

CMU Fall24 16820 Homework 1

Patrick Chen

Collaborators:

September 7, 2024

Q1.1 at page 2

Ans:

Given two camera projection matrices P_1 and P_2 corresponding to two cameras: camera 1 and camera 2, I have the following equations (1) and equations (2), where x_1 represents the projected 2D point, on the image plane of camera 1, of a 3D point, x_π , from a different 3D plane π , and x_2 represents the projected 2D point of the same 3D point, x_π , on the image plane of camera 2.

$$x_1 = P_1 x_\pi \quad (1)$$

$$x_2 = P_2 x_\pi \quad (2)$$

Both of equation (1) and (2) has the same 3D point x_π , it leads to the following equation (3):

$$P_1^{-1} x_1 = P_2^{-1} x_2 \quad (3)$$

Then I move P_1 to the right hand side by both multiply P_1 in equation (1) and (2), then I got equation (4).

$$x_1 = P_1 P_2^{-1} x_2 \quad (4)$$

It is the same form of the given equation (1) at page 2, which is the equation (5) here, where λ is a scale:

$$x_1 = \lambda H x_2 \quad (5)$$

So I get equation (6):

$$H = \lambda P_1 P_2^{-1} \quad (6)$$

Consequently, if I need to prove that H does exist, I need to prove that P_2^{-1} does exist. As x_π lies on a plane, it only has two degrees of freedom, and can be represented as a 2D point. So the projection matrix P_1 and P_2 can be reduced to simplified 3x3 Homography matrix H_1 and H_2 , since the Z dimension in 3D world coordinate of x_π is no longer needed as the projection now is only 2D plane to 2D plane transformation. So it gives the following equation:

$$H = \lambda H_1 H_2^{-1} \quad (7)$$

Under this 2D plane to 2D plan transformation, the Homography between camera plane and 2D plans, π , can be simply reduced to KR , where K is the intrinsic matrix of the camera, and R is the rotation matrix. It is because that the translation is no longer needed as the camera plane can always be rotated to be parallel to the 2D plane, π , and do the further intrinsic mapping. So $H_1 = K_1 R_1$ and $H_2 = K_2 R_2$, $H_2^{-1} = R_2^{-1} K_2^{-1}$. Because K_2 and R_2 are invertible, it makes H_2 is also invertible, so H does exist: $H = \lambda H_1 H_2^{-1}$.

Q1.2 at page 3

Ans:

1. h has 9 variables, but it only has 8 degrees of freedom. Specifically, h is the transformation of two 2D homogeneous coordinates, and if it is multiplied by a scale, λ , then the transformation equation still holds because of homogeneous coordinates. As a result, I have to add a constraint like setting the last element of h to be 1 or setting $\|h_2\| = 1$, and it decreases the degree of freedom by 1. So, h has 8 degrees of freedom.
2. Since h has 8 unknowns, and each point pair can contribute to 2 equations for solving h, I need 4 pairs of points to solve h.
3. Substitute the given equation of $\mathbf{x}_1^i \equiv H\mathbf{x}_2^i$ ($i \in \{1 \dots N\}$) with variables:

$$\begin{bmatrix} x_1^i \\ y_1^i \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_2^i \\ y_2^i \\ 1 \end{bmatrix}$$

Then, I expand the equations to get:

$$\begin{aligned} x_1^i &= h_{11}x_2^i + h_{12}y_2^i + h_{13} \\ y_1^i &= h_{21}x_2^i + h_{22}y_2^i + h_{23} \\ 1 &= h_{31}x_2^i + h_{32}y_2^i + h_{33} \end{aligned}$$

By dividing the last element of homogeneous coordinates to get real plane coordinates, I get the following two equations:

$$\begin{aligned} x_1^i &= \frac{h_{11}x_2^i + h_{12}y_2^i + h_{13}}{h_{31}x_2^i + h_{32}y_2^i + h_{33}} \\ y_1^i &= \frac{h_{21}x_2^i + h_{22}y_2^i + h_{23}}{h_{31}x_2^i + h_{32}y_2^i + h_{33}} \end{aligned}$$

Though multiplying through by denominator and then rearrange:

$$\begin{aligned} h_{11}x_2^i + h_{12}y_2^i + h_{13} - h_{31}x_2^ix_1^i - h_{32}y_2^ix_1^i - h_{33}x_1^i &= 0 \\ h_{21}x_2^i + h_{22}y_2^i + h_{23} - h_{31}x_2^iy_1^i - h_{32}y_2^iy_1^i - h_{33}y_1^i &= 0 \end{aligned}$$

Then I rearrange it to fit into the matrix form $A_i h = 0$

$$\begin{bmatrix} x_2^i & y_2^i & 1 & 0 & 0 & 0 & -x_1^ix_2^i & -x_1^iy_2^i & -x_1^i \\ 0 & 0 & 0 & x_2^i & y_2^i & 1 & -y_1^ix_2^i & -y_1^iy_2^i & -y_1^i \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Then I derive A_i :

$$A_i = \begin{bmatrix} x_2^i & y_2^i & 1 & 0 & 0 & 0 & -x_1^ix_2^i & -x_1^iy_2^i & -x_1^i \\ 0 & 0 & 0 & x_2^i & y_2^i & 1 & -y_1^ix_2^i & -y_1^iy_2^i & -y_1^i \end{bmatrix}$$

4. (a) A trivial solution for h is all 0:

$$h = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

- (b) A is not full rank. In homogeneous system, if h is not trivial, then A must not be invertible, or the only solution of h would be a zero vector (trivial). It means that A should be singular, in other words, A must not be full rank.
- (c) The impact of A being not full rank is that the smallest singular values would be zero or near zero, allowing to find a non-trivial solution h .

Q1.4.1 at page 4

Ans:

Given two cameras separated by a pure rotation, I have the following two equations:

$$x_1 = K_1 IX \quad (8)$$

$$x_2 = K_2 RX \quad (9)$$

Since X is the same data point in 3D space, then I can substitute X in equation (7) with X in equation (8) and result in the following equation:

$$x_1 = K_1 R^{-1} K_2^{-1} x_2 \quad (10)$$

Since $x_1 = \lambda H x_2$, then I can know that:

$$H = \frac{1}{\lambda} K_1 R^{-1} K_2^{-1} \quad (11)$$

Since K_2 and R are invertible, $\det(K_2) \neq 0$ and $\det(R) \neq 0$, H does exist.

Q1.4.2 at page 4

Ans:

Recall from Q1.4.1, $H = K_1 R(\theta)^{-1} K_2^{-1}$, given that θ is the angle of rotation. Since $K_1 = K_2 = K$, I have:

$$H = KR(\theta)^{-1}K^{-1} \quad (12)$$

By multiplying two H together:

$$H^2 = (KR(\theta)^{-1}K^{-1})(KR(\theta)^{-1}K^{-1}) \quad (13)$$

By removing redundant identity matrix:

$$H^2 = KR(\theta)^{-1}R(\theta)^{-1}K^{-1} \quad (14)$$

According to the property of rotation matrix $R(\theta)^{-1} = R(\theta)^T = R(-\theta)$, and $R(-\theta)R(-\theta) = R(-2\theta)$, I have $R(\theta)^{-1}R(\theta)^{-1} = R(2\theta)^{-1}$, so I have:

$$H^2 = KR(2\theta)^{-1}K^{-1} \quad (15)$$

It demonstrates that H^2 corresponds to a rotation of 2θ .

Q1.4.3 at page 4

Ans:

It is because that either one of the following two conditions is needed to be held for the planar homography to exist:

1. The points in the 3D world lie on a 2D plane.
2. Only pure rotation between two views.

As a result, planar homography is not completely sufficient to map any scene image to another viewpoint.

Q1.4.4 at page 5

Ans:

The projection matrix P is a 3×4 matrix that can map a 3D point, X, to a 2D point, x:

$$x = PX \quad (16)$$

A line in 3D space can be represented as the following equation, where t is a parameter:

$$X(t) = X_1 + t(X_2 - X_1) \quad (17)$$

If I multiply the projection matrix, P, on both hand sides of equation (16), then I have:

$$PX(t) = PX_1 + tP(X_2 - X_1) \quad (18)$$

By moving P into parenthesis, then:

$$PX(t) = PX_1 + t(PX_2 - PX_1) \quad (19)$$

Then, I got the following equation:

$$x(t) = x_1 + t(x_2 - x_1) \quad (20)$$

Equation (19) is the equation of line on a 2D plane. As a result, the line can be preserved through perspective projection P.

Q2.1.1 at page 5

Ans:

FAST detector has the following differences when comparing with Harris corner detector:

1. FAST detector is not rotation-invariant, while Harris corner detector can be rotation-invariant with some tricky implementation details.
2. FAST detector only compare intensities with neighbors in a circular range around a candidate point, while Harris corner detector uses much more expansive computations like first order derivative (gradient), summation of a window of gradients, and solving eigenvalue problems, leading to the result that FAST detector is much faster than Harris corner detector.
3. FAST detector is more sensitive to noise and the change of lighting, while Harris corner detector is more robust to noise and lighting change because Harris corner detector gets more information from gradients in multiple directions of a window plus solving eigenvalue problems to get optimized candidates of corners.
4. FAST detector is faster but less accurate in the localizing features, while Harris corner detector is slower but more accurate in localizing feature points.

Q2.1.2 at page 5

Ans:

BRIEF descriptor has the following differences when comparing with filterbanks descriptor:

1. BRIEF descriptor chooses n pairs of points within a predefined window to form a n-bit binary descriptor by setting each bit 1 or 0 according to the comparison of each pair of selected points, while filterbanks convolve the image with a series of predefined filters to form an array of responses in each pixel as the descriptor.
2. BRIEF descriptor is faster than filterbanks descriptor as BRIEF descriptor only use sampling and comparison of intensity, while filterbanks descriptor applies convolution of different filters to produce different response as the descriptor.
3. As for the robustness, filterbanks descriptor is more robust to rotation and scale as it can apply different filters to mitigate the impact from scaling and rotating, while BRIEF descriptor only compares the intensity within a window and it is not rotation-invariant and scale-invariant.

