Lok Yin LEE
20450512
lyleeaf@connect.ust.hk

Homework Assignment 1
Feature Descriptor, Homographies, & RANSAC

COMP5421
2021-03-18

1 Keypoint Detector

In this section, we will be using the Difference of Gaussian(DoG) detector to detect the key point from a grey image. To speed up the calculation of the matrices, no *for loop* will be used.

1.1 Gaussian Pyramid

A function that will create the Gaussian pyramid according to the required levels and Gaussian parameters. The Gaussian pyramid is shown in figure 1.

Figure 1: Gaussian Pyramid.

1.2 DoG Pyramid

The function *createDoGPyramid* will return a *numpy* array that store the DoG pyramid and the DoG levels indicator. The DoG pyramid is shown in figure 2.

Figure 2: DoG Pyramid.

1.3 Edge Suppression

To prevent too many edges are recognized as the key points, using the principal curvature to suppress edges will be sufficient enough for this task. In the function *computePrincipalCurvature*, we will apply 3 by 3 Sobel filter to get those derivatives. This function will then return the curvatures for each levels using the trace and determinant.

1.4 Detecting Extrema

In this part, the DoG pyramid is used to find the key points by only extracting the extrema in the pyramid. The program will separate the calculation to three parts such as the calculation of bottom layer, top layer and middle layers, since they have different number of neighbors. After finding the coordinate of the extremas, the masks of two kind of threshold will be created and applied to the extremas. The function will return those coordinates of key points.

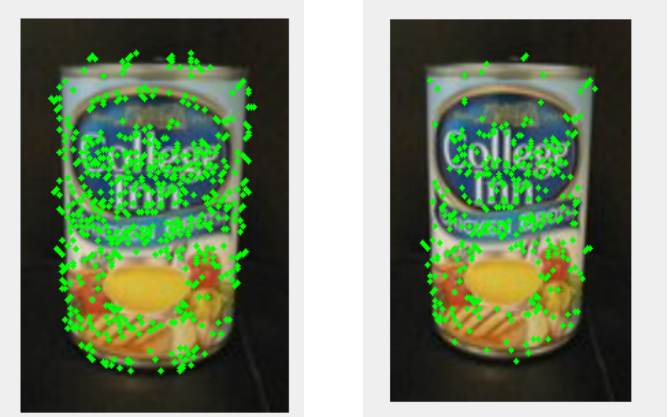

Figure 3: Keypoint Detection without edge suppression and with edge suppression

1.5 Putting it together

To wrap up all the function created, all the previous function will be warped to the *DoGdetector* function. This function will simply get the grey image array and return the key points array. The key point detected is shown in figure 3.

2 BRIEF Descriptor

In this section, we will create the program to apply BRIEF Descriptor and try to match the key points from the same object. Again, to make the program easy to modify and paralleling, no *for loop* will be added.

2.1 Creating a Set of BRIEF Tests

As the mention in the paper, the second approach give a better result in most cases. So, our X will be chosen from the normal distribution with deviation S * S / 25, and Y will be chosen from the normal distribution with deviation S * S / 100. We will generate 256 pairs and encode the 2 dimension coordinate to *compareX* and *compareY* which have the range from 0 to 81. So that the *makeTestPattern* will return matrix of 256 X and Y values.

2.2 Compute the BRIEF Descriptor

This section will compute the identity of the key points from the image and return their value. Except from the points that too close to the boundaries, all other key points will be used to apply the BRIEF pattern and return the identity comparison array.

2.3 Putting it all Together

This *briefLite* function combine all the function before and get the key point identity as return.

2.4 Check Point: Descriptor Matching

This section will be focusing on the testing for multiple images pairs. All the matching result for five chickenbroth, one incline view and five book are shown in figure 4. We can see that most matches from the chickenbroth are correct but the book may perform badly for some case. Basically, all matches with the object is rotated perform worse, which is caused from the how BRIEF identify features. Besides the rotated feature given a wrong matches, too many features from other book may also cause a lot of wrong detection. That why the matching for a standing text book looks perform the best for book matches.

