Spring 2018 CS543/ECE549

Assignment 3: Robust estimation and geometric vision

Due date: April 12, 11:59:59PM

The goal of this assignment is to implement robust homography and fundamental matrix estimation to register pairs of images, as well as attempt triangulation and single-view 3D measurements.

Contents

Part 1: Stitching pairs of images

Python code here and in Part 2 by Lavisha Aggarwal

The first step is to write code to stitch together a single pair of images. For this part, you will be working with the following pair (click on the images to download the high-resolution versions):

1. Load both images, convert to double and to grayscale.

2. Detect feature points in both images. We provide Harris detector code you can use (MATLAB, Python) or feel free to use the blob detector you wrote for Assignment 2.

3. Extract local neighborhoods around every keypoint in both images, and form descriptors simply by "flattening" the pixel values in each neighborhood to one-dimensional vectors. Experiment with different neighborhood sizes to see which one works the best. If you're using your Laplacian detector, use the detected feature scales to define the neighborhood scales.

   Optionally, feel free to experiment with SIFT descriptors.
   - **MATLAB:** Here is some basic code for computing SIFT descriptors of circular regions, such as the ones returned by the detector from Assignment 2.
   - **Python:** You can use the OpenCV library to extract keypoints through the function `cv2.xfeatures2D.SIFT_create().detect` and compute descripters through the function `cv2.xfeatures2D.SIFT_create().compute`. This tutorial provides details about using SIFT in OpenCV.

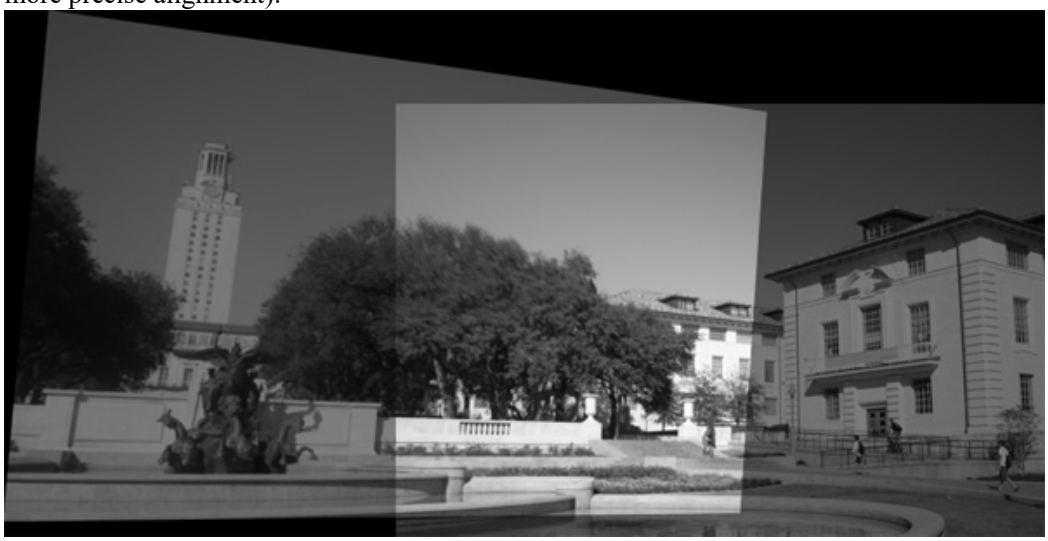
4. Compute distances between every descriptor in one image and every descriptor in the other image. In MATLAB, you can use this code for fast computation of Euclidean distance. In Python, you can use `scipy.spatial.distance.cdist(X,Y,'sqeuclidean')` for fast computation of Euclidean distance. If you are not using SIFT descriptors, you should experiment with computing normalized correlation, or Euclidean distance after normalizing all descriptors to have zero mean and unit standard deviation.

5. Select putative matches based on the matrix of pairwise descriptor distances obtained above. You can select all pairs whose descriptor distances are below a specified threshold, or select the top few hundred descriptor pairs with the smallest pairwise distances.