Q2.1.3 at page 5

Ans:

The Hamming distance is to calculate the number of different bits in two binary descriptors. For example, let x_1 be a d-dimensional binary descriptor: $x_1 = (x_1^0, x_1^1, \dots, x_1^{(d-1)})$, and x_2 is another d-dimensional binary descriptor: $x_2 = (x_2^0, x_2^1, \dots, x_2^{(d-1)})$, the Hamming distance is defined as the following:

$$d_H(\mathbf{x}_1, \mathbf{x}_2) = \sum_{i=1}^d \delta(x_1^i, x_2^i), \text{ where } \delta(x_1^i, x_2^i) = \begin{cases} 0 & \text{if } x_1^i = x_2^i, \\ 1 & \text{if } x_1^i \neq x_2^i. \end{cases} \quad (21)$$

Hamming distance has the following benefits over conventional Euclidean distance measure:

1. Hamming distance is faster than Euclidean distance because Hamming distance only needs to compute the number of different bits among two binary vectors, and it can be computed by efficient XOR and counter operations, while Euclidean distance needs to compute several complex operations, like vector subtraction, summation of square and square root operations, making Euclidean distance slow and expensive for computation.
2. In our setting, I use BREIF descriptor on our detected feature points, resulting in vectors in binary form, and it is definitely suitable to use Hamming distance to compare two binary feature descriptor vectors.

Q2.1.4 at page 5

Ans:

The best output is as the following Figure 1, and the corresponding code snippets are as the following Figure 2. The sigma is set to 0.08, and ration is set to 0.61.

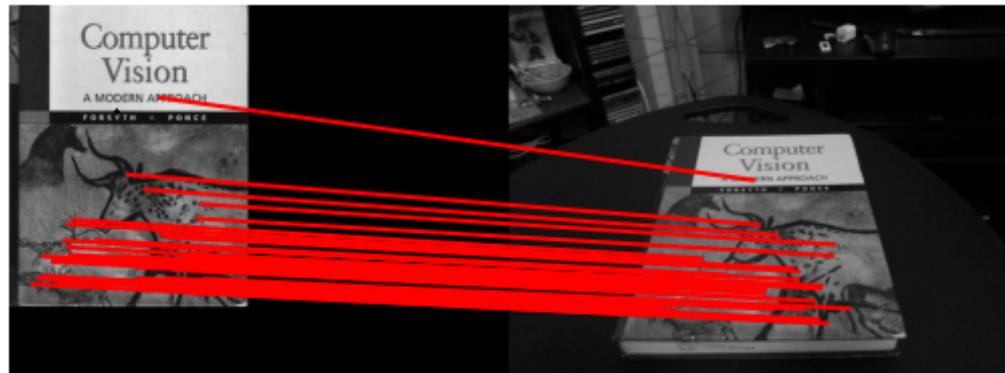


Figure 1: Best Output

```

8 # Q2.1.4
9
10 def matchPics(I1, I2, opts):
11     """
12         Match features across images
13
14         Input
15         -----
16         I1, I2: Source images
17         opts: Command line args
18
19         Returns
20         -----
21         matches: List of indices of matched features across I1, I2 [p x 2]
22         locs1, locs2: Pixel coordinates of matches [N x 2]
23     """
24
25     ratio = opts.ratio #'ratio for BRIEF feature descriptor'
26     sigma = opts.sigma #'threshold for corner detection using FAST feature detector'
27
28
29     # TODO: Convert Images to GrayScale
30     # Convert BGR order (from cv2.imread()) to RGB order
31     image1_rgb = I1[:, :, [2, 1, 0]]
32     image2_rgb = I2[:, :, [2, 1, 0]]
33     gray_I1 = skimage.color.rgb2gray(image1_rgb)
34     gray_I2 = skimage.color.rgb2gray(image2_rgb)
35
36     # TODO: Detect Features in Both Images
37     locs1 = corner_detection(gray_I1, sigma);
38     locs2 = corner_detection(gray_I2, sigma);
39
40
41     # TODO: Obtain descriptors for the computed feature locations
42     desc1, locs1 = computeBrief(gray_I1, locs1)
43     desc2, locs2 = computeBrief(gray_I2, locs2)
44
45
46     # TODO: Match features using the descriptors
47     matches = briefMatch(desc1, desc2, ratio)
48
49     return matches, locs1, locs2
50

```

Figure 2: Q2.1.4 Code Snippet

Q2.1.5 at page 6

Ans:

According to skimage API documentation, the sigma value is used in FAST detector to decide whether the pixels on the circle are brighter, darker, or similar with respect to the test pixel. The lower value of sigma would produce more detected feature points, while the higher value would produce less detected feature points. On the other hand, the ratio value is used in *skimage.feature.match_descriptors()* API. It is the ratio between the distance of the first closest descriptors to the second closest descriptors. Thus, the higher the value would cause more matched correspondence as the matched condition is more easy, while the lower the value would cause less matched correspondence. The following experiments listed in Table 1 is started with the given values of sigma=0.15 and ratio=0.7. As I increase sigma, it results in more false positive matched points, such as the red line near the top of the image in row 1 and row 2 of Table 1. So I decrease sigma to 0.10, and it shows up more unmatched points. Thus, I also decrease ratio to 0.65, and it shows less mismatched correspondence. As I decrease sigma to 0.08, and ratio to 0.61, I think I found the optimized result, as there is no mismatched correspondence. As I keep decreasing sigma to 0.7, the mismatched correspondence near the right border of the book in the right image shows up. As I decrease ratio from 0.61 to 0.55, the mismatched correspondence does not disappear. As the result, I think sigma=0.08 and ratio=0.61 is my best result.

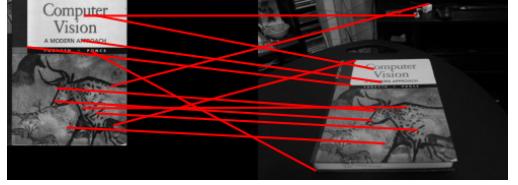
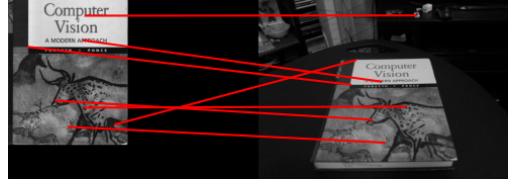
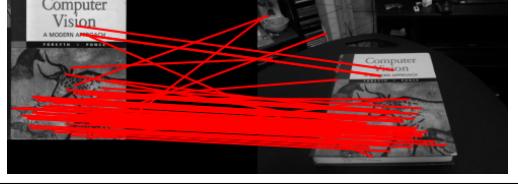
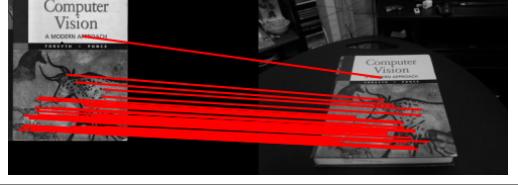
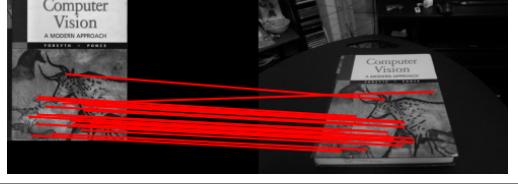
Parameter sigma	Parameter ratio	Generated Image
0.20	0.7	
0.20	0.65	
0.15	0.7	
0.10	0.7	
0.10	0.65	
0.08	0.61	
0.07	0.61	
0.06	0.55	

Table 1: Parameter Changes and Corresponding Generated Images

Q2.1.6 at page 6

Ans:

As shown in the Figure 3, the maximum of matches occurs at rotation = 0 degrees, and the rotation makes the number of matches decrease drastically. I think it is because the BRIEF descriptor and the FAST detector using in the matchPics() function is not rotation-invariant.

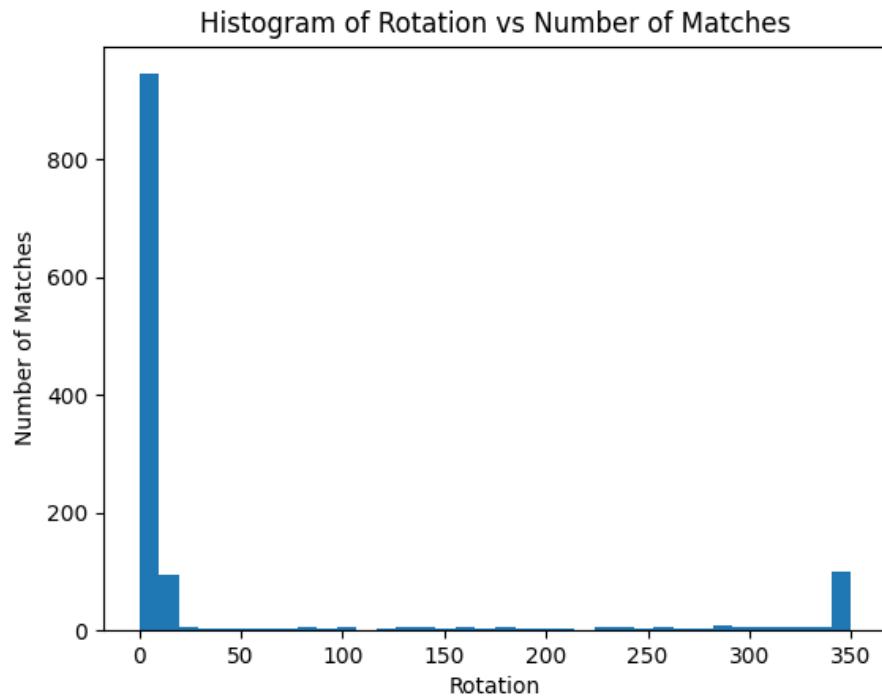


Figure 3: Histogram Result

```

python > briefRotTest.py > ...
1 import numpy as np
2 import cv2
3 import skimage.color
4 from scipy.ndimage import rotate
5 from matchPics import matchPics
6 from opts import get_opts
7
8 #to delete
9 import matplotlib.pyplot as plt
10
11 #Q2.1.6
12
13 def rotTest(opts):
14
15     # TODO: Read the image and convert to grayscale, if necessary
16     image = cv2.imread('../data/cv_cover.jpg')
17     histo = [0]*36
18
19     for i in range(36):
20
21         # TODO: Rotate Image
22         image_rotated = rotate(image, angle=i*10, reshape=True)
23
24         # TODO: Compute features, descriptors and Match features
25         matches, locs1, locs2 = matchPics(image, image_rotated, opts)
26         #print(matches)
27         #print(len(matches) = {len(matches)})
28         #cv2.imshow("image", image)
29         #cv2.imshow("image_rotated", image_rotated)
30         #displayMatched(opts, image, image_rotated)
31         #cv2.waitKey(0)
32
33         # TODO: Update histogram

