EECS 442 Computer Vision: Homework 3

Instructions

- This homework is **due at 11:59:59 p.m. on Thursday March 14th, 2019**.
- The submission includes two parts:
  1. **To Gradescope:** a `pdf` file as your write-up, including your answers to all the questions and key choices you made for solving problems.
     You might like to combine several files to make a submission. Here is an example online link for combining multiple PDF files: https://combinepdf.com/.
     Please mark where each question is on gradescope.
  2. **To Canvas:** a `zip` file including all your code.
- The write-up must be an electronic version. **No handwriting, including plotting questions.** LaTeX is recommended but not mandatory.

1 RANSAC [10 pts]

1.1 Fitting a Line [4 pts]

In this section, each question depends on on the previous one. By *putative model*, we mean the model that is picked in the inner loop of RANSAC.

- (1 pt) Suppose we are fitting a line (e.g., $y = mx + b$). How many points do we need to sample in an iteration to compute a putative model?
- (1 pt) In the previous setting, suppose the outlier ratio of the data set is 0.1. What is the probability that a putative model fails to get a desired model?
- (2 pts) In the previous settings, to exceed a confidence level of 95% for success, how many trials should we attempt to fit putative models?

1.2 Fitting Transformations [6 pts]

We'll begin by reviewing fitting transformations from 2D points to 2D points. You will need to use these results to solve the subsequent sections.

1. Recall that a matrix $\mathbf{M} \in \mathbb{R}^{2\times 2}$ is a linear transformation: $\mathbb{R}^2 \to \mathbb{R}^2$. Given a dataset $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ where $\mathbf{x}_i, \mathbf{y}_i \in \mathbb{R}^2$. We are going to fit a linear transformation $\mathbf{y} = \mathbf{M}\mathbf{x}$.
   (i) (2 pts) How many degrees of freedom does $\mathbf{M}$ have? How many samples are required to find $\mathbf{M}$?

(ii) (1 pt) Suppose we have a dataset $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ and want to fit $\mathbf{y} = \mathbf{M}\mathbf{x}$. Formulate the fitting problem into the form of a least squares problem:

$$\text{argmin}_{\mathbf{m}} \|A\mathbf{m} - b\|^2$$

where $\mathbf{m}$ is some vector that has all the parameters of $\mathbf{M}$.

2. Use `numpy.load()` to load `p1/transform.npy`. Each row contains two points x and y, represented in the form $[\mathbf{x}_i^T, \mathbf{y}_i^T]_{1\times 4}$. Fit a transformation based on different assumptions.

   (a) (1 pt) Fit a transformation

   $$\mathbf{y} = \mathbf{S}\mathbf{x} + \mathbf{t}$$

   where $\mathbf{S} \in \mathbb{R}^{2\times 2}, \mathbf{t} \in \mathbb{R}^{2\times 1}$. Solve the problem by setting up an optimization of the form of $\text{argmin}_{\mathbf{v}} \|A\mathbf{v} - b\|^2$.

   (b) (1 pt) Fit a homography

   $$\begin{bmatrix}\mathbf{y}\\ 1\end{bmatrix} \equiv \mathbf{H}\begin{bmatrix}\mathbf{x}\\ 1\end{bmatrix}$$

   where $\mathbf{H} \in \mathbb{R}^{3\times 3}$. Solve the problem by dealing with the optimization of the form of $\text{argmin}_{\mathbf{h}} \|A\mathbf{h}\|^2$ with $\|\mathbf{h}\| = 1$ where $\mathbf{h}$ has all the parameters of $\mathbf{H}$.

   (c) (1 pt) Try to explain the difference between two fitting results.
   Hint:

   - You can compare the fitting results by transforming some $\mathbf{x}$ with two models. The comparing process is not required in the report.
   - Compare the assumptions of the two transformations by finding a homography equivalent to the transformation in (a), *i.e.* finding a $\mathbf{H}_{(\mathbf{S},\mathbf{t})} \in \mathbb{R}^{3\times 3}$ s.t. $\begin{bmatrix}\mathbf{y}\\ 1\end{bmatrix} \equiv \mathbf{H}_{(\mathbf{S},\mathbf{t})}\begin{bmatrix}\mathbf{x}\\ 1\end{bmatrix}$ if and only if $\mathbf{y} = \mathbf{S}\mathbf{x} + \mathbf{t}$

2 Image Stitching [40pts]

Image stitching or photo stitching combines multiple photographic images that have overlapping fields of view to produce a segmented panorama or high-resolution image. You'll be able to do this by the end of this problem (which, along with Section 3, is derived from an assignment developed d by Svetlana Lazebnik at UIUC). For this part, you will be working with the following images.

