# EECS 442 Computer Vision: Homework 3

## Instructions

- This homework is **due at 11:59:59 p.m. on Friday February 28th, 2020**.
- 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.
     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/.
  2. **To Canvas:** a `zip` file including a single directory named after your uniqname, containing all your code (in `.py` format) and files specified in questions with **Submit**. Remember to use the script `check_submission.py` (available at https://github.com/eecs442/utils) to check the format before you submit.
- The write-up must be an electronic version. **No handwriting, including plotting questions.** LaTeX is recommended but not mandatory.

## 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.
  1. To install it on your computer, run `conda install -c menpo opencv`.
  2. To use OpenCV 3.4 on Colab, run the following codes before `import cv2`:
     - `!pip uninstall opencv-python`
     - `!pip install --force-reinstall opencv-python==3.4.2.16`
     - `!pip install --force-reinstall opencv-contrib-python==3.4.2.16`

## 1 RANSAC [30 pts]

### 1.1 Fitting a Line [10 pts]

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

1. (3 pts) 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?
2. (3 pts) 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?
3. (4 pts) In the previous settings, to exceed a probability level of 95% for success, how many trials should we attempt to fit putative models?

### 1.2 Fitting Transformations [20 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. (1 pt) Recall that a matrix $\mathbf{M} \in \mathbb{R}^{2\times 2}$ is a linear transformation: $\mathbb{R}^2 \rightarrow \mathbb{R}^2$. How many degrees of freedom does $\mathbf{M}$ have? How many samples are required to find $\mathbf{M}$?
2. (2 pts) 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:

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

   where $\mathbf{m}$ is some vector that has all the parameters of $\mathbf{M}$. **Write** out the form of $\mathbf{A}$.
3. (3 pts) Use `numpy.load()` to load `p1/transform.npy`. Each row contains two points $x$ and $y$, represented in the form $[\mathbf{x}_i^T, \mathbf{y}_i^T]_{1\times 4}$. Then 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 $\text{argmin}_{\mathbf{v}} \|\mathbf{A}\mathbf{v} - \mathbf{b}\|^2$. **Report** $\mathbf{S}$ and $\mathbf{t}$.
4. (2 pts) Plot the points $\mathbf{x}$, $\mathbf{y}$ and $\hat{\mathbf{y}} = \mathbf{S}\mathbf{x} + \mathbf{t}$ in one figure with different colors. **Display** the figure.
5. (2 pts) Explain how you transform the points in words based on $\mathbf{S}$ and $\mathbf{t}$. Is the fitting result good?
6. (4 pts) We have 8 cases of homography transformation of letter M. In each case, there are two sets of 2D points X and Y, which are represented in the same format as 1.2.3 (a N$\times$4 array with each row in the form $[\mathbf{x}_i^T, \mathbf{y}_i^T]_{1\times 4}$, we have loaded the data for you). For each case, 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 $\mathbf{H}$. **Report** matrix $\mathbf{H}$ of each case.

7. (4 pts) Visualize the original points, the target points and the points after homography transformation in one figure (three separate images in one figure). **Display** the figure of each case.

8. (2 pts) Discuss the transformations your observed in visualization with the homography matrix $\mathbf{H}$. Do they make sense to you?

## 2 Image Stitching [55 pts]

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 is derived from an assignment developed by Svetlana Lazebnik at UIUC). For this part, you will be working with the following images.

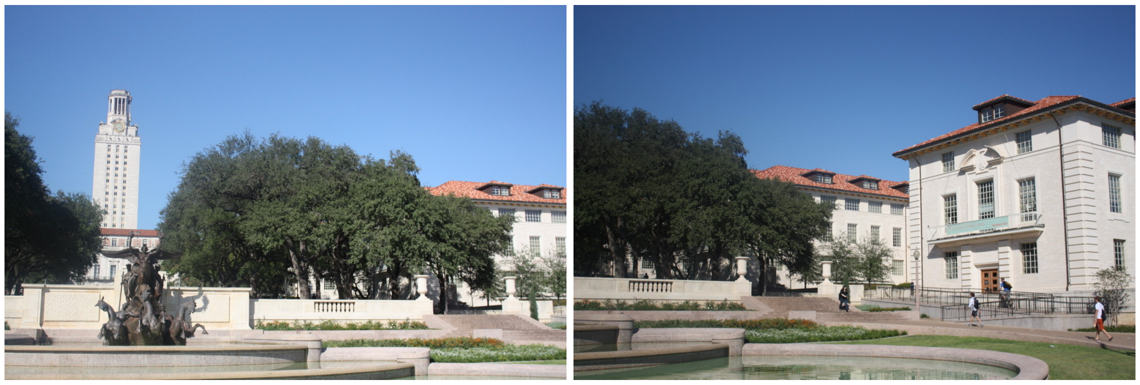

Figure 1: Input of Image Stitching.