6. Run RANSAC to estimate a homography mapping one image onto the other. Report the number of inliers and the average residual for the inliers (squared distance between the point coordinates in one image and the transformed coordinates of the matching point in the other image). Also, display the locations of inlier matches in both images.

7. Warp one image onto the other using the estimated transformation. To do this in MATLAB, you will need to learn about `maketform` and `imtransform` functions. In Python, use `skimage.transform.ProjectiveTransform` and `skimage.transform.warp`.

8. Create a new image big enough to hold the panorama and composite the two images into it. You can composite by simply averaging the pixel values where the two images overlap. Your result should look something like this (but hopefully with a more precise alignment):



You should create color panoramas by applying the same compositing step to each of the color channels separately (for estimating the transformation, it is sufficient to use grayscale images).

Tips and Details

- For RANSAC, a very simple implementation is sufficient. Use four matches to initialize the homography in each iteration. You should output a single transformation that gets the most inliers in the course of all the iterations. For the various RANSAC parameters (number of iterations, inlier threshold), play around with a few "reasonable" values and pick the ones that work best. Refer to this lecture for details on RANSAC. For randomly sampling matches, you can use the `randperm` or `randsample` functions.

- For details of homography fitting, you should review this lecture.

- Homography fitting calls for homogeneous least squares. The solution to the homogeneous least squares system AX=0 is obtained from the SVD of A by the singular vector corresponding to the smallest singular value. In MATLAB, use `[U,S,V]=svd(A); X = V(:,end);`

  In Python, `U, s, V = numpy.linalg.svd(A)` performs the singular value decomposition and `V[len(V)-1]` gives the smallest singular value.

For extra credit

- Extend your homography estimation to work on multiple images. You can use **this data**, consisting of three sequences consisting of three images each. For the "pier" sequence, sample output can look as follows (although yours may be different if you choose a different order of transformations):

Source of data and results: Arun Mallya

Alternatively, feel free to acquire your own images and stitch them.

- Experiment with registering very "difficult" image pairs or sequences -- for instance, try to find a modern and a historical view of the same location to mimic the kinds of composites found here. Or try to find two views of the same location taken at different times of day, different times of year, etc. Another idea is to try to register images with a lot of repetition, or images separated by an extreme transformation (large rotation, scaling, etc.). To make stitching work for such challenging situations, you may need to experiment with alternative feature detectors and/or descriptors, as well as feature space outlier rejection techniques such as Lowe's ratio test.

- Try to implement a more complete version of a system for "Recognizing panoramas" -- i.e., a system that can take as input a "pile" of input images (including possible outliers), figure out the subsets that should be stitched together, and then stitch them together. As data for this, either use images you take yourself or combine all the provided input images into one folder (plus, feel free to add outlier images that do not match any of the provided ones).

- Implement bundle adjustment or global nonlinear optimization to simultaneously refine transformation parameters between all pairs of images.

- Learn about and experiment with image blending techniques and panorama mapping techniques (cylindrical or spherical).

Part 2: Fundamental Matrix Estimation and Triangulation

You will be using these two image pairs:

Download full size images, matching points, camera matrices, and MATLAB sample code. Python sample code can be found here.

1. Load the image pair and matching points file (see sample code in the data file).

2. Fit a **fundamental matrix** to the matching points. Use the sample code provided to visualize the results. Implement both the **normalized** and the **unnormalized** algorithms (see this lecture for the methods). In each case, report your residual, or the mean squared distance *in pixels* between points in both images and the corresponding epipolar lines.

3. Now use your putative match generation and RANSAC code from Part 1 to estimate fundamental matrices *without ground-truth matches*. For this part, only use the normalized algorithm. Report the number of inliers and the average residual for the inliers. In your report, compare the quality of the result with the one you get from ground-truth matches.