2.5 BRIEF and rotations

In this section, we will use the same image but different rotation angle *modelchickenbroth.jpg* to perform key point matching. To get the ”correct” matches, we first do the padding to the images, and then rotate the second image and perform the matching. To identify which are the ”correct” matches, we will also rotate the

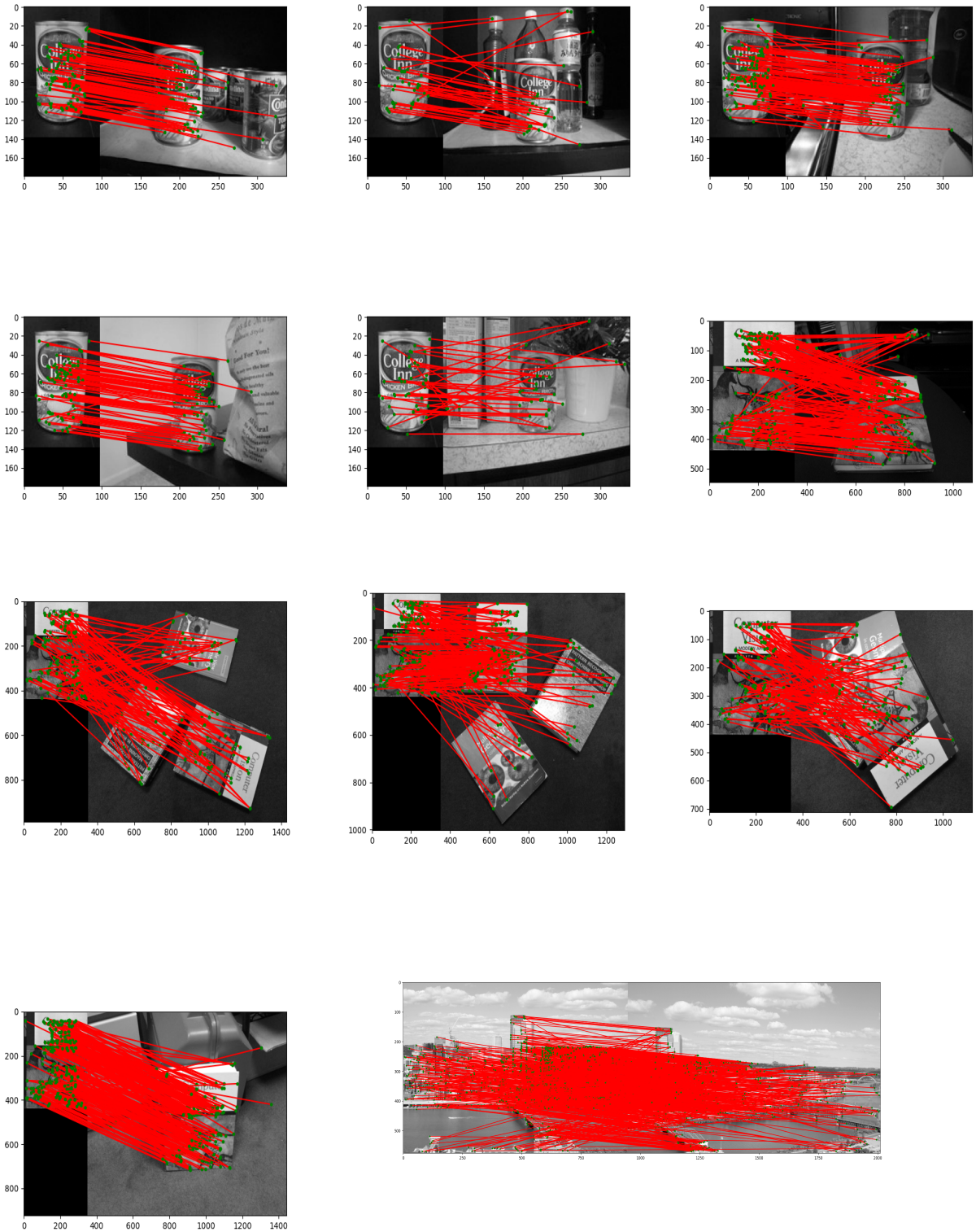

Figure 4: Matches using BRIEF Descriptor

coordinate of the matches in the image 1, this coordinates should be very close to those found from matches in image 2. Also, all the coordinate of the matches is close as the others with a threshold 1.0 will be consider to a ”correct” match. The test result is shown in figure 5 and table 1. Since the BRIEF descriptor encode the features from some fixed points, the sequence of feature points will have a huge difference when the image is rotate, although the total number of each features count may similar. After above 30 degree rotation, most the feature sequence are corrupted and they will not able to match.

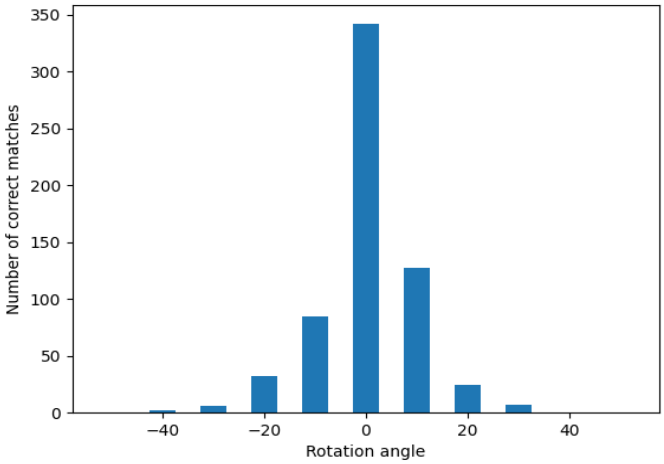

Figure 5: Number of correct for rotated image.

Rotation Angle	-50	-40	-30	-20	-10	0	10	20	30	40	50
Accuracy	0%	10.0%	26.1%	65.3%	78.7%	100%	92.7%	61.5%	26.9%	0%	0%

Table 1: Accuracy of key points for rotated image.

3 Planar Homographies: Theory

a) By using the similar calculation in the lecture, we can find:

$$\mathbf{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tag{1}$$

$$u_i h_{11} + v_i h_{12} + h_{13} - x_i u_i h_{31} - x_i v_i h_{32} - x_i h_{33} = 0 \tag{2}$$

$$-u_i h_{21} - v_i h_{22} - h_{23} + y_i u_i h_{31} + y_i v_i h_{32} + y_i h_{33} = 0 \tag{3}$$

$$\mathbf{Ah} = \begin{bmatrix} u_1 & v_1 & 1 & 0 & 0 & 0 & -x_1u_1 & -x_1v_1 & -x_1 \\ 0 & 0 & 0 & -u_1 & -v_1 & -1 & y_1u_1 & y_1v_1 & y_1 \\ u_2 & v_2 & 1 & 0 & 0 & 0 & -x_2u_2 & -x_2v_2 & -x_2 \\ 0 & 0 & 0 & -u_2 & -v_2 & -2 & y_2u_2 & y_2v_2 & y_2 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ u_n & v_n & 1 & 0 & 0 & 0 & -x_nu_n & -x_nv_n & -x_n \\ 0 & 0 & 0 & -u_n & -v_n & -1 & y_nu_n & y_nv_n & y_n \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} \tag{4}$$

b) There are 9 element in **h**.

c) 4 points is required. Only 8 degrees of freedom are in **H**, since the 9th element of the matrix is just

a scale and it can be normalized to 1.

d) Since

$$\mathbf{A} = \mathbf{USV}^T \tag{5}$$

$$||\mathbf{USV}^T\mathbf{h}||^2 = \mathbf{h}^T\mathbf{VS}^T\mathbf{USV}^T\mathbf{h} = \mathbf{h}^T\mathbf{VS}^T\mathbf{SV}^T\mathbf{h} = ||\mathbf{SV}^T\mathbf{h}||^2 \tag{6}$$

We can instead minimize $||\mathbf{SV}^T\mathbf{h}||$ by setting $\mathbf{V}^T\mathbf{h}$ to $[0, ..., 1]^T$. So that solving the **h** can minimize this homogeneous linear least squares system.