```

Figure 4: Q2.1.6 Code Snippet 1

```

34     histo[i] = len(matches)
35
36     # TODO: Display histogram
37     plt.hist([i*10 for i in range(36)], bins=36, weights=histo)
38     plt.xlabel('Rotation')
39     plt.ylabel('Number of Matches')
40     plt.title('Histogram of Rotation vs Number of Matches')
41     plt.show()
42
43
44 if __name__ == "__main__":
45
46     opts = get_opts()
47     rotTest(opts)
48

```

Figure 5: Q2.1.6 Code Snippet 2

Q2.1.7 at page 6

Ans:
//to-do

Q2.2.1 at page 7

Ans:

Figure 6 shows the code snippet.

```
9  def computeH(x1, x2):
10     #Q2.2.1
11     # TODO: Compute the homography between two sets of points
12     # Construct A matrix
13     A = np.zeros((2*x1.shape[0], 9))
14     for i in range(x1.shape[0]):
15         xi1 = x1[i][0]
16         yi1 = x1[i][1]
17         xi2 = x2[i][0]
18         yi2 = x2[i][1]
19         A[2*i+0] = [xi2, yi2, 1, 0, 0, 0, -xi1*xi2, -xi1*yi2, -xi1]
20         A[2*i+1] = [0, 0, 0, xi2, yi2, 1, -yi1*xi2, -yi1*yi2, -yi1]
21
22     # Construct ATA
23     ATA = A.T@A
24
25     # Calculate eigenvalues, eigenvectors
26     eigenval, eigenvec = np.linalg.eig(ATA)
27
28     # Choose the eigenvector w/ smallest eigenvalues
29     smallest_eigenvec = eigenvec[:, np.argsort(eigenval)[0]]
30     H2to1 = np.reshape(smallest_eigenvec, (3, 3))
31     H2to1 = np.array(H2to1, dtype=np.float32)
32
33     #test
34     '''
35     x2_homo = np.hstack([x2, np.ones((x2.shape[0], 1))]) #Nx3
36     x1_recover_all = H2to1@x2_homo.T #3xN
37     x1_recover_all = x1_recover_all.T #Nx3
38     x1_col1 = x1_recover_all[:, 0]/x1_recover_all[:, 2]
39     x1_col2 = x1_recover_all[:, 1]/x1_recover_all[:, 2]
40     x1_recover_all = np.column_stack((x1_col1, x1_col2)) #Nx2
41     print(f"original, x1_recover_all = {x1_recover_all}")
42     '''
43
44     return H2to1
```

Figure 6: Q2.2.1 Code Snippet

Q2.2.2 at page 7

Ans:

Figure 7 - Figure 9 show the code snippet.

```

46 def computeDist(x, cx):
47     # x : np.array: (2,)
48     # cx: np.array: (2,)
49     return np.linalg.norm(x-cx)
50
51 def computeLongestDist(x, cx):
52     # x : np.array: (N, 2)
53     # cx: np.array: (2,)
54     ...
55     dist = -1
56     cand = np.array([0, 0])
57     for xi in x:
58         computed_dist = computeDist(xi, cx)
59         if computed_dist > dist:
60             dist = computed_dist
61             cand[0] = xi[0]
62             cand[1] = xi[1]
63     return dist, cand
64
65 return np.sqrt(np.sum((x-cx)**2, axis=1))
66
67 def computeH_norm(x1, x2):#
68     #Q2.2.2
69     # TODO: Compute the centroid of the points
70     cx1, cy1 = np.mean(x1[:, 0]), np.mean(x1[:, 1])
71     cx2, cy2 = np.mean(x2[:, 0]), np.mean(x2[:, 1])
72     c1 = np.array([cx1, cy1])
73     c2 = np.array([cx2, cy2])
74
75     #test
76     ...
77     image = np.zeros((600, 600, 3), dtype = np.uint8)
78     for point in x1:
79         cv2.circle(image, tuple(map(int, point)), radius=5, color=(0, 255, 0), thickness=-1)
80     for point in x2:
81         cv2.circle(image, tuple(map(int, point)), radius=5, color=(255, 0, 0), thickness=-1)
82     cv2.circle(image, tuple(map(int, c1)), radius=5, color=(0, 0, 255), thickness=1)
83     cv2.circle(image, tuple(map(int, c2)), radius=5, color=(0, 0, 255), thickness=1)
84     ...
85
86     # TODO: Shift the origin of the points to the centroid
87     # TODO: Normalize the points so that the largest distance from the origin is equal to sqrt(2)
88     # Merge these two TODO to form T1 and T2
89     # Calculate the largest distance from the center (cx1, cy1), (cx2, cy2)
90     longest_dist1 = computeLongestDist(x1, c1)
91     longest_dist2 = computeLongestDist(x2, c2)
92     norm_factor1 = np.sqrt(2)/longest_dist1
93     norm_factor2 = np.sqrt(2)/longest_dist2
94
95     #test
96     ...
97     cv2.circle(image, tuple(map(int, c1)), radius=5, color=(20, 140, 255), thickness=-1)
98     cv2.circle(image, tuple(map(int, c2)), radius=5, color=(20, 140, 255), thickness=-1)
99     cv2.line(image, tuple(map(int, c1)), c1, color=(255, 255, 255), thickness=2)
100    cv2.line(image, tuple(map(int, c2)), c2, color=(255, 255, 255), thickness=2)
101
102    cv2.imshow("Points on Blank Image", image)

```

Figure 7: Q2.2.2 Code Snippet 1

```

102     ...
103     ...
104
105     # TODO: Similarity transform 1
106     T1 = np.zeros((3, 3))
107     T1[0][0] = norm_factor1
108     T1[0][2] = norm_factor1*c1[0]
109     T1[1][1] = norm_factor1
110     T1[1][2] = norm_factor1*c1[1]
111     T1[2][2] = 1
112
113     x1_norm = np.hstack([x1, np.ones((x1.shape[0], 1))]) #Nx3
114     x1_norm = T1@x1_norm.T #Nx3
115     x1_col1 = x1_norm[:, 0]/x1_norm[:, 2]
116     x1_col2 = x1_norm[:, 1]/x1_norm[:, 2]
117     x1_norm = np.column_stack([x1_col1, x1_col2]) #Nx2
118
119     #test
120     ...
121
122     x2_norm_homo = np.hstack([x2, np.ones((x2.shape[0], 1))]) #Nx3
123     x2_norm_homo = np.linalg.inv(T2)@x2_norm_homo.T
124     x2_recover = x2_norm_homo.T
125     x1_recover = np.column_stack([x1_recover[:, 0]/x1_recover[:, 2], x1_recover[:, 1]/x1_recover[:, 2]])
126     print("x1_recover = ", x1_recover)
127
128
129     # TODO: Similarity transform 2
130     T2 = np.zeros((3, 3))
131     T2[0][0] = norm_factor2
132     T2[0][2] = norm_factor2*c2[0]
133     T2[1][1] = norm_factor2
134     T2[1][2] = norm_factor2*c2[1]
135     T2[2][2] = 1
136
137     x2_norm = np.hstack([x2, np.ones((x2.shape[0], 1))]) #Nx3
138     x2_norm = T2@x2_norm.T #Nx3
139     x2_col1 = x2_norm[:, 0]/x2_norm[:, 2]
140     x2_col2 = x2_norm[:, 1]/x2_norm[:, 2]
141     x2_norm = np.column_stack([x2_col1, x2_col2]) #Nx2
142
143     #test
144     ...
145
146     x2_norm_homo = np.hstack([x2_norm, np.ones((x2_norm.shape[0], 1))])
147     x2_recover = np.linalg.inv(T2)@x2_norm_homo.T
148     x2_recover = x2_recover.T
149     x2_recover = np.column_stack([x2_recover[:, 0]/x2_recover[:, 2], x2_recover[:, 1]/x2_recover[:, 2]])
150     print("x2_recover = ", x2_recover)
151
152
153     # TODO: Compute homography
154     H2to1 = computeH_norm(x1_norm, x2_norm)
155
156     # TODO: Denormalization
157     H2to1 = np.linalg.inv(T1)@H2to1@norm#T2
158
159
160
161
162
163
164
165
166
167
168
169

```

Figure 8: Q2.2.2. Code Snippet 2

```

158     ...
159     x2_homo = np.hstack([x2, np.ones((x2.shape[0], 1))]) #Nx3
160     x1_recover_all = H2to1@x2_homo.T #3xN
161     x1_recover_all = x1_recover_all.T #Nx3
162     x1_col1 = x1_recover_all[:, 0]/x1_recover_all[:, 2]
163     x1_col2 = x1_recover_all[:, 1]/x1_recover_all[:, 2]
164     x1_recover_all = np.column_stack([x1_col1, x1_col2]) #Nx2
165     print("norm, x1_recover_all = {x1_recover_all}")
166
167
168     return H2to1
169

