COMP5421 Computer Vision, Spring 2022

Homework Assignment 3
Homographies & RANSAC & Image Stitching

Total Points: 125

Instructions/Hints

1. Please pack your system into a single file named `<ustlogin-id>.zip`, see the complete submission checklist in the overview.
2. All questions marked with a **Q** require a submission.
3. For the implementation part, please stick to the headers, variable names, and file conventions provided.
4. **Start early!** It will make you more flexible to finish this homework.
5. **Attempt to verify your implementation as you proceed**: If you don't verify that your implementation is correct on toy examples, you will risk having a huge mess when you put everything together.
6. **Do not import external functions/packages other than the ones already imported in the files**: The current imported functions and packages are enough for you to complete this assignment.

Introduction

1 Planar Homographies: Theory (20 pts)

Suppose we have two cameras looking at a common plane in 3D space. Any 3D point $\mathbf{w}$ on this plane generates a projected 2D point located at $\widetilde{\mathbf{u}} = [u_1, v_1, 1]^T$ on the first camera and $\widetilde{\mathbf{x}} = [x_2, y_2, 1]^T$ on the second camera. Since $\mathbf{w}$ is confined to a plane, we expect that there is a relationship between $\widetilde{\mathbf{u}}$ and $\widetilde{\mathbf{x}}$.

In particular, there exists a common 3×3 matrix $\mathbf{H}$, so that for any $\mathbf{w}$, the following condition holds:

$$\lambda \widetilde{\mathbf{x}} = \mathbf{H}\widetilde{\mathbf{u}} \tag{1}$$

where λ is an arbitrary scalar weighting. We call this relationship a *planar homography*. Recall that the $\widetilde{}$ operator implies a vector is employing homogenous coordinates such $\widetilde{\mathbf{x}} = [\mathbf{x}, 1]^T$. It turns out this homography relationship is also true for cameras that are related by pure rotation without the planar constraint.

Q1.1 We have a set of N 2D homogeneous coordinates $\{\widetilde{\mathbf{x}}_1, \cdots, \widetilde{\mathbf{x}}_N, \}$ taken at one camera view, and $\{\widetilde{\mathbf{u}}_1, \cdots, \widetilde{\mathbf{u}}_N, \}$ taken at another. Suppose we know there exists an unknown homography $\mathbf{H}$ between the two views such that,

$$\lambda_n \widetilde{\mathbf{x}}_n = \mathbf{H}\widetilde{\mathbf{u}}_n \qquad \text{for} \;\; n = 1 : N \tag{2}$$

where again λ_n is an arbitrary scalar weighting.

