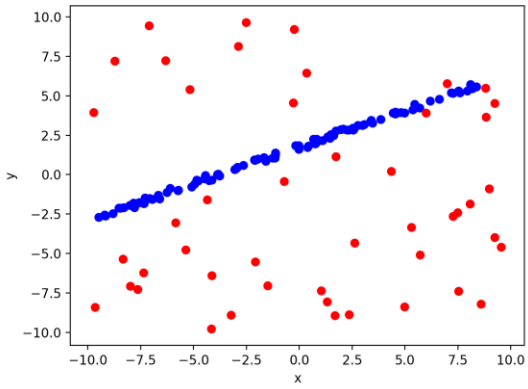
1) Select the minimum number of 2D point correspondences required to perform the F-matrix DLT in Hartley's algorithm.
2) Estimate the fundamental matrix F from this point set using Hartley's algorithm.
3) For *all* given 2D point correspondences $\left(\boldsymbol{x}_1^{\boldsymbol{i}}, \boldsymbol{x}_2^{\boldsymbol{i}}\right)$, calculate the distance (in pixels) of $\boldsymbol{x}_2^{\boldsymbol{i}}$ to the epipolar line defined by $F\boldsymbol{x}_1^{\boldsymbol{i}}$. Mark correspondences as *inlier correspondences* if this distance is less than some threshold τ. Don't forget: proper point-to-line distance requires that the epipolar line $\boldsymbol{l}$ be normalized to $\boldsymbol{l} = (n_x, n_y, d)$ with $n_x^2 + n_y^2 = 1$.
4) Repeat steps (1) through (3) until the stopping conditions for RANSAC have been met.
5) Compute a final matrix F using Hartley's algorithm on all inlier correspondences. Also, re-compute the final set of inlier correspondences using this matrix.

---

[1] Note that numpy's svd implementation returns V^T, not V.

Code

We have provided you with a complete RANSAC example for fitting a 2D line to 2D point data. You are encouraged (but not required) to use this code as a basis for your F-matrix RANSAC implementation. In any case, executing the `line_fitting_ransac.py` file should give you the following output (inliers shown in blue, outliers in red):




```
Actual Model:
(0.42262, -0.90631, 1.57077)

Inlier Ratio (before refinement): 0.673
Best Fit Model (before refinement):
(0.43525, -0.90031, 1.58592)

Iterations: 8
Inlier Ratio: 0.687
Best Fit Model:
(0.42298, -0.90614, 1.59531)
```

The text on the right shows the program output. The actual 2D line model parameters (n_x, n_y, d) are printed at the top. Parameters after RANSAC estimation are printed next, and at the bottom are the model parameters after a final DLT on all RANSAC inliers. As you can see, the line equation and inlier ratio improves slightly after this final model estimation.

Concerning the fundamental matrix estimation, we have provided skeleton code in the file `f_matrix_ransac.py`. All algorithm parameters and data inputs have been set up for you, and you only need to implement the actual F-matrix RANSAC code; for usage, execute:

```
python f_matrix_ransac.py --help
```

You are free to make changes to the code as you see fit, but please keep the `np.random.seed(0)` line at the top, so that we can deterministically evaluate your results. We suggest writing separate functions (perhaps in a separate file) that 1) implement Hartley's algorithm on a set of correspondences and 2) compute inlier correspondences for a given fundamental matrix. These functions will allow you to avoid duplicate code when calculating the final fundamental matrix from all inlier correspondences.

Submission

We have provided a single image pair with corresponding 2D keypoints. Implement F-matrix RANSAC for this image pair and show 1) the inlier matches, 2) the outlier matches, and 3) the epipolar lines for the inlier matches (visualization code already provided in `util.py`). Also report the number of RANSAC iterations taken, the final inlier ratio, and the values of the final estimated fundamental matrix. Use the default RANSAC parameters specified in the command-line arguments (inlier threshold of 2px, RANSAC confidence threshold of 99%, and a maximum of 10,000 RANSAC iterations).

Also in your PDF, please include your name and a link to your code in your department Google Drive account. Be sure to share the folder with Jan-Michael Frahm, Marc Eder, and True Price.