**Here are the instructions:**

1. (3 pts) Load both images: `p2/uttower_left.jpg` and `p2/uttower_right.jpg`. Convert them to double and to grayscale. **Display** the grayscale images.

2. (4 pts) Use SIFT/SURF descriptors in `OpenCV` to detect feature points in both images. **Display** both the images along with the feature points.

3. (6 pts) 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.

4. (10 pts) 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.

5. (12 pts) 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.drawMatches` 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, it's a good idea to use them to debug your implementation.

6. (8 pts) 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.

7. (8 pts) 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:

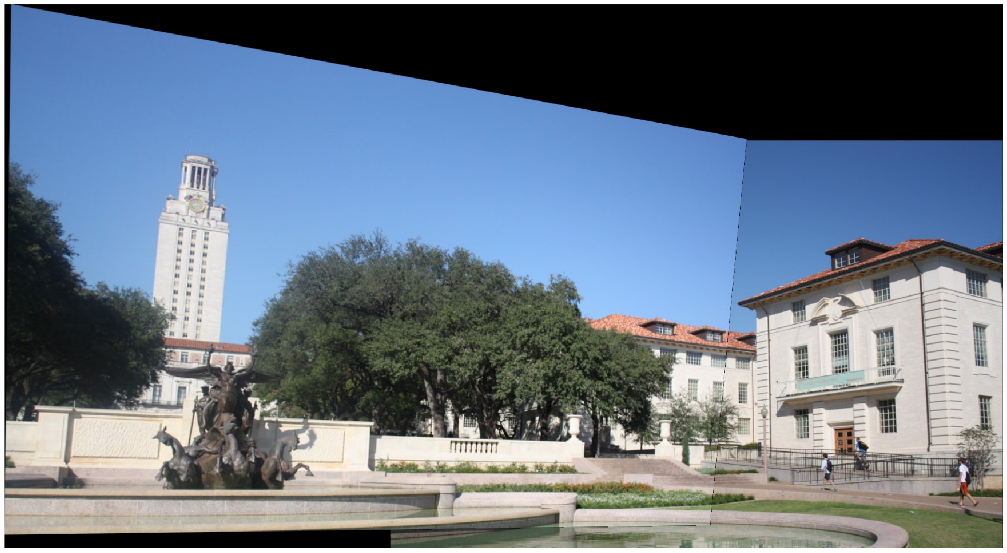

Figure 2: Sample output of Image Stitching.

8. (4 pts) Try to run your code on `p2/bbb_left.jpg` and `p2/bbb_right.jpg`. Display the detected feature points, the matching result, the inliers after RANSAC, and the stitched image.

## 3 Homography Rendering [15 pts]

In this section, you will add some patterns/objects to a specific area of a picture with a non-frontal view. An example of inputs is shown as followings.

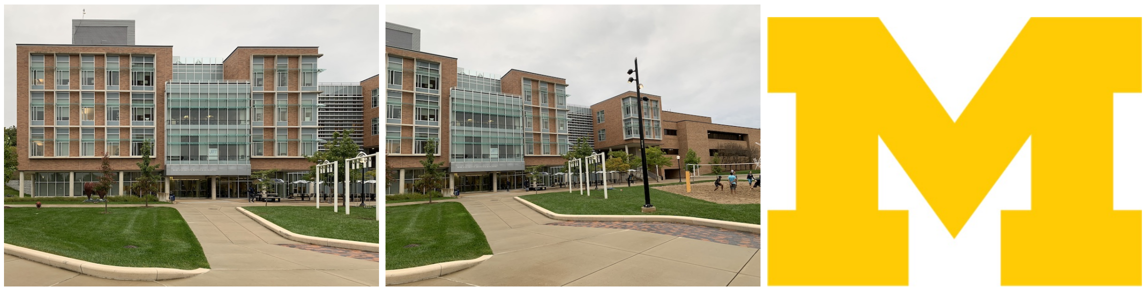

Figure 3: Inputs of homography render (frontal view, non-frontal view, object).