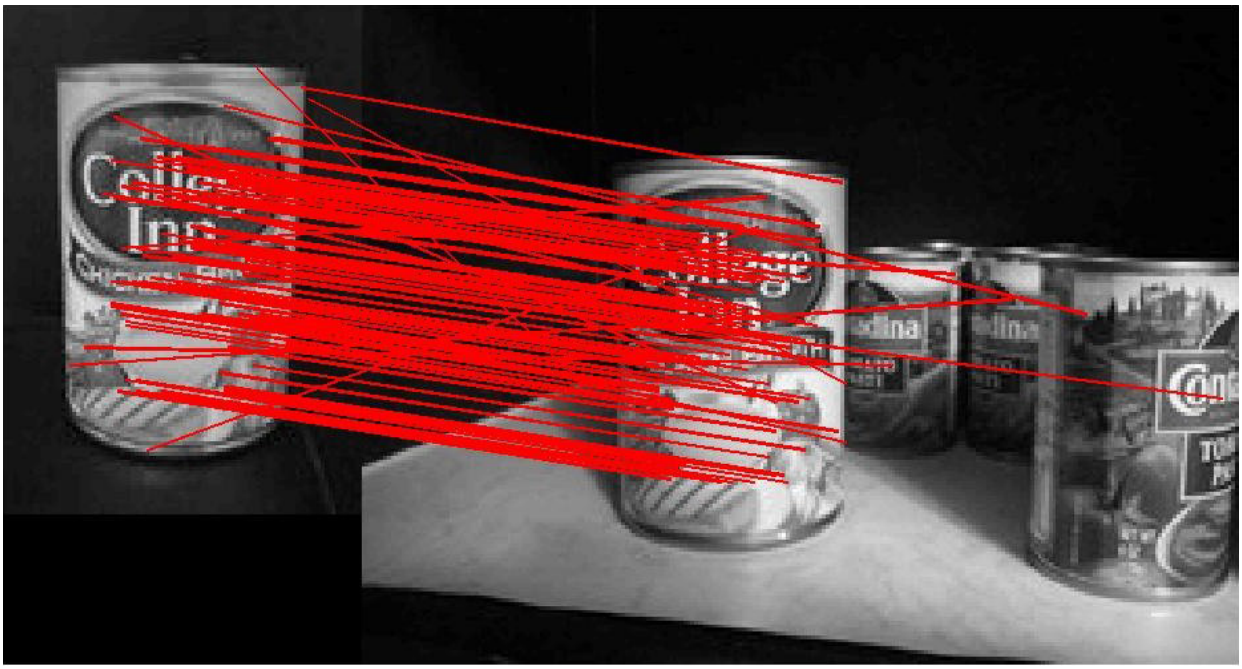


Figure 1: Example of BRIFF matches for `model_chickenbroth.jpg` and `chickenbroth_01.jpg`.

(a) Given the N correspondences across the two views and using Equation 2, derive a set of $2N$ independent linear equations in the form:

$$\mathbf{A}\mathbf{h} = \mathbf{0} \tag{3}$$

where $\mathbf{h}$ is a vector of the elements of $\mathbf{H}$ and $\mathbf{A}$ is a matrix composed of elements derived from the point coordinates. Write out an expression for $\mathbf{A}$.

Hint: Start by writing out Equation 2 in terms of the elements of $\mathbf{H}$ and the homogeneous coordinates $\tilde{\mathbf{u}}_n$ and $\tilde{\mathbf{x}}_n$.

(b) How many elements are there in $\mathbf{h}$?

(c) How many point pairs (correspondences) are required to solve this system?

Hint: How many degrees of freedom are in $\mathbf{H}$? How much information does each point correspondence give?

(d) Show how to estimate the elements in $\mathbf{h}$ to find a solution to minimize this homogeneous linear least squares system. Step us through this procedure.

Hint: Use the Rayleigh quotient theorem (homogeneous least squares).

2 Planar Homographies: Implementation (15 pts)

Note: Implement the method in `planarH.py`.

Now that we have derived how to find $\mathbf{H}$ mathematically in **Q 1.1**, we will implement in this section.

Q 2.1 (15pts) Implement the function

```
H2to1 = computeH(X1,X2)
```

Inputs: `X1` and `X2` should be $2 \times N$ matrices of corresponding $(x, y)^T$ coordinates between two images.
Outputs: `H2to1` should be a 3×3 matrix encoding the homography that best matches the linear equation derived above for Equation 2 (in the least squares sense). *Hint*: Remember that a homography is only determined up to scale. The `numpy.linalg` function `eigh()` or `svd()` will be useful. Note that this function can be written without an explicit for-loop over the data points.

3 RANSAC (25 pts)

Note: Implement the method in `planarH.py`.

The least squares method you implemented for computing homographies is not robust to outliers. If all the correspondences are good matches, this is not a problem. But even a single false correspondence can completely throw off the homography estimation. When correspondences are determined automatically (using feature matching for instance), some mismatches in a set of point correspondences are almost certain. RANSAC (Random Sample Consensus can be used to fit models robustly in the presence of outliers.