4 Planar Homographies: Implementation

In this section, we will implement the equation that mentioned before. This *computeH* will able to compute size (2 x N) which **N** can be any number of coordinates, and return the size (3 x 3) H matrix by singular value decomposition(SVD). In case the number of points maybe too large, the *A* matrix will be created using *numpy* assignment but not for loop.

5 RANSAC

After we get all the required parameter for RANSAC such as matches and points location, we can do a number of iteration to find out which **H** matrix have more inliers as the best **H**. For each iteration, four points will be selected and their **H** matrix will be calculated, this **H** will be used to examine how suitable it is, the H with the highest number of inliers will be returned.

6 Stitching it together: Panoramas

In this section, we will create two function *imageStitching* and *imageStitching_noClip* which will get two image and the **H** matrix. Then, the original image will be extended and the matrix will be applied to the second image. However, if we want the translation between both image be smooth, we need to apply the weighting for them. So, the function *get_blend_weights* will able to calculate the correct weighting for both image using the distance transform as the pixel of image 2 is warped. Since the part of image 1, image 2 and the overlap will be combined with their weighting mask, a extra function *replace_image* is made for this purpose. After all weighting and image is combined, the function will return the *pano_im*. Depends on differnt image stitching function, the image will or will not clip the top area of image 1. For the last function *generatePanorama*, it will combine the BRIEF descriptor, the RANSAC matching and the image stitching function to return the final image from only image 1 and 2. All the result is shown in figure 6 and 7.

7 Augmented Reality

In this section, we will recreate the ball shape from the points in *sphere.txt* to the image. Before calculating the extrinsics, we will use the *computeH* function to compute the matrix **H** from **W** and **D**. Then, we will compute the rotation matrix **R** and the translation matrix **t**.

$$R = \begin{bmatrix} -0.99938527 & -0.03404437 & 0.00837006 \\ 0.02925233 & -0.67817586 & 0.73431724 \\ -0.019323 & 0.73411068 & 0.67875484 \end{bmatrix} \tag{7}$$

$$t = \begin{bmatrix} 10.62869105 \\ 11.83615806 \\ -45.37273426 \end{bmatrix} \tag{8}$$

For the sphere projection, we set the target as **W** = [[833], [1642], [1]], which is the position of the character ”o”. By collecting all the points of the sphere, we create a matrix to store all the points and apply the multiplication for the intrinsic and extrinsics matrix. During the multiplication, we apply a offset to the z axis as we want the bottom of the sphere stand on the book. After we find the points of **X**, we can simply plot the points using the *matplotlib*. The result is shown in figure 8.

