16-720B Computer Vision: Homework 2

Feature Descriptors, Homographies & RANSAC

Due: October 10 at 11:59pm

[redacted] (AndrewID: [redacted])

1 Keypoint Detector

1.1 Gaussian Pyramid

1.2 The DoG Pyramid

Q 1.2: Function

```
DoGPyramid, DoGLevels = createDoGPyramid(GaussianPyramid, levels)
```

is completed to construct a Difference of Gaussian pyramid.
The output DoG pyramid is shown as Fig. 1

Figure 1: DoG pyramid for model `chickenbroth.jpg`

1.3 Edge Suppression

Q 1.3: Function

```
PrincipalCurvature = computePrincipalCurvature(DoGPyramid)
```

is completed to calculate the principle curvature ratio for each point of each layer in Difference of Gaussian pyramid.

1.4 Detecting Extrema

Q 1.4: Function

```
locsDoG = getLocalExtrema(DoGPyramid, DoGLevels, PrincipalCurvature,
                          th_contrast, th_r)
```

is completed to find the local exrema in both scale and space after removing points with small local exrema value and edge-like points.

1.5 Putting it together

Q 1.5: Function

```
locsDoG, GaussianPyramid = DoGdetector(im, sigma0, k, levels,
                          th_contrast, th_r)
```

is completed to combine all the steps to construct a DoG detector.
The image with detected keypoints is shown as Fig. 2

Figure 2: examples with detected keypoints.

2 BRIEF Descriptor

2.1 Creating a Set of BRIEF Tests

Q 2.1: Function

```
compareX, compareY = makeTestPattern(patchWidth, nbits)
```

is completed to generate the linear indices into the image patch.

Here, the method $(\mathbf{X}, \mathbf{Y}) \sim i.i.d.$ Gaussian$(0, \frac{1}{25}S^2)$ is implemented. The tests are sampled from an isotropic Gaussian distribution.

Given parameters `patchWidth` $= 9$ and `nbits` $= 256$, result has been saved in file `testPattern.npy`.

2.2 Compute the BRIEF Descriptor

Q 2.2: Function

```
locs,desc = computeBrief(im, GaussianPyramid, locsDoG, k, levels,
                         compareX, compareY)
```

is completed to compute the BRIEF descriptor for the detected keypoints.

2.3 Putting it all Together

Q 2.3: Function

```
locs, desc = briefLite(im)
```

is completed to combine all the necessary steps to extract the BRIEF descriptor for the input image.

2.4 Check Point: Descriptor Matching

Q 2.4:

For `chickenbroth` images, the keypoints detection and match results are shown as Fig. 3

For `incline` images, the keypoints detection and match result is shown as Fig. 4

For computer vision textbook cover page image (pf_scan_scaled.jpg), the keypoints detection and match results are shown as Fig. 5

We can see from these results that for `chickenbroth` images, the matching performs well in all cases. Some wrong matches when there are other similar objects besides the target object.

For `chickenbroth` images, the matching also performs well since there is no rotation of these two images. With only translation, the extracted features would be similar and would be easier to match.

Compared to `chickenbroth` images and `chickenbroth` images, the matching between the computer vision textbook cover page images performs worse. It is because that there are rotations of the textbook. When there is rotation, the extracted BRIEF features might be different in different images for the corresponding point by using the same test patterns when calculating the feature. On the other hand, some other books that with similar texture also would cause wrong matching, just like in the third matching result. Among these five matching result, the last one is the best, since its rotation is small and there is no other similar books around the textbook.