```

Figure 9: Q2.2.2 Code Snippet3

Q2.2.3 at page 7

Ans:

Figure 10 - Figure 11 show the code snippet.

```

170 def compute_rmse(loc1, loc2, opts, fit_inlier_list, tol = False, fit_inlier_list_num: int = 4):
171     #Q2.3.3
172     """
173         Compute the best fitting homography given a list of matching points
174         max_iters = opts.max_iters # the number of iterations to run RANSAC for
175         inlier_tol = opts.inlier_tol # the tolerance value for considering a point to be an inlier
176
177     if loc1.shape[0] < 4:
178         raise ValueError("There are less than 4 matching points from loc1, loc2")
179     max_inlier_num = -1
180     max_inliers = np.zeros(loc1.shape[0], dtype = int)
181     for i in range(max_iters):
182         # sample 4 points from loc1, loc2
183         random_index = random.sample(range(0, loc1.shape[0]), 4)
184         loc1_sample1 = loc1[random_index, :]
185         loc1_sample2 = loc2[random_index, :]
186
187         # calculate the Homography
188         H2col1 = compute_norm(loc1_sample1, loc2_sample2)
189
190         # calculate the inlier number by recovering x1 points
191         loc1_recover_all = np.column_stack([loc1, np.ones((loc1.shape[0], 1))]) #Nx3
192         loc2_recover_all = np.column_stack([loc2, np.ones((loc2.shape[0], 1))]) #Nx3
193         local_recover_all1 = loc1_recover_all[:, 1] #Nx3
194         local_recover_all2 = loc1_recover_all[:, 0] #Nx3
195         local_col1 = np.column_stack([local_recover_all1[:, 1], local_recover_all1[:, 2]])
196         local_col2 = np.column_stack([local_recover_all2[:, 1], local_recover_all2[:, 2]])
197         local_recover_all1 = np.column_stack([local_col1, local_col2]) #Nx2
198
199         distance_sq = np.sum((loc1_recover_all[:, 1]-x1)**2, axis=1)
200         inliers = distance_sq < inlier_tol**2
201         ...
202         for i in range(loc1_recover_all.shape[0]):
203             recovered_pt = loc1_recover_all[i]
204             original_pt = loc1[i]
205             dist = np.sqrt(np.sum((recovered_pt - original_pt)**2))
206             if dist < inlier_tol:
207                 inliers[i] = 1
208             ...
209
210         inlier_num = np.sum(inliers)
211         if inlier_num > max_inlier_num:
212             max_inlier_num = inlier_num
213             max_inliers = inliers
214             bestH2tol = tol
215             max_inliers
216
217     #fit the inlier again
218     if fit_inlier_list:
219         indices = [i for i, x in enumerate(inliers) if x == 1]
220         if len(indices) < fit_inlier_list_num:
221             loc1_opt = loc1[indices, :]
222             loc2_opt = loc2[indices, :]
223             bestH2tol = compute_norm(loc1_opt, loc2_opt)
224
225     #test

```

Figure 10: Q2.2.3 Code Snippet 1

```

226     """
227     x2 = locs2
228     x2_homo = np.hstack([x2, np.ones((x2.shape[0], 1))]) #Nx3
229     x1_recover_all = bestH2tol@x2_homo.T #Nx3
230     x1_recover_all = x1_recover_all[:, 1] #Nx3
231     x1_col1 = x1_recover_all[:, 0]/x1_recover_all[:, 2]
232     x1_col2 = x1_recover_all[:, 1]/x1_recover_all[:, 2]
233     x1_recover_all = np.column_stack([x1_col1, x1_col2]) #Nx2
234     print(F"ransac, x1_recover_all = {x1_recover_all}")
235     ...
236
237     return bestH2tol, inliers

```

Figure 11: Q2.2.3. Code Snippet 2

Q2.2.4 at page 8

Ans:

Figure 12 shows the result image, and Figure 13 - Figure 14 show the code snippet. Besides, in the step4, the reason that the book space in cv_desk.png is not filled up is that the size of hp_cover.jpg is not the same as cv_cover.jpg, and the homography warping cause some blank pixels without any mapping intensity values. The solution is to resize the hp_cover.jpg to the size of cv_cover.jpg as shown in Figure 13 line 32.

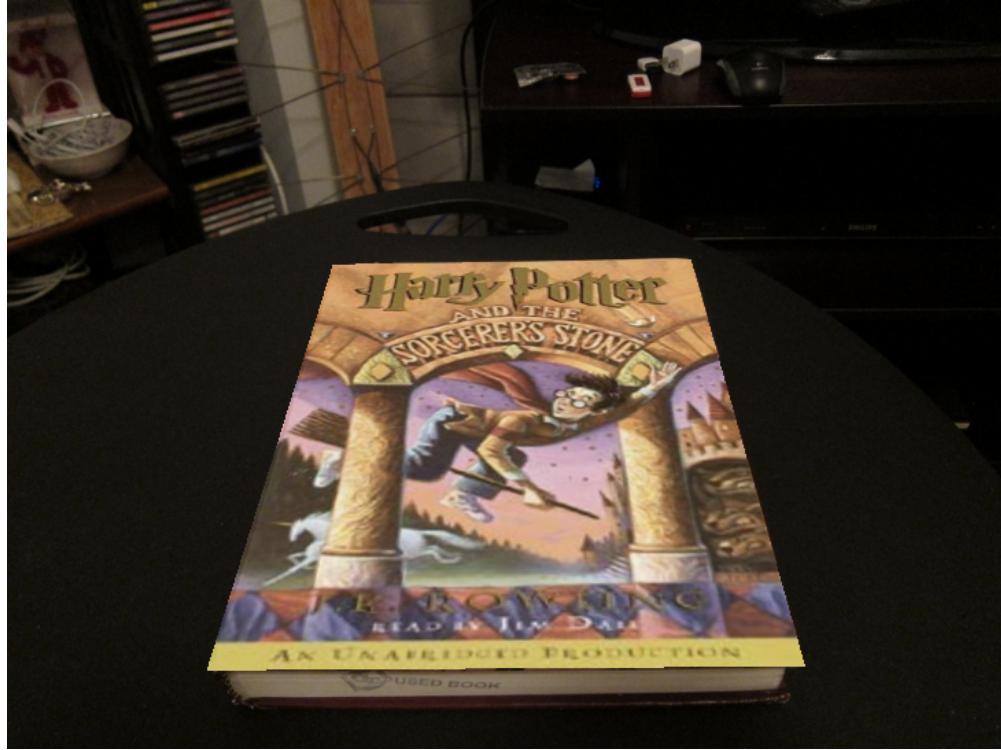


Figure 12: Q2.2.4 Result Image

```

15 # Q2.2.4
16 def warpImage(opts):
17     # Read Input Image
18     image_cv_desk = cv2.imread('../data/cv_desk.jpg')
19     image_cv_cover = cv2.imread('../data/cv_cover.jpg')
20     image_hp_cover = cv2.imread('../data/hp_cover.jpg')
21
22     # Compute Homography
23     matches, locs1, locs2 = matchPics(image_cv_desk, image_cv_cover, opts)
24     x1 = np.float32(locs1[:, 1])
25     x2 = np.float32(locs2[:, 1])
26     x1_correct_pt = np.flip(x1, axis=1)
27     x2_correct_pt = np.flip(x2, axis=1)
28     H2to1_ransac, inliers = computeRansac(x1_correct_pt, x2_correct_pt, opts, fit_inlier_last=True, fit_inlier_last_n=4)
29
30     # Resize the image_hp_cover to the size of image_cv_cover
31     resizeImage_hp_cover = cv2.resize(image_hp_cover, (image_cv_cover.shape[1], image_cv_cover.shape[0]))
32
33     # Warping hp_cover.jpg to cv_desk.png
34     warped_hp_image = cv2.warpPerspective(resizeImage_hp_cover, H2to1_ransac, (image_cv_desk.shape[1], image_cv_desk.shape[0]))
35
36     # Composite the warped hp_cover.jpg with cv_desk.png
37     composite_img = compositeH(H2to1_ransac, resizeImage_hp_cover, image_cv_desk)
38
39     cv2.imshow('composite_img', composite_img)
40     cv2.waitKey(0)
41
42

```

Figure 13: Q2.2.4 warpImage() Code Snippet

```

239 def compositeH(H2to1, template, img):
240
241     #Create a composite image after warping the template image on top
242     #of the image using the homography
243
244     #Note that the homography we compute is from the image to the template;
245     #x_template = H2to1*x_photo
246     #For warping the template to the image, we need to invert it.
247
248
249     # TODO: Create mask of same size as template
250     mask = np.full((template.shape[0], template.shape[1], 1), 255, dtype=np.uint8)
251     composite_img = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)
252
253
254     # TODO: Warp mask by appropriate homography
255     warped_mask = cv2.warpPerspective(mask, H2to1, (img.shape[1], img.shape[0]))
256
257     # TODO: Warp template by appropriate homography
258     warped_template = cv2.warpPerspective(template, H2to1, (img.shape[1], img.shape[0]))
259
260     # TODO: Use mask to combine the warped template and the image
261     inverse_mask = cv2.bitwise_not(warped_mask)
262     masked_template = cv2.bitwise_and(warped_template, warped_template, mask=warped_mask)
263     masked_img = cv2.bitwise_and(img, img, mask=inverse_mask)
264     composite_img = cv2.add(masked_template, masked_img)
265
266     ...
267     for i in range(img.shape[0]):
268         for j in range(img.shape[1]):
269             if warped_mask[i, j] == 255:
270                 composite_img[i, j] = warped_template[i, j]
271             else:
272                 composite_img[i, j] = img[i, j]
273
274
275     return composite_img
276

```

Figure 14: Q2.2.4. CompositeH() Code Snippet

Q2.2.5 at page 8

Ans:

Table 2 shows the result of different setting among parameters 'max_iters' and 'inlier_tol'. According to equation (22) specified in the slides of lecture, as the inlier_tol decreases, it makes the feature points harder to be considered as an inlier, so the inlier rate w would decrease, and it makes the k (max_iters), the number of trials that can gaurantee a return of a good model, increased too. On the opposite, if inlier_tol increases, w would increase, and thus k (max_iters) would decrease. However, in Table 2, 2nd row increase the inlier_tol from 2.0 to 10.0, and the result seems not quite good. The reason is that at the end of computeH_ransac() function, I would use all inliers again to fit an optimized homography. If the inlier_tol is too large, then some outlier match points may be deemed as inlier, and they would be included into the last step calculating of optimized homography, making the result worse. On the other hand, from 4th row to 5th row, the inlier_tol is decreased from 1 to 0.001, and the result seems worse in a great amount. The reason is that when iter_tol is too low, then most of the time the homography is calculated using outliers. Sometimes the homography is truly computed by true inliers, but the number of inlier just equals to those fitting using outliers because of the very low value of iter_tol. Accordingly, the RANSAC model selection process through inliers would become ineffective. In this problem, according to my experiments, the appropriate range for inlier_tol should be [0.5, 8], and the max_iters set to 500 is enough.

$$k = \frac{\log(1 - p)}{\log(1 - w^n)} \quad (22)$$

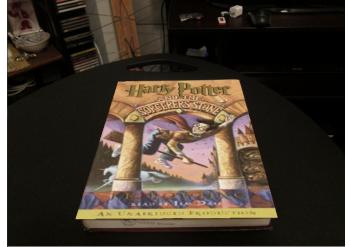
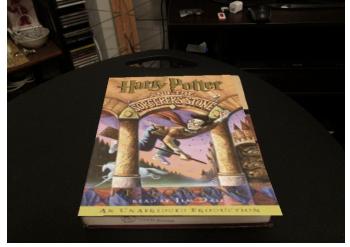
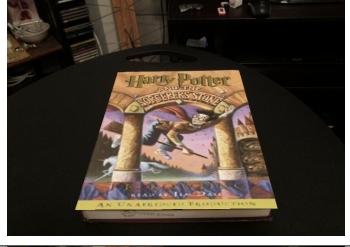
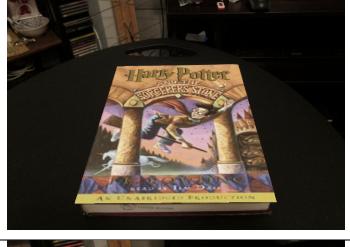
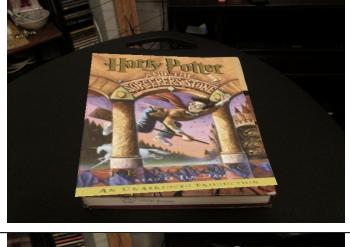
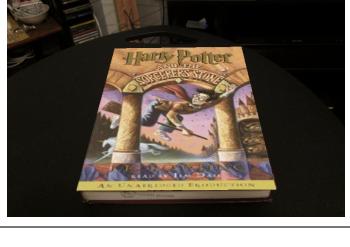
Parameter max_iters	Parameter inlier_tol	Composite Image
500	2.0	
500	10.0	
5000	2.0	
500	1.0	
500	0.001	
5000	0.001	

Table 2: Parameter Changes and Corresponding Composite Images

Q3.1 at page 9

Ans:

Figure 15 - Figure 17 shows the results of composite video where the timestamps displaying overlays at left, right, and center of the video frame. The resulted video is here: ar.avi, and the code snippets are as the following Figure 18 - Figure 20.

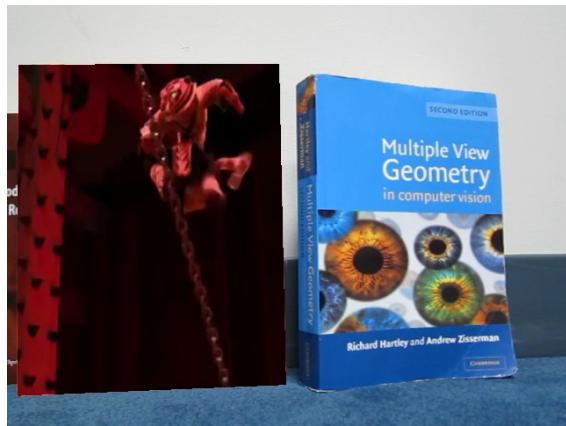


Figure 15: Q3.1 Screenshot of Result at Left Timestamp



Figure 16: Q3.1 Screenshot of Result at Right Timestamp



Figure 17: Q3.1 Screenshot of Result at Center Timestamp

```

16 import cv2
17 import numpy as np
18
19 # Import necessary functions
20 from helper import loadvid
21 from opts import get_opts
22 from helpers import listdir
23 from displayMatch import displayMatched
24 from planarH import computeH
25 from planarL import computeL_ransac
26 import os
27
28 # Write script for Q3.1
29 #####
30 # Main Routine #
31 #####
32 def main():
33     # Input arguments: load input videos
34     opts = get_opts()
35     if not os.path.exists(opts.out_folder):
36         os.makedirs(opts.out_folder)
37     frames_ar_source = loadvid(opts.input_src_file)
38     frames_book = loadvid(opts.input_dst_file)
39
40     # Process each frame
41     frame_result = generalHierarchyPoterize(frames_ar_source, frames_book, opts)
42
43     # Write out video
44     writeOutVideo(frame_result, opts)
45
46     # Select out three distinct timestamps
47     frameidx_book_left = 123
48     frameidx_book_center = 333
49     frameidx_book_right = 333
50
51     cv2.imwrite(f'{opts.out_folder}/Q3_1_ar_result_overlay_left.jpg', frame_result[frameidx_book_left])
52     cv2.imwrite(f'{opts.out_folder}/Q3_1_ar_result_overlay_right.jpg', frame_result[frameidx_book_right])
53     cv2.imwrite(f'{opts.out_folder}/Q3_1_ar_result_overlay_center.jpg', frame_result[frameidx_book_center])
54
55     print("Done")
56
57 #####
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116

```

Figure 18: Q3.1 Code Snippet 1

```

1 # Load image
2 ar_cv_cover = cv2.imread('Q3_1_ar_cv_cover.jpg')
3 book_cv_cover = cv2.imread('Q3_1_book_cv_cover.jpg')
4
5 # Create a copy of the source image
6 ar_cv_copy = ar_cv_cover.copy()
7 book_cv_copy = book_cv_cover.copy()
8
9 # Create a copy of the destination image
10 ar_cv_copy2 = ar_cv_copy.copy()
11 book_cv_copy2 = book_cv_copy.copy()
12
13 # Compute the homography matrix
14 H_ar2book = cv2.findHomography(ar_cv_copy, book_cv_copy, cv2.RANSAC)[0]
15
16 # Compute the homography matrix
17 H_book2ar = cv2.findHomography(book_cv_copy, ar_cv_copy, cv2.RANSAC)[0]
18
19 # Render the frame_ar_cv_copy to the size of frame_book_cv_copy
20 ar_cv_copy2 = cv2.resize(ar_cv_copy2, (book_cv_copy.shape[1], book_cv_copy.shape[0]))
21
22 # Render the frame_book_cv_copy to the size of frame_ar_cv_copy
23 book_cv_copy2 = cv2.resize(book_cv_copy2, (ar_cv_copy.shape[1], ar_cv_copy.shape[0]))
24
25 # Overlay the frame_ar_cv_copy onto the frame_book_cv_copy
26 book_cv_copy2 = cv2.warpPerspective(ar_cv_copy2, H_ar2book, (book_cv_copy.shape[1], book_cv_copy.shape[0]), borderMode=cv2.BORDER_CONSTANT, borderValue=(0, 0, 0))
27
28 # Overlay the frame_book_cv_copy onto the frame_ar_cv_copy
29 ar_cv_copy2 = cv2.warpPerspective(book_cv_copy2, H_book2ar, (ar_cv_copy.shape[1], ar_cv_copy.shape[0]), borderMode=cv2.BORDER_CONSTANT, borderValue=(0, 0, 0))
30
31 # Add two masks
32 mask_ar = np.zeros(ar_cv_copy2.shape, np.uint8)
33 mask_book = np.zeros(book_cv_copy2.shape, np.uint8)
34
35 # Create binary mask for the book
36 mask_ar[book_cv_copy2 > 0] = 255
37 mask_book[ar_cv_copy2 > 0] = 255
38
39 # Create binary mask for the book
40 mask_ar2 = np.zeros(ar_cv_copy2.shape, np.uint8)
41 mask_book2 = np.zeros(book_cv_copy2.shape, np.uint8)
42
43 # Create binary mask for the book
44 mask_ar3 = np.zeros(ar_cv_copy2.shape, np.uint8)
45 mask_book3 = np.zeros(book_cv_copy2.shape, np.uint8)
46
47 # Create binary mask for the book
48 mask_ar4 = np.zeros(ar_cv_copy2.shape, np.uint8)
49 mask_book4 = np.zeros(book_cv_copy2.shape, np.uint8)
50
51 # Create binary mask for the book
52 mask_ar5 = np.zeros(ar_cv_copy2.shape, np.uint8)
53 mask_book5 = np.zeros(book_cv_copy2.shape, np.uint8)
54
55 # Create binary mask for the book
56 mask_ar6 = np.zeros(ar_cv_copy2.shape, np.uint8)
57 mask_book6 = np.zeros(book_cv_copy2.shape, np.uint8)
58
59 # Create binary mask for the book
60 mask_ar7 = np.zeros(ar_cv_copy2.shape, np.uint8)
61 mask_book7 = np.zeros(book_cv_copy2.shape, np.uint8)
62
63 # Create binary mask for the book
64 mask_ar8 = np.zeros(ar_cv_copy2.shape, np.uint8)
65 mask_book8 = np.zeros(book_cv_copy2.shape, np.uint8)
66
67 # Create binary mask for the book
68 mask_ar9 = np.zeros(ar_cv_copy2.shape, np.uint8)
69 mask_book9 = np.zeros(book_cv_copy2.shape, np.uint8)
70
71 # Create binary mask for the book
72 mask_ar10 = np.zeros(ar_cv_copy2.shape, np.uint8)
73 mask_book10 = np.zeros(book_cv_copy2.shape, np.uint8)
74
75 # Create binary mask for the book
76 mask_ar11 = np.zeros(ar_cv_copy2.shape, np.uint8)
77 mask_book11 = np.zeros(book_cv_copy2.shape, np.uint8)
78
79 # Create binary mask for the book
80 mask_ar12 = np.zeros(ar_cv_copy2.shape, np.uint8)
81 mask_book12 = np.zeros(book_cv_copy2.shape, np.uint8)
82
83 # Create binary mask for the book
84 mask_ar13 = np.zeros(ar_cv_copy2.shape, np.uint8)
85 mask_book13 = np.zeros(book_cv_copy2.shape, np.uint8)
86
87 # Create binary mask for the book
88 mask_ar14 = np.zeros(ar_cv_copy2.shape, np.uint8)
89 mask_book14 = np.zeros(book_cv_copy2.shape, np.uint8)
90
91 # Create binary mask for the book
92 mask_ar15 = np.zeros(ar_cv_copy2.shape, np.uint8)
93 mask_book15 = np.zeros(book_cv_copy2.shape, np.uint8)
94
95 # Create binary mask for the book
96 mask_ar16 = np.zeros(ar_cv_copy2.shape, np.uint8)
97 mask_book16 = np.zeros(book_cv_copy2.shape, np.uint8)
98
99 # Create binary mask for the book
100 mask_ar17 = np.zeros(ar_cv_copy2.shape, np.uint8)
101 mask_book17 = np.zeros(book_cv_copy2.shape, np.uint8)
102
103 # Create binary mask for the book
104 mask_ar18 = np.zeros(ar_cv_copy2.shape, np.uint8)
105 mask_book18 = np.zeros(book_cv_copy2.shape, np.uint8)
106
107 # Create binary mask for the book
108 mask_ar19 = np.zeros(ar_cv_copy2.shape, np.uint8)
109 mask_book19 = np.zeros(book_cv_copy2.shape, np.uint8)
110
111 # Create binary mask for the book
112 mask_ar20 = np.zeros(ar_cv_copy2.shape, np.uint8)
113 mask_book20 = np.zeros(book_cv_copy2.shape, np.uint8)
114
115 #-----Execution-----#
116 | if __name__ == '__main__':
117 |     main()