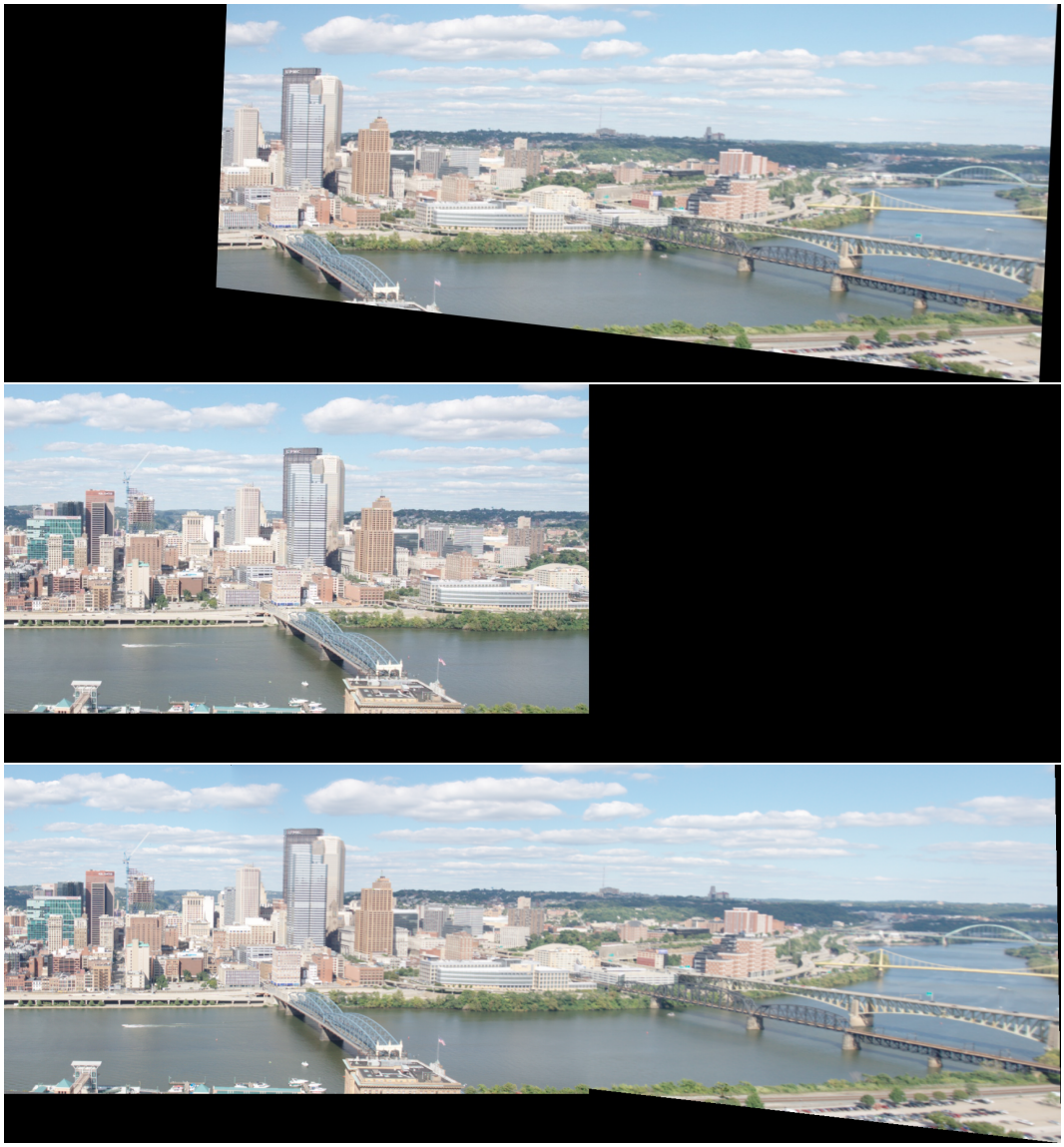

Figure 6: Stitching result with clipped image.