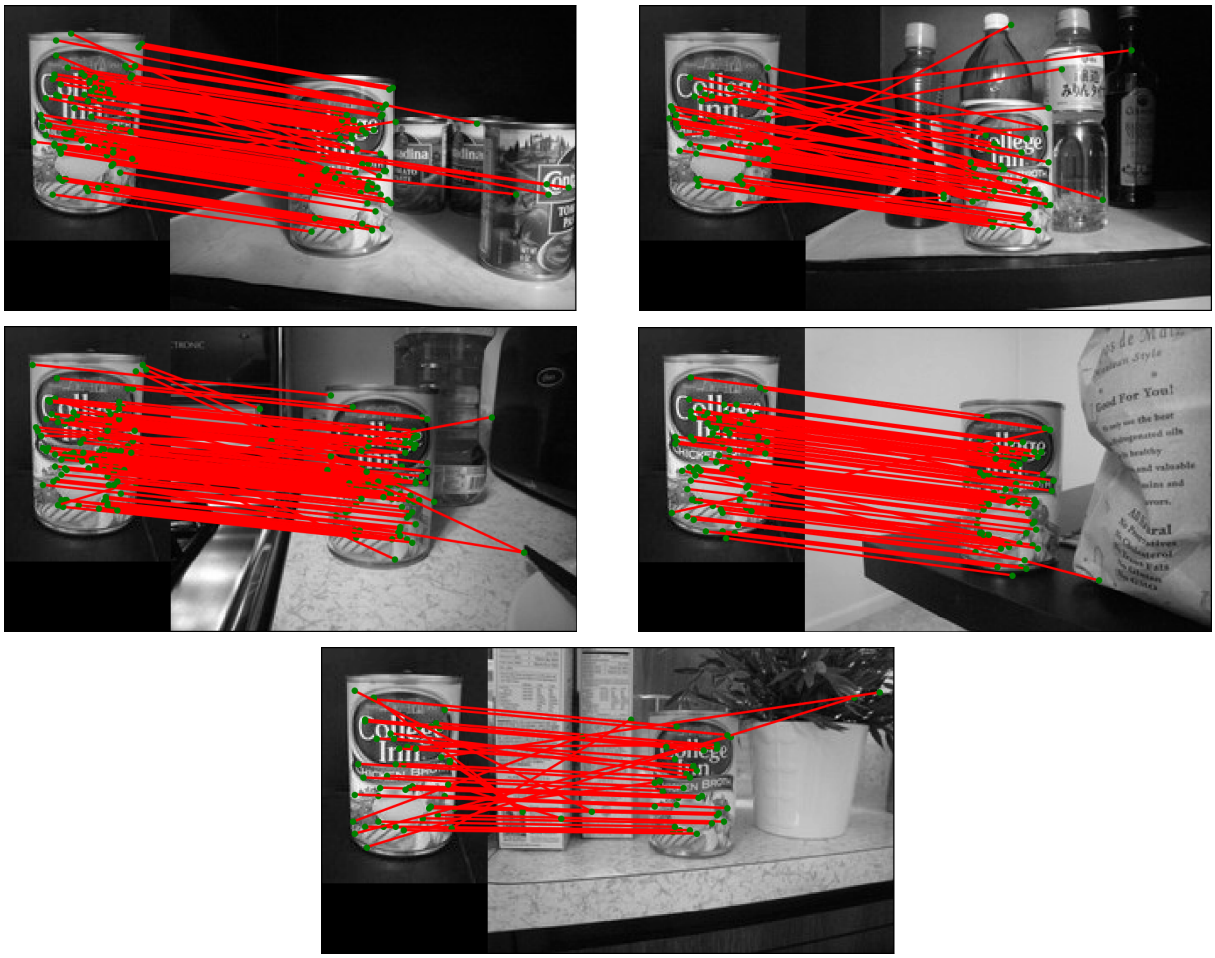

Figure 3: Keypoints detection and match result for **chickenbroth** images

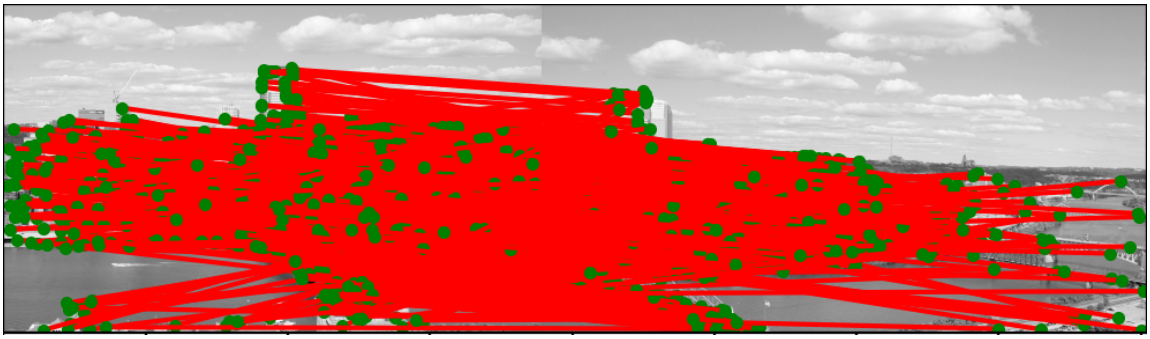

Figure 4: Keypoints detection and match result for **incline** images

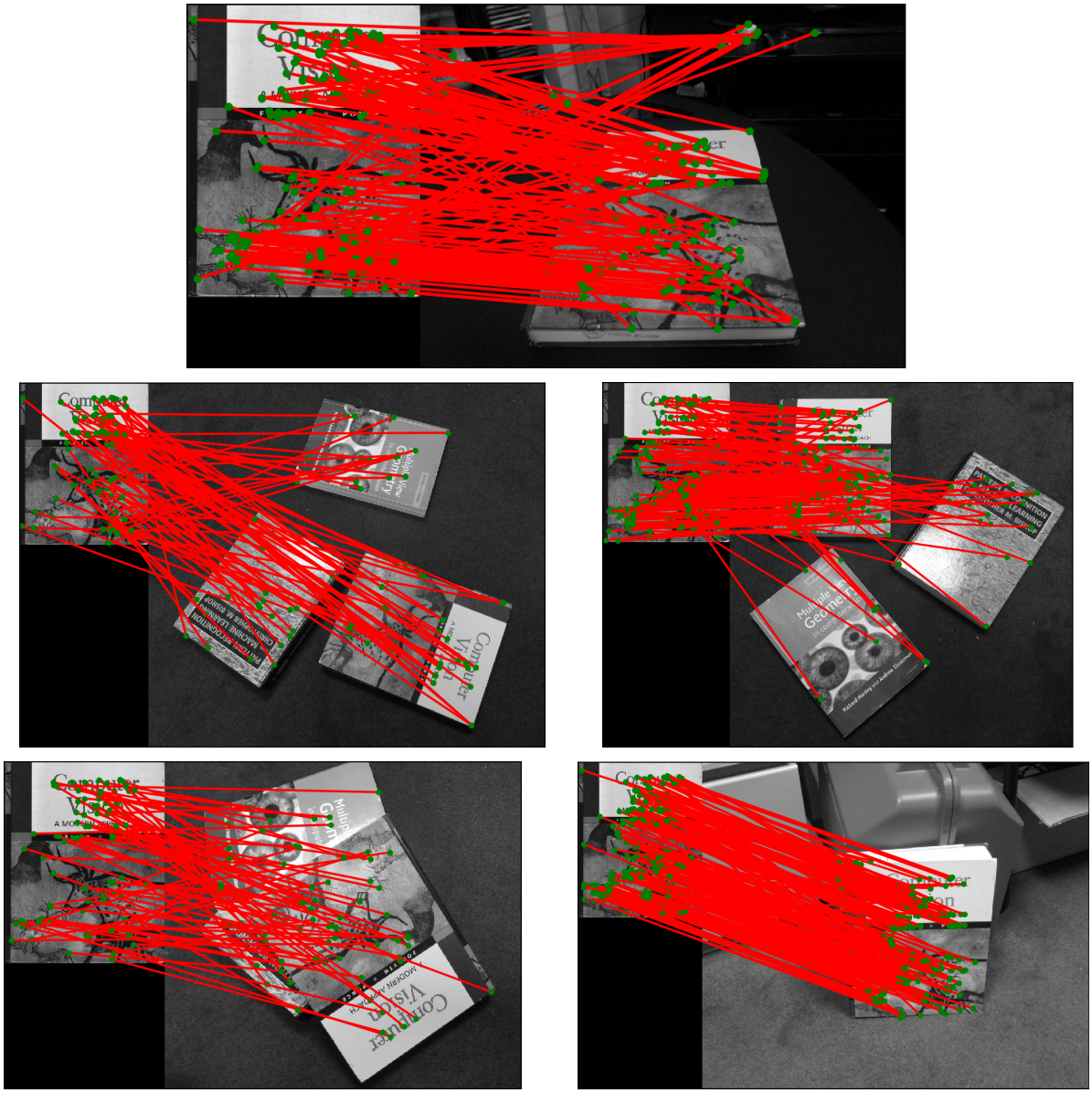

Figure 5: Keypoints detection and match result for computer vision textbook cover page image.

2.5 BRIEF and rotations

Q 2.5:

Script `briefRotTest.py` is completed to construct a bar graph showing rotation angle vs the number of correct matches.

The bar graph is shown as Fig. 6

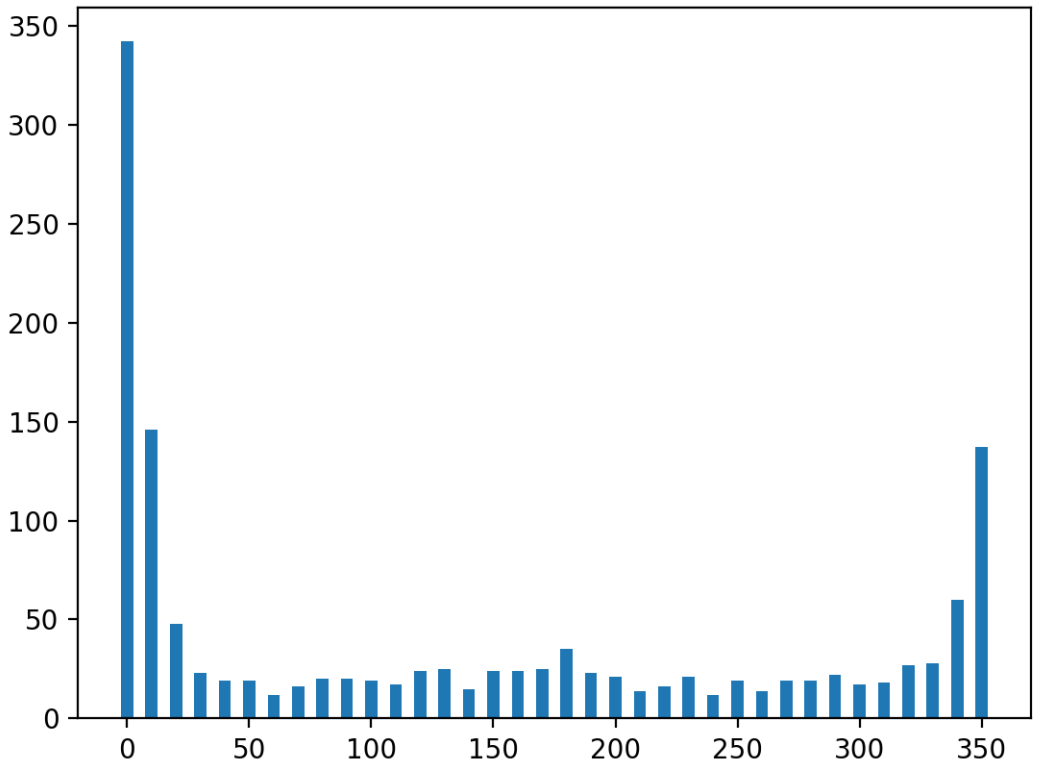

Figure 6: Rotation angle vs the number of correct matches

We can see from the graph that when there is rotation, the number of matched points decreases drastically. When the image is only slightly rotated, like with 10 degrees and 350 degrees (can be regarded as -10 degrees), the performance is relatively better than those with larger degrees. The reason could be that the features of the same point in the original image and in the rotated image might be quite different with rotation, since the BRIEF features are extracted around the keypoints using the same pattern of point pairs selection. So the number of matched points would decrease greatly.