Figure 1: Input of Image Stitching.

Here are the instructions:

1. Load both images: `data_ttower_left.jpg` and `data_ttower_right.jpg`. Convert them to double and to grayscale.

2. Detect feature points in both images. You can use the corner detector code developed in the last homework. Feel free to experiment with SURF/SIFT descriptors in `OpenCV`. **Report** your choice. **Display** both the images along with the feature points.

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 instead using descriptors in `OpenCV`, describe how you get those descriptors and what is the meaning of the descriptors. **Report** your method.

4. Compute distances between every descriptor in one image and every descriptor in the other image. Alternatively, experiment with computing normalized correlation, or Euclidean distance after normalizing all descriptors to have zero mean and unit standard deviation. **Report** your choices.
   Note: You are not allowed to use built-in functions to match features for you, including but not limit to `cv2.BFMatcher`. However, you can use them to debug your code and compare to your implementation.

5. Select putative matches based on the matrix of pairwise descriptor distances obtained above. You can (i) select all pairs whose descriptor distances are below a specified threshold; (ii) select the top few hundred descriptor pairs with the smallest pairwise distances; or (iii) as described in lecture, compute the ratio test described in lecture (nearest-neighbor to second-nearest-neighbor ratio test). **Report** your choices.

6. Run RANSAC to estimate a homography mapping one image onto the other. For the best fit, **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. (i.e. for the inliers matches, show lines between matching locations in the two images). You can use `cv2.drawKeypoints` to draw matches.
   Note: You need to implement RANSAC and calculate the transform matrix. You are not allowed to use functions that do RANSAC in one line, including but not limit to `cv2.findHomography` or `cv2.getPerspectiveTransform`. However, you can use them to debug your code and compare to your implementation.

7. Warp one image onto the other using the estimated transformation. To do this, you will need to learn about `cv2.warpPerspective`. Please read the documentation.

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. **Display** a colored image, which might look like this:

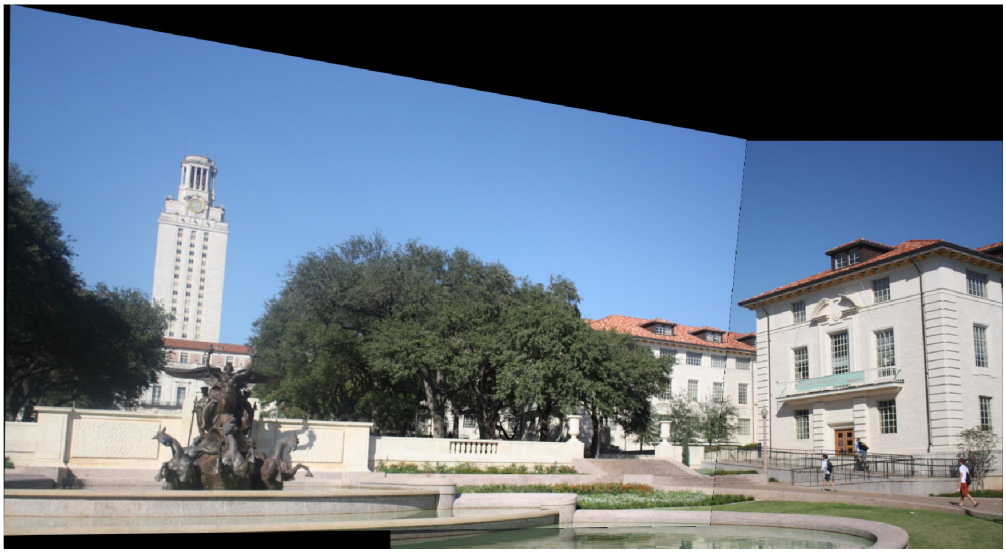

Figure 2: Sample output of Image Stitching.

9. Try to run your code on `BBB_left.jpg` and `BBB_right.jpg`. Display the detected feature points, the matching result, the inliers after RANSAC, and the stitched image.