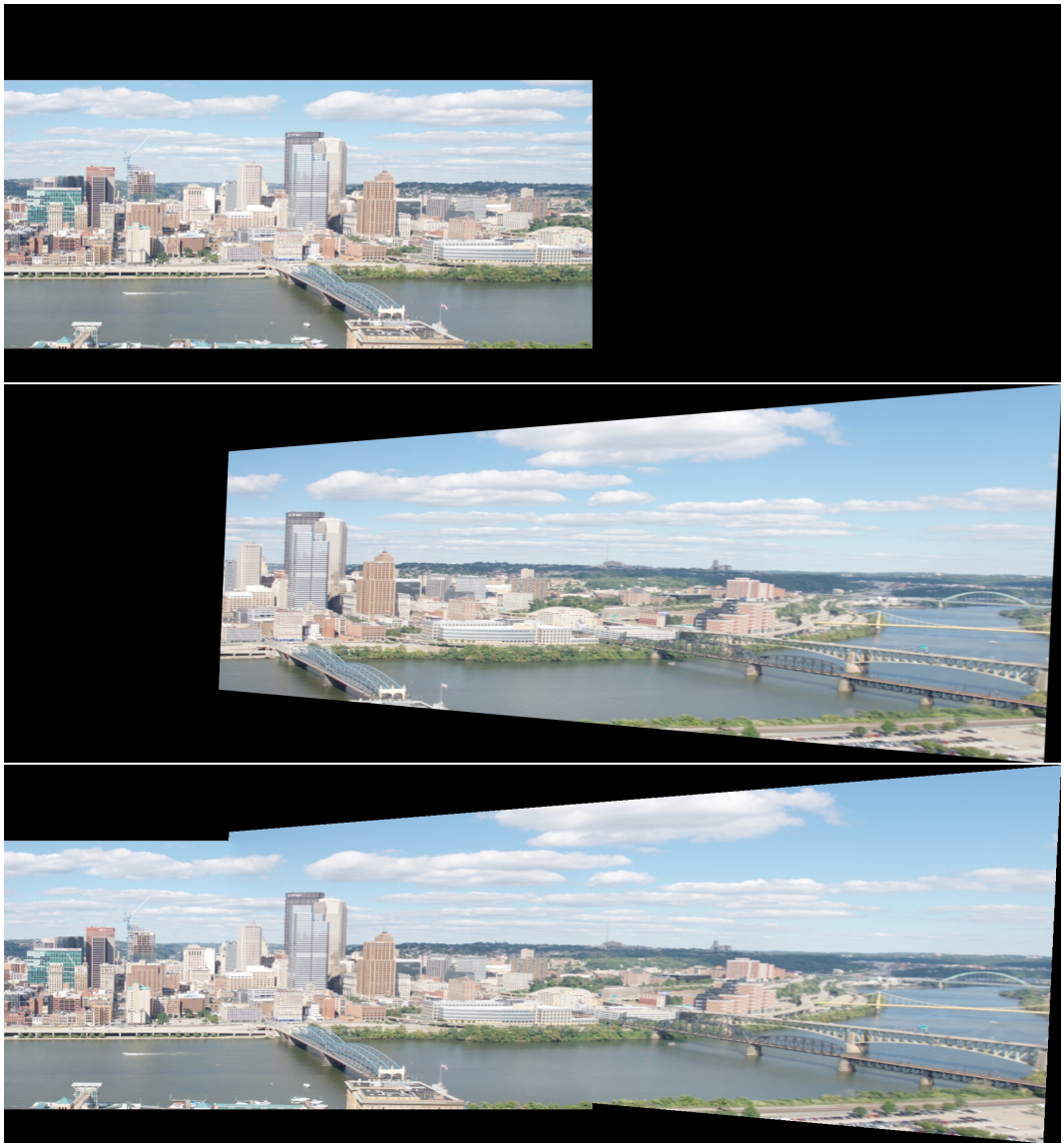

Figure 7: Stitching result without clipped image.

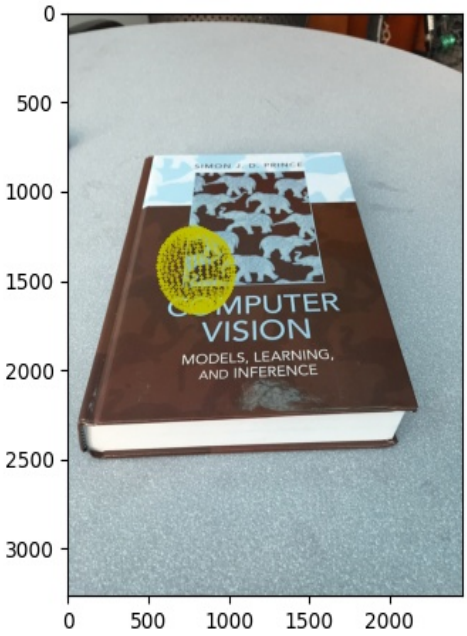

Figure 8: Project a sphere on the book.