3 Planar Homographies: Theory

Q 3.1:

1.
$$\lambda_n \widetilde{\mathbf{x}}_n = \mathbf{H}\widetilde{\mathbf{u}}_n, \quad for\ n = 1:N$$
$$\widetilde{\mathbf{x}}_i = [x_i, y_i, 1]^T, \quad \widetilde{\mathbf{u}}_n = [u_i, v_i, 1]^T$$
$$\mathbf{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

So we can get that:
$$x_i = \frac{h_{11}u_i + h_{12}v_i + h_{13}}{h_{31}u_i + h_{32}v_i + h_{33}}$$
$$y_i = \frac{h_{21}u_i + h_{22}v_i + h_{23}}{h_{31}u_i + h_{32}v_i + h_{33}}$$

Further, we can obtain equations:
$$u_i h_{11} + v_i h_{12} + h_{13} - x_i u_i h_{31} - x_i v_i h_{32} - x_i h_{33} = 0$$
$$-u_i h_{21} - v_i h_{22} - h_{23} + y_i u_i h_{31} + y_i v_i h_{32} + y_i h_{33} = 0$$

With N correspondences across the two views, we can get $2N$ equations and can be presented in form $\mathbf{Ah} = 0$ as below:
$$\mathbf{Ah} = \begin{bmatrix} u_1 & v_1 & 1 & 0 & 0 & 0 & -x_1u_1 & -x_1v_1 & -x_1 \\ 0 & 0 & 0 & -u_1 & -v_1 & -1 & y_1u_1 & y_1v_1 & y_1 \\ u_2 & v_2 & 2 & 0 & 0 & 0 & -x_2u_2 & -x_2v_2 & -x_2 \\ 0 & 0 & 0 & -u_2 & -v_2 & -1 & y_2u_2 & y_2v_2 & y_2 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ u_N & v_N & 1 & 0 & 0 & 0 & -x_Nu_N & -x_Nv_N & -x_N \\ 0 & 0 & 0 & -u_N & -v_N & -1 & y_Nu_N & y_Nv_N & y_N \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = 0$$

2. There are **9** elements in $\mathbf{h}$.

3. **4** point pairs (correspondences) are required to solve this system.

4. Apply SVD decomposition to $\mathbf{A}$, get $\mathbf{A}=\mathbf{USV}^T$, where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal, $\mathbf{S}$ is diagonal.
   Since $\mathbf{U}$ is orthonormal, we have:
$$||\mathbf{USV}^T\mathbf{h}||^2 = \mathbf{h}^T\mathbf{VS}^T\mathbf{U}^T\mathbf{USV}^T\mathbf{h} = \mathbf{h}^T\mathbf{VS}^T\mathbf{SV}^T\mathbf{h} = ||\mathbf{SV}^T\mathbf{h}||^2$$
   If we want to minimize $||\mathbf{Ah}||$, then we need to minimize $||\mathbf{SV}^T\mathbf{h}||$ according to the equation above. Let $\mathbf{y} = \mathbf{V}^T\mathbf{h}$, so the goal becomes minimizing $||\mathbf{Sy}||$ subject to $||\mathbf{y}|| = 1$. Since $\mathbf{S}$ is diagonal and the eigenvalues are sorted in descending order, let $\mathbf{y} = [0, 0, ..., 1]^T$ would minimize $||\mathbf{Sy}||$. So $\mathbf{h} = \mathbf{Vy}$ would be the last column of $\mathbf{V}$.

4 Planar Homographies: Implementation

Q 4.1: Function

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

is completed to compute the homography from `X2` TO `X1` given the correspondence.

5 RANSAC

Q 5.1: Function

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

is completed to compute homographies automatically between two images using RANSAC.

6 Stitching it together: Panoramas

Q 6.1: Function

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

is completed to generate the stitched panorama given two images and the homography.