1. (5 pts) 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 (e.g., a letter M) 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). **Report** your transformation and **Display** the pictures. An example of the output looks like this:

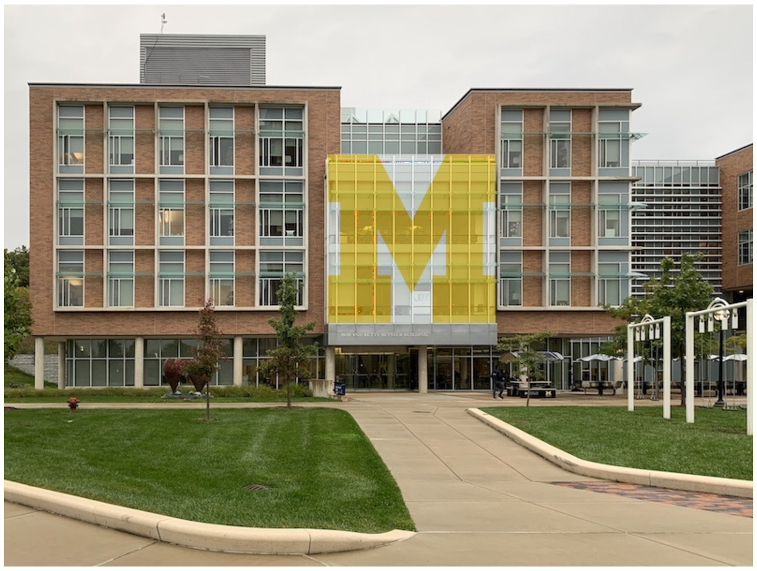

Figure 4: Render object on frontal view

2. (5 pts) Then take a picture from another view and find a transformation to the fronto-parallel view you got in the previous problem. **Display** the pictures and **Report** your transformation.

3. (5 pts) Warp the pattern or object from the fronto-parallel view to non-fronto-parallel view. **Display** the resulting image to your report and describe your algorithm. An example of the output looks like this:

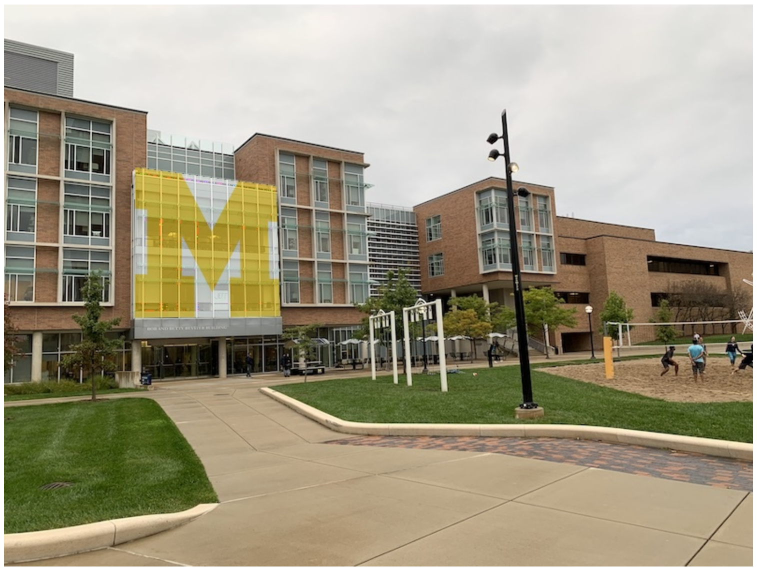

Figure 5: Render object on non-frontal view

## Canvas Submission Checklist

In the `zip` file you submit to Canvas, the directory named after your uniqname should include the following files:

- `main.py`
- `transform.png`
- `case_(0-7).png`
- `uttower.jpg`
- `bbb.jpg`
- `(uttower,bbb)_(left,right)_gray.jpg`: the left and right of uttower and bbb images in gray scale
- `(uttower,bbb)_(left,right)_descriptor.jpg`: the left and right of uttower and bbb images along with the feature points
- `(uttower,bbb)_matches.jpg`: the matches of left and right in uttower and bbb images
- `frontal.jpg`: one picture of a building in the frontal/fronto-parallel view
- `patterns.jpg`: some patterns/objects
- `render_frontal.jpg`: rendered image with pattern on frontal image
- `non_frontal.jpg`: one picture of a building from another view
- `render_non_frontal.jpg`: rendered image with pattern on nonfrontal image

The rest should be all included in your `pdf` report submitted to Gradescope.

## References

Recognising Panoramas: http://matthewalunbrown.com/papers/iccv2003.pdf.

## 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.