```

Figure 19: Q3.1 Code Snippet 2

```

113
114  #-----Execution-----#
115  if __name__ == '__main__':
116      main()

```

Figure 20: Q3.1 Code Snippet 3

Q3.2 at page 10

Ans:

In order to accelerate the whole program to approach the real-time performance, I firstly need to know which part of the program takes long time to process. To get the 30 FPS result, each for loop to process a frame should use less than 1/30 seconds, nearly 0.03 seconds. Figure 21 shows the code I added to do the profiling, and Figure 22 shows the result. As we can see from the result, both the time used to compute BRIEF descriptor and the match of the BRIEF descriptors take longer than 0.03 seconds, that is, original BRIEF descriptor takes 7.23 seconds to complete, and BRIEF matching takes 0.341 seconds to complete. One can reproduce the profiling result by referring to the command specified in python/README.md. When it comes to improving the whole process, my first thought is to down-sample the input frame, and use the smaller images (frames) to calculate feature detection, description, and matching, and then calculate the corresponding homography and rescale it to the original scale. The benefit is obvious: calculation on lower resolution reduces the processing time greatly. Hence, I did the following three changes:

1. Include a down_sample_factor input argument to specify the factor of down-sampling. Down-sample each of the frames before calculating feature detection, descriptor, matching, and homography. After calculating the homography, rescale it back to the original scale.
2. Since I have the change of scaling, and the original implementation of FAST detector and BRIEF descriptor are not scale-invariant, I choose to replace them with OpenCV built-in ORB detector and descriptor, which has both scale-invariance and rotation-invariance.
3. As I have changed to the ORB detector and descriptor, the matching method should also be changed. According to this discussion and OpenCV website, I would choose FLANN based Matcher over BF Matcher.

```

1)  for i in range(0,100):
2)      # Load frame
3)      frame = cv2.imread('frame%d.jpg' % i)
4)      # Compute homography
5)      start_time = time.time()
6)      frame_ar, hom, local, loc_ar, time_cnn_dt, time_brief, time_match = matchbf(frames_book[i], image_cv_cover, optb, time_cnn_dt, time_brief, time_match, i, report_str)
7)      x1 = local[match[:, 0, :]]
8)      y1 = local[match[:, 1, :]]
9)      x2_correct_pt = np.float32(x1)
10)     y2_correct_pt = np.float32(y1)
11)     x2_correct_pt = np.flip(x2_correct_pt, 1)
12)     hom = np.linalg.inv(hom)
13)     hom = np.array(hom).reshape((1, 9))
14)     time_match = end_match - start_match
15)
16)     try:
17)         start_time = time.time()
18)         # Create warped image
19)         image_cv_ar = cv2.warpPerspective(image_cv_cover, hom, (image_cv_cover.shape[1], image_cv_cover.shape[0]))
20)         # Reshape the frame of ar_source to the size of frame of book
21)         start_time = time.time()
22)         image_cv_ar, image_cv_ar.shape[0], image_cv_ar.shape[1] = cv2.resize(image_cv_ar, (image_cv_ar.shape[0], image_cv_ar.shape[1]), int(frame_ar_source[1]/2), int(frame_ar_source[1]/2)) #(x,y)
23)         ar_center = (int(frame_ar_source[1]/2), int(frame_ar_source[1]/2))
24)         min_ar_center = (int(ar_center[0]-roi_range[1]/2), int(ar_center[1]-roi_range[1]/2))
25)         max_ar_center = (int(ar_center[0]+roi_range[1]/2), int(ar_center[1]+roi_range[1]/2))
26)         roi_ar_center = (int(ar_center[0]-roi_range[0]/2), int(ar_center[1]-roi_range[0]/2), 0 : int(ar_center[0]+roi_range[0]/2), int(ar_center[1]+roi_range[0]/2))
27)         image_cv_ar = image_cv_ar[min_ar_center[1]:max_ar_center[1], min_ar_center[0]:max_ar_center[0], 0 : int(ar_center[0]+roi_range[0]/2), int(ar_center[1]+roi_range[0]/2)]
28)         end_time = time.time()
29)         time_resize = end_time - start_time
30)
31)         # Create a composite image_ar cover to frames_book
32)         start_time = time.time()
33)         image_cv_ar = cv2.resize(image_cv_ar, (image_cv_ar.shape[1], image_cv_ar.shape[0]))
34)         composite_img = cv2.warpPerspective(image_cv_ar, hom, (image_cv_ar.shape[1], image_cv_ar.shape[0]))
35)         image_cv_ar = cv2.resize(image_cv_ar, (image_cv_ar.shape[1], image_cv_ar.shape[0]))
36)         time_composite = end_time - start_time
37)
38)         last_H = H2st_ransac
39)         last_time = time.time()
40)         # Add to list
41)         frame_ar.append(composite_img)
42)
43)     # Profiling report
44)     if i % 10 == 0:
45)         print("i = %d, progress = (%d/%d)*100%" % (i, 100))
46)         report_ar.append("%d frame: %d time_cnn_dt = %f time_brief = %f time_match = %f time_resize = %f time_composite = %f" % (i, 100, time_cnn_dt, time_brief, time_match, time_resize, time_composite))
47)
48)

```

Figure 21: Q3.2 Profiling Code Snippet

```

HW1_my_work > result_profiling > ar_profiling.txt
1   time_locs = 0.001379369909636676
2   time_desc = 3.467086744029075
3
4
5   time_locs = 0.00194452702999115
6   time_desc = 4.099308079981711
7   0 frame:
8   time_crn_dt = 0.11866433301474899
9   time_brief = 7.5705288159660995
10  time_match = 0.36149454896883945
11  time_homo = 0.017591832031030208
12  time_resize = 0.002214112959364894
13  time_composite = 0.006723149970639497
14 -----
15
16  time_locs = 0.0013170900056138635
17  time_desc = 3.367591536953114
18
19  time_locs = 0.0014562139986082911
20  time_desc = 3.8778143319650553
21  20 frame:
22  time_crn_dt = 0.10044569504680112
23  time_brief = 7.248388179054018
24  time_match = 0.3427254020352848
25  time_homo = 0.005725611990783364
26  time_resize = 0.0009581120102666318
27  time_composite = 0.0024902020231820643
28 -----
29
30  time_locs = 0.001287577033508569
31  time_desc = 3.3483148170053028
32
33  time_locs = 0.0014632249949499965
34  time_desc = 3.874739424034875
35  40 frame:
36  time_crn_dt = 0.09615042497171089
37  time_brief = 7.226034865016118
38  time_match = 0.34129151998786256
39  time_homo = 0.005046261008828878
40  time_resize = 0.0012259199866093695
41  time_composite = 0.002812358026858419
42 -----
43  Total Speed = 1/(total_time/total_frames) = 0.12845543164401513 FPS
44

```

Figure 22: Q3.2 Profiling Result

Figure 23 shows the code snippet of change 1 occurring in ec/matchPics.py matchPics(), which I down-sample the two gray images before calculating the feature points. Figure 24 shows the code snippet of change 1 occurring in ec/planarH.py computeH_ransac(), which I scale back the calculated homography to the original scale by multiplying the down_sample_factor on the translation part of the homography, and leaves other parts unchanged as other parts do not change due to down-sampling.

```

# Scale down the image
gray_I1 = cv2.resize(gray_I1, (gray_I1.shape[1] // down_sample_factor, gray_I1.shape[0] // down_sample_factor))
gray_I2 = cv2.resize(gray_I2, (gray_I2.shape[1] // down_sample_factor, gray_I2.shape[0] // down_sample_factor))

#-----Corner Detection-----#