4. Load the 3x4 camera matrices for the two images. In MATLAB, they can be loaded with the `load` command, i.e., `P1 = load('house1_camera.txt');` In Python, use `P1 = numpy.loadtxt('house1_camera.txt')`. Find the centers of the two cameras. Use linear least squares to **triangulate** the position of each matching pair of points given the two cameras (see this lecture for the method). Display the two camera centers and reconstructed points in 3D. Also report the residuals between the observed 2D points and the projected 3D points in the two images.

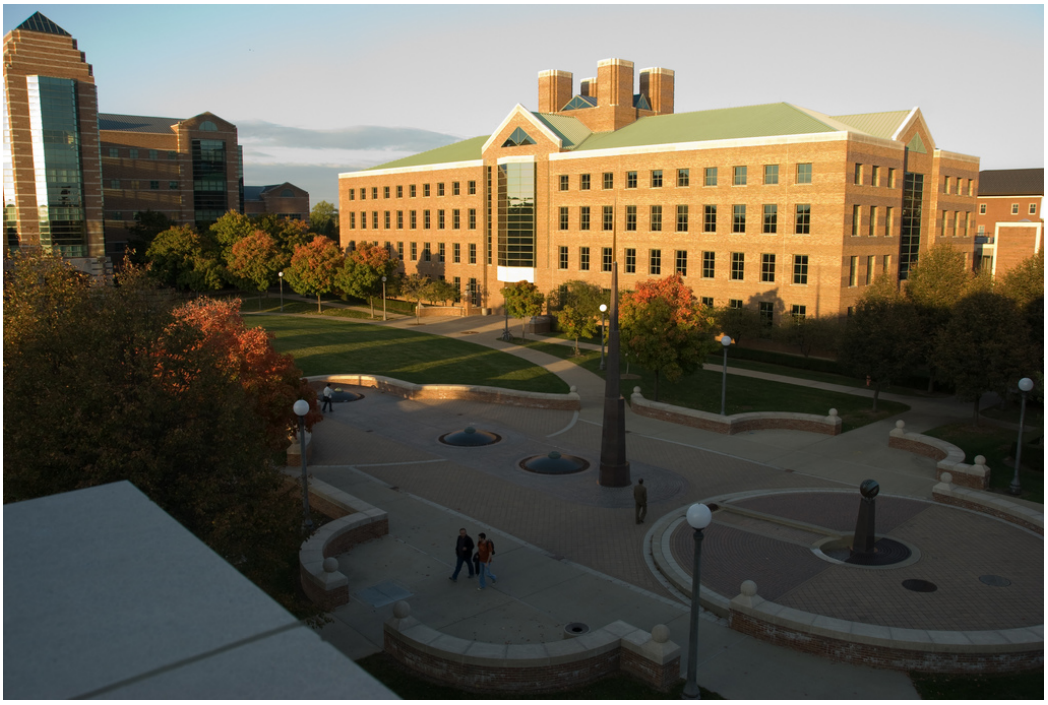
   Note: you do not need the camera centers to solve the triangulation problem. They are used just for the visualization.

   Note 2: it is sufficient to only use the provided ground-truth matches for this part. But if you wish, feel free to also generate and compare results with the inlier matches you have found in #3 above.

Tips and Details

- For fundamental matrix estimation, don't forget to enforce the rank-2 constraint. This can be done by taking the SVD of F, setting the smallest singular value to zero, and recomputing F.
- Lecture 16 shows two slightly different linear least squares setups for estimating the fundamental matrix (one involves a homogeneous system and one involves a non-homogeneous system). You may want to compare the two and determine which one is better in terms of numerics.
- Recall that the camera centers are given by the null spaces of the matrices. They can be found by taking the SVD of the camera matrix and taking the last column of V.
- For triangulation with linear least squares, it is not necessary to use data normalization (in my implementation, normalization made very little difference for this part).
- In MATLAB, plotting in 3D can be done using the `plot3` command. Use the `axis equal` option to avoid automatic nonuniform scaling of the 3D space. In Python, use the `Axes3D` function from `mpl_toolkits.mplot3dplot3` library. To show the structure clearly, you may want to include snapshots from several viewpoints. In the past, some students have also been able to produce animated GIF's, and those have worked really well.