The calculate homography is saved into `results/q6_1.npy` and is shown as below:

$$\begin{bmatrix} 1.83597564e-03 & -5.17074887e-05 & 9.98452642e-01 \\ -2.23787893e-04 & 2.47807821e-03 & -5.54535834e-02 \\ -9.91964033e-07 & 5.25455227e-08 & 2.76706333e-03 \end{bmatrix}$$

The warped image is shown as Fig.7 (the largest projected coordinates are chosen to determine the size of the target image) and the obtained panorama image is shown as Fig.8:

Figure 7: The warped image of `incline_R.jpg (img2)`.

Figure 8: The panorama image after stitching.

Q 6.2: Function

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

is completed to generate the stitched panorama without clipping.

The matrix $\mathbf{M}$ is as below:

$$\begin{bmatrix} 1. & 0. & 0. \\ 0. & 1. & 164. \\ 0. & 0. & 1. \end{bmatrix}$$

The warped results of `incline_L.jpg (img1)` and `incline_R.jpg (img2)` are shown as Fig.9

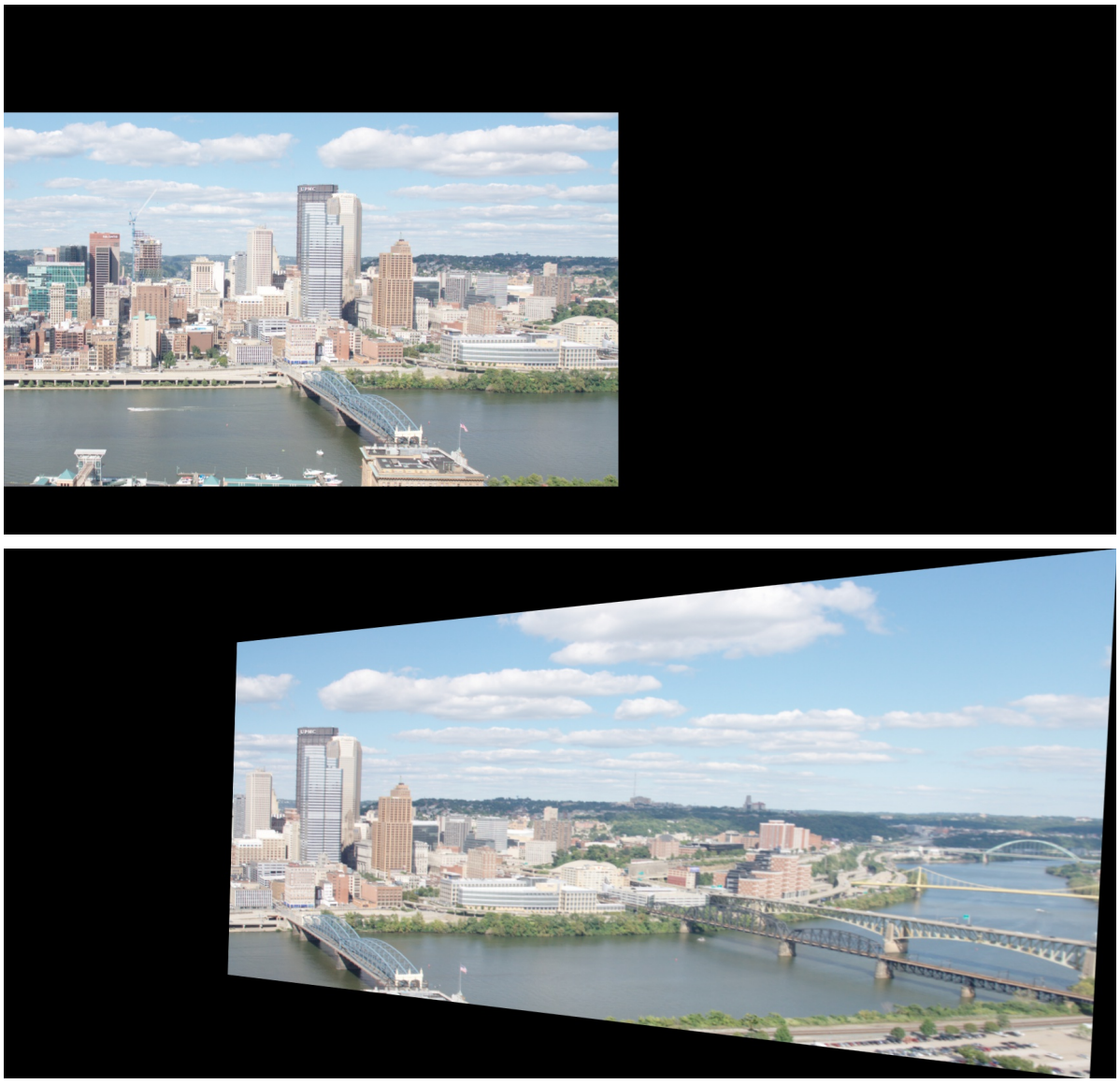

Figure 9: The warped images of `incline_L.jpg (img1)` and `incline_R.jpg (img2)`.

The obtained panorama is shown as Fig.10

Figure 10: The panorama image without clipping.

Q 6.3: Function

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

is completed to automatically generate panoramas.

The obtained panorama is shown as Fig.11

Figure 11: The panorama image without clipping.

7 Augmented Reality

For this part, the functions are implemented in script `augmentReality.py`.

Q 7.1: Function

```
R,t = compute_extrinsics(K, H)
```

is completed to compute the extrinsic rotation $\mathbf{R}$ and translation $\mathbf{t}$ given the camera intrinsic parameters $\mathbf{K}$ and the estimated homography $\mathbf{H}$.