```

Figure 23: Q3.2 Change 1 Code Snippet 1

```

175  if down_sample_factor > 1:
176      bestH2to1[0, 2] *= down_sample_factor
177      bestH2to1[1, 2] *= down_sample_factor
178
179  #test

```

Figure 24: Q3.2 Change 1 Code Snippet 2

Figure 25 shows the change 2 occurring in ec/matchPics.py matchPics(), where I call the OpenCV built-in ORB function to find the corresponding detectors and descriptors.

```

41     #-----Corner Detection-----#
42
43     # TODO: Detect Features in Both Images
44     locs1 = corner_detection(gray_I1, sigma);
45     locs2 = corner_detection(gray_I2, sigma);
46
47     orb = cv2.ORB_create(nfeatures=2000)
48     orb_locs1, orb_desc1 = orb.detectAndCompute(gray_I1, None);
49     orb_locs2, orb_desc2 = orb.detectAndCompute(gray_I2, None);
50
51     orb_locs1_coord = np.array([(int(x.pt[1]), int(x.pt[0])) for x in orb_locs1])
52     orb_locs2_coord = np.array([(int(x.pt[1]), int(x.pt[0])) for x in orb_locs2])
53
54     #print(f"locs1 = {locs1}")
55     #print(f"locs2 = {locs2}")
56     #print(f"locs1.shape = {locs1.shape}")
57     #print(f"locs2.shape = {locs2.shape}")
58
59     #print(f"orb_locs1_coord = {orb_locs1_coord}")
60     #print(f"orb_locs2_coord = {orb_locs2_coord}")
61     #print(f"orb_locs1_coord.shape = {orb_locs1_coord.shape}")
62     #print(f"orb_locs2_coord.shape = {orb_locs2_coord.shape}")
63
64     #-----Descriptor-----#
65     # TODO: Obtain descriptors for the computed feature locations
66     desc1, locs1 = computeBrief(gray_I1, locs1)
67     desc2, locs2 = computeBrief(gray_I2, locs2)
68
69

```

Figure 25: Q3.2 Change 2 Code Snippet

Figure 26 shows the change 3 occuring in ec/matchPics.py matchPics(), where I call the Opencv built-in ORB matching function, called FLANN based Matcher, to find the corresponding matching feature points.

```

70     #-----Matching-----#
71     # TODO: Match features using the descriptors
72     matches = briefMatch(desc1, desc2, ratio)
73     # FLANN parameters for ORB
74     index_params = dict(algorithm=6, # FLANN_INDEX_LSH (for ORB)
75                          table_number=12, # 12
76                          key_size=20, # 20
77                          multi_probe_level=2) # 2
78     search_params = dict(checks=1000) # Specify the number of times the trees in the index should be recursively traversed.
79
80     # Create the FLANN matcher
81     flann = cv2.FlannBasedMatcher(index_params, search_params)
82
83     # Perform matching
84     matches = flann.knnMatch(desc1, desc2, k=2)
85
86     # Apply ratio test to keep the best matches
87     good_matches = []
88     for match in matches:
89         if len(match) == 2: # avoid the crashed case that less than 2 matches were found.
90             if match[0].distance < 0.75 * match[1].distance:
91                 good_matches.append(match[0])
92     matches_orb = good_matches
93
94     # Create a BFMatcher object with default parameters (L2 norm for ORB)
95     bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
96
97     # Match descriptors
98     #matches_orb = bf.match(desc1, desc2)
99     matches_orb_idx1 = np.array([match.queryIdx for match in matches_orb])
100    matches_orb_idx2 = np.array([match.trainIdx for match in matches_orb])
101    matches_orb_idx = np.stack((matches_orb_idx1, matches_orb_idx2), axis=1)
102
103    #print(f"matches.shape = {matches.shape}")
104    #print(f"matches_orb_idx.shape = {matches_orb_idx.shape}")
105    #print(f"matches = {matches}")
106    #rint(f"matches_orb_idx = {matches_orb_idx}")
107
108    matches = matches_orb_idx
109    locs1 = orb_locs1_coord
110    locs2 = orb_locs2_coord
111
112    return matches, locs1, locs2
113

```

Figure 26: Q3.2 Change 3 Code Snippet

Figure 28 - Figure 30 show the code snippet of the main function in ec/ar_ec.py. One can refer to ec/README.md for the command to run the real-time AR application. The final FPS will be automatically reported on the standard output and the report will be stored under the folder specified by `-output_file` argument. The resulting video will also be stored in the same output folder. The final performance un-

der my environment is nearly 31 FPS as shown in Figure 27, which is 242 times faster over the original implementation with 0.128 FPS shown in Figure 22.

```
HW1_my_work > result_ec > ar_ec_profiling.txt
1 Total Speed = 1/(total_time/total_frames) = 31.13726077249852 FPS
```

Figure 27: Q3.2 ar_ec.py Final Speed

```
1 import numpy as np
2 import cv2
3
4 # Import necessary functions
5
6 from loadVid import loadVid
7
8
9 # Q3.2
10
11 #Import necessary functions
12 from opts import get_opts
13 from typing import List, Any, Tuple
14 from displayMatch import displayMatched
15 from planarH import compositeH
16 from matchPics import matchPics
17 from planarH import computeH_ransac
18 import os
19 import time
20 import shutil
21
22 #Write script for Q3.1
23
24 ##### Main-Routine #####
25 #
26 #####
27 def main():
28     # Input arguments/Load input videos
29     opts = get_opts()
30     out_folder = os.path.dirname(opts.output_file)
31     if not os.path.exists(out_folder):
32         os.makedirs(out_folder)
33     else:
34         shutil.rmtree(out_folder)
35         os.makedirs(out_folder)
36
37 frames_ar_source = loadVid(opts.input_src_file)
38 frames_book      = loadVid(opts.input_dst_file)
39
40 # Process each frame
41 frame_result = generalHarryPoterize(frames_ar_source, frames_book, opts, out_folder)
42
43 # Wriet out video
44 writeOutVideo(frame_result, opts)
45
46 # Wriet out three distinct timestamps
47 #frameidx_book_left  = 123
48 #frameidx_book_cnter = 230
49 #frameidx_book_right = 331
50 #cv2.imwrite(f'{out_folder}/Q3_1_ar_result_overlay_left.jpg', frame_result[frameidx_book_left])
51 #cv2.imwrite(f'{out_folder}/Q3_1_ar_result_overlay_right.jpg', frame_result[frameidx_book_right])
52 #cv2.imwrite(f'{out_folder}/Q3_1_ar_result_overlay_cnter.jpg', frame_result[frameidx_book_cnter])
53
54 #####
55 # Sub-Routine #
56 #####
```

Figure 28: Q3.2 ar_ec.py Code Snippet 1

```

57 def writeOutVideo(frame_result: List[Any], opts) -> None:
58     fourcc = cv2.VideoWriter_fourcc(*'XVID')
59     frame_size = (frame_result[0].shape[1], frame_result[0].shape[0])
60     out = cv2.VideoWriter(opts.output_file, fourcc, 30.0, frame_size)
61
62     for frame in frame_result:
63         out.write(frame)
64
65 def generalHarryPotterize(frame_ar_source: List[Any], frames_book: List[Any], opts, out_folder: str) -> List[Any]:
66     image_cv_cover = cv2.imread('../data/cv_cover.jpg')
67     min_len = min(len(frame_ar_source), len(frames_book))
68     frame_result = []
69     down_sample_factor = opts.down_sample_factor
70     frame_interval_same_H = opts.frame_interval_same
71     total_time = 0
72
73     for i in range(min_len):
74         time_loop_start = time.perf_counter()
75         # Start of the Computation
76         # Compute Homography
77         if not frame_interval_same_H == 0:
78             matches, locs1, locs2 = matchPics(frames_book[i], image_cv_cover, opts, down_sample_factor)
79             x1 = locs1[matches[:, 0], :]
80             x2 = locs2[matches[:, 1], :]
81             x1_correct_pt = np.flip(x1, axis=1)
82             x2_correct_pt = np.flip(x2, axis=1)
83
84             try:
85                 H2to1_ransac, inliers = computeH_ransac(x1_correct_pt, x2_correct_pt, opts, True, 10, down_sample_factor)
86             except ValueError:
87                 H2to1_ransac = last_H
88         else:
89             H2to1_ransac = last_H
90
91         # Resize the frame of ar_source to the size of frame of book
92         image_cv_height, image_cv_width = image_cv_cover.shape[0], image_cv_cover.shape[1]
93         ar_center = (int(frame_ar_source[i].shape[1]/2), int(frame_ar_source[i].shape[0]/2)) #(x, y)
94         roi_range = (200, 200)
95
96         image_ar_cover = frame_ar_source[i][max(ar_center[1]-int(roi_range[1]/2), 48) : min(ar_center[1]+int(roi_range[1]/2), 310), max(ar_center[0]-int(roi_range[0]/2), 0) : min(ar_center[0]+int(roi_range[0]/2), 640)]
97         resize_image_ar_cover = cv2.resize(image_ar_cover, (image_cv_cover.shape[1], image_cv_cover.shape[0]))
98
99         # Warp & Composite resize image ar cover to frames book
100        composite_img = compositeH(H2to1_ransac, resize_image_ar_cover, frames_book[i])
101
102        # End of the computation
103        time_loop_end = time.perf_counter()
104        total_time += (time_loop_end-time_loop_start)
105
106
107        # Add to result
108        frame_result.append(composite_img)
109        last_H = H2to1_ransac
110
111        cv2.imshow("composite_img", composite_img)
112        cv2.waitKey(1)
113        #cv2.destroyAllWindows()

```

Figure 29: Q3.2 ar_ec.py Code Snippet 2

```

114
115     # Stop if 'q' is pressed
116     if cv2.waitKey(1) & 0xFF == ord('q'):
117         break
118
119     result_fps = f"Total Speed = 1/(total_time/total_frames) = {1/(total_time/min_len)} FPS"
120     print(result_fps)
121     with open(f'{out_folder}/ar_ec_profiling.txt', "w") as f:
122         f.write(result_fps)
123     return frame_result
124
125 #-----Execution-----#
126 if __name__ == '__main__':
127     main()

```

Figure 30: Q3.2 ar_ec.py Code Snippet 3

There is another input argument that I originally developed to further speedup, but I currently do not use it. It is the `-frame_interval_same`. It is used to bypass some frames by using the same homography as previous frame. The reason that one can do it is to assume that the targeted frames will not have large movement over '`frame_interval_same`' frames. Under this assumption, the mapping of homography can be deemed as nearly unchanged. The default value of this argument is set to 2 as I think that `data/book.mov` does not have big motion between continuous frames. Although I can use it to further speedup my performance, which may be up to 50 FPS, I do not turn it on by setting this argument to 1, as the command specified in `ec/README.md`, because I have already reached the performance over 30 FPS without the usage of this argument.