Q 3.1 (25pts): Write a function that uses RANSAC to compute homographies automatically between two images:

```
bestH = ransacH(matches, locs1, locs2, nIter, tol)
```

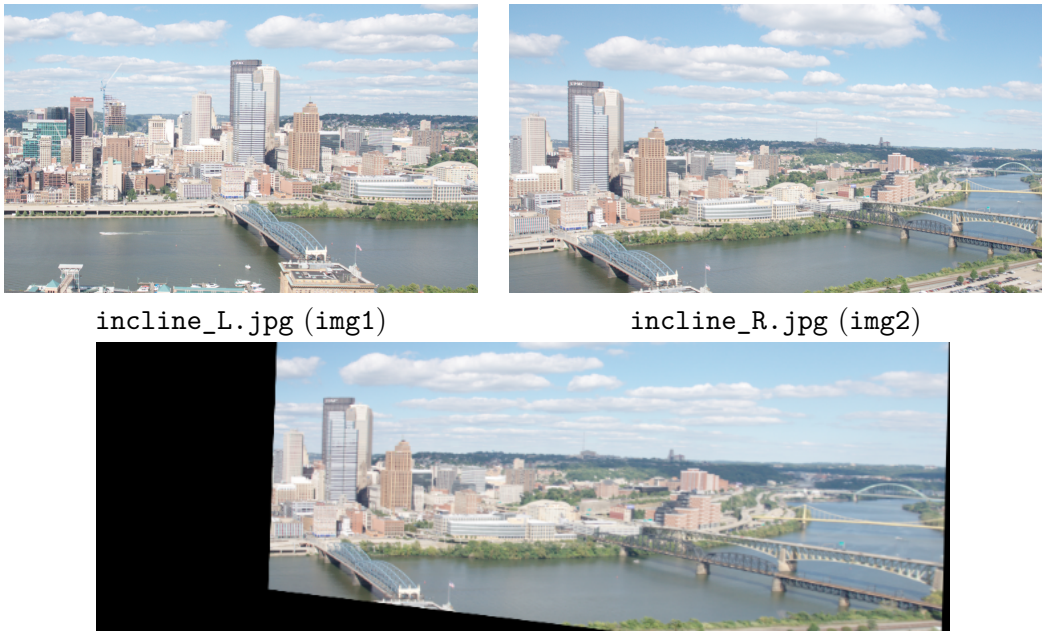
The inputs and output of this function should be as follows:

- Inputs: `locs1` and `locs2` are matrices specifying point locations in each of the images and matches is a matrix specifying matches between these two sets of point locations. These matrices are formatted identically to the output of the already provided `briefMatch` function in `BRIEF.py` to perform feature matching. You need to check into it to find out how the `briefMatch` function can be used for images. An example of BRIFF matches is shown in Figure 1.
- Algorithm Input Parameters: `nIter` is the number of iterations to run RANSAC for, tol is the tolerance value for considering a point to be an inlier. Define your function so that these two parameters have reasonable default values.
- Outputs: bestH should be the homography model with the most inliers found during RANSAC.

4 Stitching it together: Panoramas (40 pts)

NOTE: All the functions to implement here are in `panoramas.py`.

We can also use homographies to create a panorama image from multiple views of the same scene. This is possible for example when there is no camera translation between the views (e.g., only rotation about the camera center). First, you will generate panoramas using matched point correspondences between images using the BRIEF matching provided. **We will assume that there is no error in your matched point correspondences between images (Although there might be some errors, and even small errors can have drastic impacts).** In the next section you will extend the technique to deal with the noisy keypoint matches.



`incline_L.jpg` (`img1`) `incline_R.jpg` (`img2`)

`img2` warped to `img1`'s frame

Figure 2: Example output for **Q 4.1**: Original images `img1` and `img2` (left and center) and `img2` warped to fit `img1` (right). Notice that the warped image clips out of the image. We will fix this in **Q 4.2**.

You will need to use the perspective warping function from OpenCV:
`warp_im = cv2.warpPerspective(im, H, out size)`, which warps image `im` using the homography transform `H`. The pixels in `warp_im` are sampled at coordinates in the rectangle $(0, 0)$ to (`out_size[0]-1, out_size[1]-1`). The coordinates of the pixels in the source image are taken to be $(0, 0)$ to (`im.shape[1]-1, im.shape[0]-1`) and transformed according to `H`.

Q 4.1 (15pts) In this problem you will implement and use the function:

```
panoImg = imageStitching(img1, img2, H2to1)
```

on two images from the Dusquesne incline. This function accepts two images and the output from the homography estimation function. This function:

(a) Warps `img2` into `img1`'s reference frame using the aforementioned perspective warping function

(b) Blends `img1` and warped `img2` and outputs the panorama image.