3 Panoramic Recognition and Multiple Images Stitching [30pts]

Now, we want to implement a panoramic recognition algorithm. The panoramic images are those include a wide-angle view or representation of a physical space. In folder `p3`, there are two image sets, `muggle_world` and `wizarding_world`, containing several images from panoramas and noise images (those do not overlap with the others).

Here are the instructions:

1. The algorithm should be implemented to automatically identify images that belong to the same panorama by computing the number of matches between each images.

   The easiest way to implement the matching algorithm is to match features in every pair of images. The cost is $O(N^2M^2)$ where N is the number of images and M is the number of features in an image. Here, we provide a reference for a faster implementation:

   The general idea of matching is:

   - Extract M features from each of the N images
   - Match each of the $N \times M$ features to its k nearest neighbors in other images.
   - Match the images based on the matching of image features

   In this problem, you may find SIFT/SURF feature detector and descriptor to be better than Harris detector and using the normalized patches. **Report** your choice of feature detector. **Explain** how you implement the panoramic recognition algorithm. **List** the names of images that belong to the same panorama.

2. After finding images that belong to the same panorama, the next step is to extend the code in part 2 for pairwise stitching to work on multiple images. You can test this part independently. The basic idea is - for every pair of images, determine the number of inliers of RANSAC for each transformation; use this information to determine which pair of images should be merged first (and of these, which one should be the "source" and which the "destination"); merge this pair if should. Then proceed recursively for pairs of images or merged images. **Explain** how you implement the multiple images stitching algorithm, especially how to determine the order of stitching.
   For the "pier" sequence, sample output can look as follows (although yours may be different if you choose a different order of transformations).

Figure 3: Stitch results for pier. Source of data and results: Arun Mallya.

3. Create your own set of images and stitch them using your panorama pipeline.
4. **Display** the panoramic images from your panorama pipeline.

Note: For this part, you should experiment with and discover your own strategy for panoramic recognition and multi-image stitching. For full credit, the image selection and the order of merging should be determined automatically. Here is a sample output from Google Photos, you are not required you to output the exact same image as Google produces. And you are not required to implement blending in this part. It is fine to keep the black background as shown in Figure 3. But we will give you bonus on improving the stitching quality by implementing image blending techniques and panorama mapping techniques (cylindrical or spherical) as mentioned in the Bonus points section.

Figure 4: Sample output for panoramic recognition.

4 Fitting planes on a point cloud [20pts]

In this question, you are going to detect planes in a depth image/point cloud and label it on the corresponding RGB image. In folder `p4`, you will find a colored image and a depth image. Those images are from NYU Depth Dataset captured by Microsoft Kinect, which features a RGB camera and a depth sensor. The sensor captures a normal color image and a corresponding depthmap so each pixel has RGB and depth. You will convert a depth image into a point cloud and then use Sequential RANSAC (aka run RANSAC multiple times) to fit planes in the scene.

A plane is a flat surface in 3D space. As you probably know from geometry class, a plane through the origin can be described by the simple equation $ax + by + cz = 0$.

A plane is parameterized by its surface normal, $n \in \mathbb{R}^3$ s.t. $||n||_2 = 1$ (describing the direction the plane faces) as well as an offset $d \in \mathbb{R}$. Further, all points (x, y, z) on the plane must satisfy $n^T[x, y, z] - d = 0$ (i.e., $n_1x + n_2y + n_3z - d = 0$).

Fitting a plane then involves finding the normal n and d that minimize the above equation for a set of points. **Hint:** because we derived it in general, you can just update the total least squares results from the slides to have x in 3D.

Here is the general outline of what you should do to solve this problem:

1. Load both the RGB image and the depth image. The depth image describes the distance of the corresponding point to the plane passing through the camera center parallel to the image plane. Basically, if you set up a coordinate frame at the camera center with X and Y as usual in the image, Z sticks out towards the scene. We provide code to do this in the starter code.

2. Convert a depth image to a point cloud. This is a set of points $K \times 3$ where K is the number of points (one per pixel) and each row represents a single point. You can visualize a point cloud using `draw_geometries()` in Open3D. In the starter code, we already have done this for you.