The returned extrinsic rotation $\mathbf{R}$ and translation $\mathbf{t}$ are:

$$\mathbf{R} = \begin{bmatrix} -0.99938527 & -0.03404437 & 0.00837006 \\ 0.02925233 & -0.67817586 & 0.73431724 \\ -0.019323 & 0.73411068 & 0.67875484 \end{bmatrix} \quad \mathbf{t} = \begin{bmatrix} 10.62869105 \\ 11.83615806 \\ -45.37273426 \end{bmatrix}$$

Q 7.2: Function

```
X=project_extrinsics(K, W, R, t)
```

is implemented to project the 3D points in the `file_sphere.txt`, which would be presented in homogeneous coordinates, onto the 2D image given the intrinsic matrix $\mathbf{K}$ and extrinsic matrix $\mathbf{R}$ and $\mathbf{t}$.

With the obtained extrinsic and intrinsic parameters above, the result of projecting is shown as Fig.12

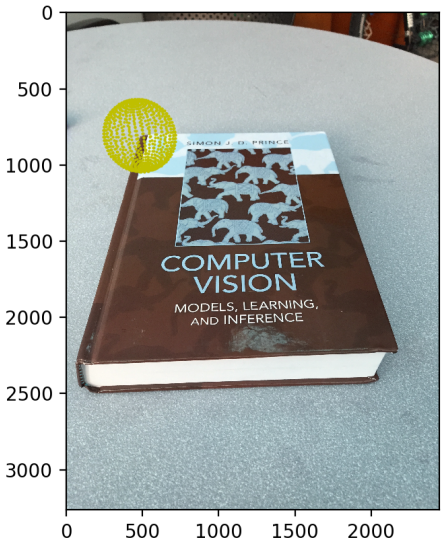

Figure 12: The original projection

Then we try to place the bottom of the ball in the middle of the "o" in "Computer" text of the book title.

In this task, I implemented a function

```
target = get_target_point(im)
```

to get the target point selected by users with mouse click. With the selected target point on the given image `im`, the bottom of the ball in the 3D space would be projected onto that point. With this function, users can select wherever they want to place the sphere onto the image. Function

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

is implemented to project the points in 3D space onto the image in 2D space given the intrinsic matrix **K** and extrinsic matrix **R** and **t**, and also the target point **target** where the user want to place the 3D object to.