Part 3: Single-View Geometry



Adapted from Derek Hoiem by Tom Paine. Python code by Hsiao-Ching Chang

1. You will be working with the above image of the North Quad (save it to get the full-resolution version). First, you need to estimate the three major orthogonal vanishing points. Use at least three manually selected lines to solve for each vanishing point. This MATLAB code and this Python code provide an interface for selecting and drawing the lines, but the code for computing the vanishing point needs to be inserted. For details on estimating vanishing points, see Derek Hoiem's book chapter (section 4). You should also refer to this chapter and the single-view metrology lecture for details on the subsequent steps. In your report, you should:
    - Plot the VPs and the lines used to estimate them on the image plane using the provided code.
    - Specify the VP pixel coordinates.
    - Plot the ground horizon line and specify its parameters in the form a * x + b * y + c = 0. Normalize the parameters so that: a^2 + b^2 = 1.

2. Using the fact that the vanishing directions are orthogonal, solve for the focal length and optical center (principal point) of the camera. Show all your work.

3. Compute the rotation matrix for the camera, setting the vertical vanishing point as the Y-direction, the right-most vanishing point as the X-direction, and the left-most vanishing point as the Z-direction.

4. Estimate the heights of (a) the CSL building, (b) the spike statue, and (c) the lamp posts assuming that the person nearest to the spike is 5ft 6in tall. In the report, show all the lines and measurements used to perform the calculation. How do the answers change if you assume the person is 6ft tall?

Part 3 Extra Credit

- Perform additional measurements on the image: which of the people visible are the tallest? What are the heights of the windows? etc.

- Compute and display rectified views of the ground plane and the facades of the CSL building.

- Attempt to fit lines to the image and estimate vanishing points automatically either using your own code or code downloaded from the web.

- Find or take other images with three prominently visible orthogonal vanishing points and demonstrate varions measurements on those images.

Grading checklist

1. Homography estimation:
    a. Describe your solution, including any interesting parameters or implementation choices for feature extraction, putative matching, RANSAC, etc.
    b. For the `uttower` pair provided, report the number of homography inliers and the average residual for the inliers (squared distance between the point coordinates in one image and the transformed coordinates of the matching point in the other image). Also, display the locations of inlier matches in both images.
    c. Display the final result of your stitching.
2. Fundamental matrix estimation:
    a. For both image pairs, for both unnormalized and normalized estimation using ground truth matches, display your result and report your residual.
    b. For both image pairs, for normalized estimation without ground truth matches, display your result and report your number of inliers and residual for inliers.
    c. For both image pairs, visualize 3D camera centers and triangulated 3D points.
3. Single-view geometry: See items 1-4 in Part 3 above.

Instructions for turning in the assignment

As usual, submit your PDF report and zipped code archive, named **lastname_firstname_a3.pdf** and **lastname_firstname_a3.zip** via Compass2g.

Multiple attempts will be allowed but by default, only your last submission will be graded. **We reserve the right to take off points for not following directions.**

Late policy: You lose 25% of the points for every day the assignment is late. If you have a compelling reason for not being able to submit the assignment on time and would like to make a special arrangement, you must send me email **at least a week before the due date** (any genuine emergency situations will be handled on an individual basis).

Academic integrity: Feel free to discuss the assignment with each other in general terms, and to search the Web for general guidance (not for complete solutions). Coding should be done individually. If you make substantial use of some code snippets or information from outside sources, be sure to acknowledge the sources in your report. At the first instance of cheating (copying from other students or unacknowledged sources on the Web), a grade of zero will be given for the assignment. At the second instance, you will automatically receive an F for the entire course.