For this problem, use the provided images `incline_L` as `img1` and `incline_R` as `img2`. The point correspondences `pts` are generated by your BRIEF descriptor matching.

Apply your `ransacH()` to these correspondences to compute `H2to1`, which is the homography from `incline_R` onto `incline_L`. Then apply this homography to `incline_R` using `warpH()`.

Note: Since the warped image will be translated to the right, you will need a larger target image.

Visualize the warped image and save this figure as `results/4_1.jpg` using OpenCV's `cv2.imwrite()` function and save only `H2to1` as `results/q4_1.npy` using Numpy `np.save()` function.

Q 4.2 (15pts) Notice how the output from Q 4.1 is clipped at the edges? We will fix this now. Implement a function

```
[panoImg] = imageStitching noClip(img1, img2, H2to1)
```

that takes in the same input types and produces the same outputs as in Q 4.1.

To prevent clipping at the edges, we instead need to warp *both* image 1 and image 2 into a common third reference frame in which we can display both images without any clipping. Specifically, we want to find a matrix M that *only* does scaling and translation such that:

```
warp_im1 = cv2.warpPerspective(im1, M, out size)
warp im2 = cv2.warpPerspective(im2, np.matmul(M,H2to1), out size)
```

This produces warped images in a common reference frame where all points in im1 and im2 are visible. To do this, we will only take as input either the width or height of out size and compute the other one based on the given images such that the warped images are not squeezed or elongated in the panorama image. For now, assume we only take as input the width of the image (i.e., `out_size[0]`) and should therefore compute the correct height(i.e., `out_size[1]`).

Hint: The computation will be done in terms of `H2to1` and the extreme points (corners) of the two images. Make sure M includes only scale (find the aspect ratio of the full-sized panorama image) and translation.

Again, pass `incline_L` as `img1` and `incline_R` as `img2`. Save the resulting panorama in `results/q4_2 pan.jpg`.

Q 4.3 (10pts): You now have all the tools you need to automatically generate panoramas. Write a function that accepts two images as input, computes keypoints and descriptors for both the images, finds putative feature correspondences by matching keypoint descriptors, estimates a homography using RANSAC and then warps one of the images with the homography so that they are aligned and then overlays them.

```
im3 = generatePanorama(im1, im2)
```

Run your code on the image pair `data/incline_L.jpg`, `data/incline_R.jpg`. However during debugging, try on scaled down versions of the images to keep running time low. Save the resulting panorama on the full sized images as `results/q4_3.jpg`. (see Figure 3 for example output). Include the figure in your writeup.

5 Augmented Reality (25 pts)

Homographies are also really useful for Augmented Reality (AR) applications. For this AR exercise we will be using the image in Figure 4 with the following data points:

$$\mathbf{W} = \begin{bmatrix} 0.0 & 18.2 & 18.2 & 0.0 \\ 0.0 & 0.0 & 26.0 & 26.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} \tag{4}$$

Figure 3: Final panorama view. With homography estimated using RANSAC.

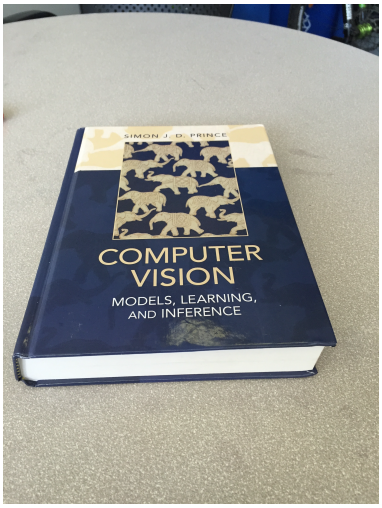


Figure 4: Image file we will be using in this exercise, where we provide you with the 3D planar points of corners of the book (**W**) the 2D projected points (**D**) in the image, and the camera intrinsic parameters (**K**).