By selecting the middle point of the "o" in "Computer" text of the book title as the target point, the projection result is shown as Fig.13

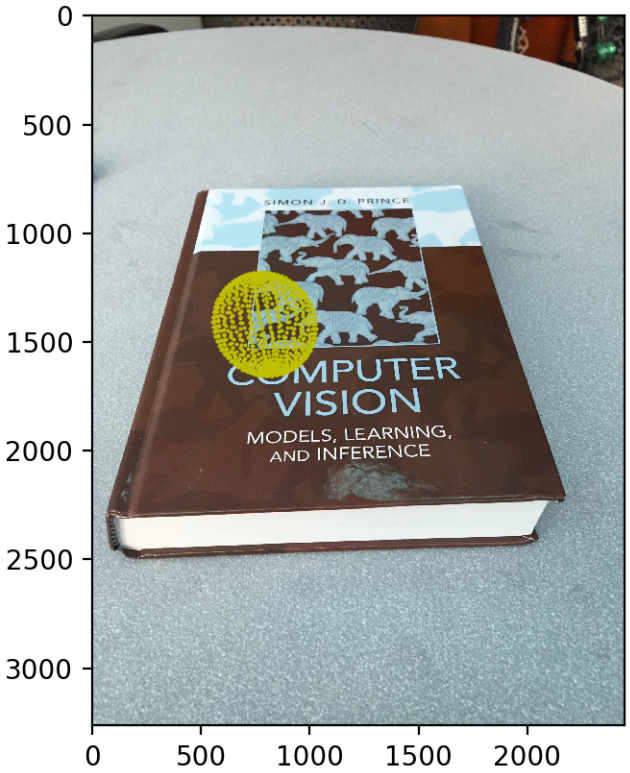

Figure 13: The projection result by placing the bottom of the ball in middle point of the "o" in "Computer" text of the book title

Since I used mouse clicking to select the point where to put the sphere, we can actually choose any point on the image `prince_book.jpg`. Just click on an arbitrary point when selecting the target image, and we'll get the projection result that place the bottom of the ball onto that selected point. Fig.14 is just an example by arbitrarily selecting a target point.

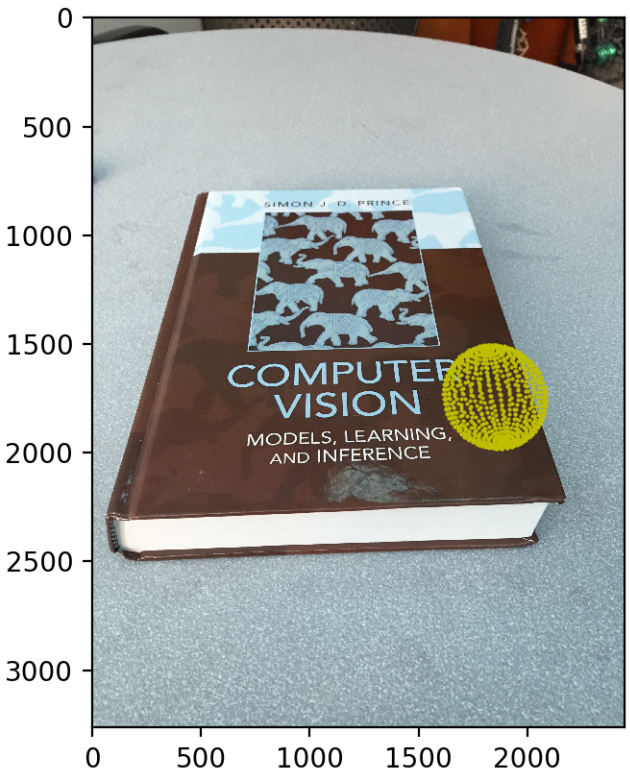

Figure 14: The projection result by placing the bottom of the ball onto the selected target point in the image `prince_book.jpg`.

NOTE: Just run the script `augmentReality.py` and would get the projection results both without the target point (just as Fig.12(b)) and the with the target point (just as Fig.13(b)). When running the script, it would first plot the original projection without the target point. After closing the plotting window, it would show the image `prince_book.jpg` to let you choose the target image. You can select one point on the image with mouse clicking. Then the script would get the selected point and then project the sphere onto that point and plot the projection result.