3. Implement RANSAC to detect planes in the point cloud. The basic idea of plane detection is

   - Use RANSAC to fit one plane at a time. For k iterations, sample the least number of points d in the point cloud to fit a plane m and find the inliers of a fitting. Find the best fit in k runs and return the plane.
   - Check the quality of the plane (based on the shape and size of the plane). If the quality is good (you can use some heuristic), then label the corresponding RGB image to be a detected plane and remove the detected plane from the point cloud.
   - Repeat previous steps for several times until there is no plane in good quality detected.

4. For each plane, project its corresponding 3D points to 2D using the camera intrinsic matrix.

5. In the report, describe how you implement the plane detection algorithm, the parameters used in RANSAC, and how you check the quality of the plane. For each image, label all the planes you detected with different colors and display a figure of the labeled image and the number of points in the planes. Here is a sample output of a detected plane.

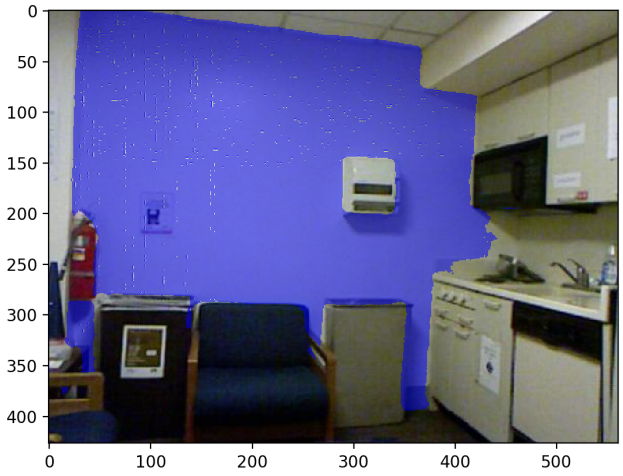

Figure 5: Fitting a plane from a point cloud to an image.

5 Bonus points [10pts]

- (2pts) Take one picture of a building (for example Bob and Betty Beyster Building) in the frontal/fronto-parallel view (i.e., so the image plane is parallel to the building facade). Then mark a rectangle on top of it. Render some patterns/object on the building from the frontal view by computing a similarity transform (recall that if the plane is parallel to the image plane, projection is just scaling). Then take a picture from another view and find a transformation to the fronto-parallel view. Warp the pattern or object from the fronto-parallel view to non-fronto-parallel view. Add the test images and the resulting image to your report and describe your algorithm. (1pt extra credit for the most creative rendering)

- (up to 3pts) 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. 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 receive full credit, these images should have obvious difference. 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. Add the test images and the resulting stitched image to your report. Also, describe your algorithm.

- (up to 5pts) Fix the artifacts of boundary in panoramas. Learn about and experiment with image blending techniques and panorama mapping techniques (cylindrical or spherical). One method is to render the image with multi-band blending. Add the resulting image to your report. Also, describe your algorithm.

Python Environment We are using Python 3.7 for this course. You can find references for the Python standard library here: https://docs.python.org/3.7/library/index.html. To make your life easier, we **recommend** you to install the latest Anaconda for Python 3.7 (https://www.anaconda.com/download/). This is a Python package manager that includes most of the modules you need for this course.

We will make use of the following packages extensively in this course:

- Numpy (https://docs.scipy.org/doc/numpy-dev/user/quickstart.html).
- Matplotlib (http://matplotlib.org/users/pyplot_tutorial.html).
- OpenCV (https://opencv.org/). Especially, we'll use OpenCV 3.4 in this homework. To install it, run `conda install -c menpo opencv`.
- Open3D (http://www.open3d.org/). We'll use the latest Open3D in part 4 to process point cloud. To install it, run `conda install -c open3d-admin open3d`.

References

- Recognising Panoramas: http://matthewalunbrown.com/papers/iccv2003.pdf.
- Open3D: http://www.open3d.org
- NYU Depth Dataset V2: https://cs.nyu.edu/ silberman/datasets/nyu_depth_v2.html
- SUN RGB-D: http://rgbd.cs.princeton.edu

Acknowledgement

Part of the homework is taken from the class CS543/ECE549 at University of Illinois Urbana-Champaign by Svetlana Lazebnik. We are grateful for her generosity. Please feel free to similarly re-use our problems while similarly crediting us.