are the 3D planar points of the textbook (in cm);

$$\mathbf{D} = \begin{bmatrix} 483 & 1704 & 2175 & 67 \\ 810 & 781 & 2217 & 2286 \end{bmatrix} \tag{5}$$

are the corresponding projected points; and

$$\mathbf{K} = \begin{bmatrix} 3043.72 & 0.0 & 1196.00 \\ 0.0 & 3043.72 & 1604.00 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{6}$$

contains the intrinsics matrix for the image. The relationship between 3D planar and 2D projected point can be defined through,

$$\lambda_n \widetilde{\mathbf{x}}_n = \mathbf{K} \begin{bmatrix} \mathbf{R} \mid bt \end{bmatrix} \widetilde{\mathbf{w}}_n \tag{7}$$

where $\mathbf{x}_n$ and $\mathbf{w}_n$ relate to the n-th column vector of $\mathbf{D}$ and $\mathbf{W}$.

Q5.1 (10pts) Using the method from Section 15.4.1 (pp. 335–337) of Prince's textbook, implement the function

```
R,t = compute extrinsics(K, H)
```

where `K` contains the intrinsic parameters and `H` contains the estimated homography. As discussed in class the extrinsic parameters refer to the relative 3D rotation $\mathbf{R}$ and translation $\mathbf{t}$ of the camera between views. *Remember*: Prince uses the notation $\mathbf{\Omega}$ and τ to refer the extrinsic parameters $\mathbf{R}$ and $\mathbf{t}$ respectively. They also employ the notation $\mathbf{\Lambda}$ to refer to the intrinsics matrix $\mathbf{K}$.

Q5.2 (15pts) Next implement the function

```
X=project extrinsics(K, W, R, t)
```

that will apply a pinhole camera projection (described in Equation 7) of any set of 3D points using a pre-deterimined set of extrinsics and intrinsics. Use this function then project the set of 3D points in the file sphere.txt (which has been fixed to have the same radius as a tennis ball - of 6.858l) onto the image `prince_book.jpg`. Specifically, attempt to place the bottom of the ball in the middle of the "o" in "Computer" text of the book title. Use matplotlib's `plot` function to display the points of the sphere in yellow. For this question disregard self occlusion (i.e. points on the projected sphere occluding other points). Capture the image and include it in your written response.

6 Submission and Deliverables

The assignment should be submitted to Canvas as a zip file named `<ustlogin-id>.zip` with a pdf file named `<ustlogin-id>_hw3.pdf`. You can use check `files.py` to ensure you have all required files ready before submission. The zip file should contain:

- A folder `code` containing all the `.py` files you were asked to write.
- PDF `<ustlogin-id>_hw3.pdf`.
- A folder `results` containing all the `.npy` files you were asked to generate.

The PDF `<ustlogin-id>_hw3.pdf` should contain the results, explanations and images asked for in the assignment along with to the answers to the questions on homographies. Submit all the code needed to make your panorama generator run. Make sure all the `.py` files that need to run are accessible from the `code` folder without any editing of the path variable. You may leave the `data` folder in your submission, but it is not needed.

Note: Missing to follow the structure will incur huge penalty in scores!

Appendix: Image Blending

Note: This section is not for credit and is for information purposes only.

For overlapping pixels, it is common to blend the values of both images. You can simply average the values but that will leave a seam at the edges of the overlapping images. Alternatively, you can obtain a blending value for each image that fades one image into the other. To do this, first create a mask like this for each image you wish to blend:

```
mask = np.zeros((im.shape[0], im.shape[1]))
mask[0,:] = 1
mask[-1,:] = 1
mask[:,0] = 1
mask[:,-1] = 1
mask = scipy.ndimage.morphology.distance_transform_edt(1-mask)
mask = mask/mask.max(0)
```

The function `distance_transform_edt` computes the distance transform of the binarized input image, so this mask will be zero at the borders and 1 at the center of the image. You can warp this mask just as you warped your images. How would you use the mask weights to compute a linear combination of the pixels in the overlap region? Your function should behave well where one or both of the blending constants are zero.

References

1. P. Burt and E. Adelson. The Laplacian Pyramid as a Compact Image Code. *IEEE Transactions on Communications*, 31(4):532–540, April 1983.

2. Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. BRIEF : Binary Robust Independent Elementary Features.

3. David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.