Q4 at page 10

Ans:

Figure 31 shows my input left image, and Figure 32 shows my input right image, and Figure 33 shows the result of generated panorama. Figure 34 - Figure 37 show my snippets of code that implement panorama.py.



Figure 31: Q4. My Input Left Image



Figure 32: Q4. My Input Right Image



Figure 33: Q4. Result of Panorama

```

HW1_my_work> python > panorama.py > main
 1 import numpy as np
 2 import cv2
 3
 4 # Import necessary functions
 5 from opts import get_opts
 6 from typing import List, Any, Tuple
 7 from displayMatch import displayMatched
 8 from planarH import computeH_ransac
 9 from matchPics import matchPics
10 import os
11 import time
12 import shutil
13 import skimage.color
14 import numpy.typing as npt
15
16
17
18 # Q4
19
20 ##### Main-Routine #####
21 #####
22 #####
23 def main():
24     # Input arguments/Load input videos
25     opts = get_opts()
26     out_folder = os.path.dirname(opts.output_file)
27     if not os.path.exists(out_folder):
28         os.makedirs(out_folder)
29     else:
30         #shutil.rmtree(out_folder)
31         #os.makedirs(out_folder)
32         pass
33
34     image_left    = cv2.imread(opts.input_left_img_file)
35     image_right   = cv2.imread(opts.input_right_img_file)
36
37     # Process each frame
38     generatePanorama(image_right, image_left, opts)
39
40 #####
41     # Sub-Routine #####
42 #####
43 def generatePanorama(img1: npt.NDArray[Tuple[Any, Any]], img2: npt.NDArray[Tuple[Any, Any]], opts) -> None:
44     down_sample_factor = 1
45
46     # Start of the Computation
47     # Compute Features Matching
48     matches, locs1, locs2 = matchPicsPanorama(img1, img2, opts, down_sample_factor)
49     #matches, locs1, locs2 = matchPics(img1, img2, opts)
50     x1 = locs1[matches[:, 0], :]
51     x2 = locs2[matches[:, 1], :]
52     x1_correct_pt = np.flip(x1, axis=1)
53     x2_correct_pt = np.flip(x2, axis=1)
54
55     # Compute Homography
56     H2to1_ransac, inliers = computeH_ransac(x1_correct_pt, x2_correct_pt, opts, True, 10)

```

Figure 34: Q4. Code Snippet 1

```

57     # Warp & Composite img2 to frames_book
58     composite_img = compositeHPanorama(H2to1_ransac, img2, img1)
59
60     #cv2.imshow("Panorama Result", composite_img)
61     #cv2.waitKey(0)
62     cv2.imwrite(f"(opts.output_file)", composite_img)
63
64 def compositeHPanorama(H2to1, template, img):
65     composite_width = template.shape[1] + img.shape[1]
66     composite_height = img.shape[0]
67     composite_img = np.zeros((composite_height, composite_width, 3), dtype=np.uint8)
68
69     # TODO: Create mask of same size as template
70     mask = np.full((template.shape[0], template.shape[1], 1), 255, dtype=np.uint8)
71
72     # TODO: Warp mask by appropriate homography
73     warped_mask = cv2.warpPerspective(mask, H2to1, (composite_width, composite_height))
74
75     # TODO: Warp template by appropriate homography
76     warped_template = cv2.warpPerspective(template, H2to1, (composite_width, composite_height))
77
78     # TODO: Use mask to combine the warped template and the image
79     ...
80     inverse_mask = cv2.bitwise_not(warped_mask)
81     masked_template = cv2.bitwise_and(warped_template, warped_template, mask=warped_mask)
82     masked_img = cv2.bitwise_and(img, img, mask=inverse_mask)
83     composite_img = cv2.add(masked_template, masked_img)
84
85     ...
86     last_column = -1
87     for i in range(composite_img.shape[0]):
88         for j in range(composite_img.shape[1]):
89             if warped_mask[i, j] == 255:
90                 composite_img[i, j] = warped_template[i, j]
91                 last_column = max(j, last_column)
92             else:
93                 if i < img.shape[0] and j < img.shape[1]:
94                     composite_img[i, j] = img[i, j]
95                     last_column = max(j, last_column)
96
97     # Crop the image to remove blank areas
98     composite_img = composite_img[:, :last_column+1]
99
100    return composite_img
101
102 def matchPicsPanorama(I1, I2, opts, down_sample_factor: int):
103     """
104     Match features across images
105
106     Input
107     -----
108     I1, I2: Source images
109     opts: Command line args
110

```

Figure 35: Q4. Code Snippet 2

```

111     Returns
112     -----
113     matches: List of indices of matched features across I1, I2 [p x 2]
114     locs1, locs2: Pixel coordinates of matches [N x 2]
115     """
116
117     ratio = opts.ratio #'ratio for BRIEF feature descriptor'
118     sigma = opts.sigma #'threshold for corner detection using FAST feature detector'
119
120
121     # TODO: Convert Images to GrayScale
122     # Convert BGR order (from cv2.imread()) to RGB order
123     image1_rgb = I1[:, :, [2, 1, 0]]
124     image2_rgb = I2[:, :, [2, 1, 0]]
125     gray_I1 = (skimage.color.rgb2gray(image1_rgb) * 255).astype('uint8')
126     gray_I2 = (skimage.color.rgb2gray(image2_rgb) * 255).astype('uint8')
127
128     # Scale down the image
129     gray_I1 = cv2.resize(gray_I1, (gray_I1.shape[1] // down_sample_factor, gray_I1.shape[0] // down_sample_factor))
130     gray_I2 = cv2.resize(gray_I2, (gray_I2.shape[1] // down_sample_factor, gray_I2.shape[0] // down_sample_factor))
131
132
133     #-----Corner Detection-----#
134
135     # TODO: Detect Features in Both Images
136     #locs1 = corner_detection(gray_I1, sigma);
137     #locs2 = corner_detection(gray_I2, sigma);
138
139     orb = cv2.ORB_create(nfeatures=20000)
140     orb_locs1, orb_desc1 = orb.detectAndCompute(gray_I1, None);
141     orb_locs2, orb_desc2 = orb.detectAndCompute(gray_I2, None);
142
143     orb_locs1_coord = np.array([(int(x.pt[1]), int(x.pt[0])) for x in orb_locs1])
144     orb_locs2_coord = np.array([(int(x.pt[1]), int(x.pt[0])) for x in orb_locs2])
145
146     #print(f"locs1 = {locs1}")
147     #print(f"locs2 = {locs2}")
148     #print(f"locs1.shape = {locs1.shape}")
149     #print(f"locs2.shape = {locs2.shape}")
150
151     #print(f"orb_locs1_coord = {orb_locs1_coord}")
152     #print(f"orb_locs2_coord = {orb_locs2_coord}")
153     #print(f"orb_locs1_coord.shape = {orb_locs1_coord.shape}")
154     #print(f"orb_locs2_coord.shape = {orb_locs2_coord.shape}")
155
156     #-----Descriptor-----#
157     # TODO: Obtain descriptors for the computed feature locations
158     #desc1, locs1 = computeBrief(gray_I1, locs1)
159     #desc2, locs2 = computeBrief(gray_I2, locs2)
160
161
162     #-----Matching-----#
163     # TODO: Match features using the descriptors
164     #matches = briefMatch(desc1, desc2, ratio)
165
166     index_params = dict(algorithm=6, # FLANN_INDEX_LSH (for ORB)

```

Figure 36: Q4. Code Snippet 3

```

111     Returns
112     -----
113     matches: List of indices of matched features across I1, I2 [p x 2]
114     locs1, locs2: Pixel coordinates of matches [N x 2]
115     """
116
117     ratio = opts.ratio #'ratio for BRIEF feature descriptor'
118     sigma = opts.sigma #'threshold for corner detection using FAST feature detector'
119
120
121     # TODO: Convert Images to GrayScale
122     # Convert BGR order (from cv2.imread()) to RGB order
123     image1_rgb = I1[:, :, [2, 1, 0]]
124     image2_rgb = I2[:, :, [2, 1, 0]]
125     gray_I1 = (skimage.color.rgb2gray(image1_rgb) * 255).astype('uint8')
126     gray_I2 = (skimage.color.rgb2gray(image2_rgb) * 255).astype('uint8')
127
128     # Scale down the image
129     gray_I1 = cv2.resize(gray_I1, (gray_I1.shape[1] // down_sample_factor, gray_I1.shape[0] // down_sample_factor))
130     gray_I2 = cv2.resize(gray_I2, (gray_I2.shape[1] // down_sample_factor, gray_I2.shape[0] // down_sample_factor))
131
132
133     #-----Corner Detection-----#
134
135     # TODO: Detect Features in Both Images
136     #locs1 = corner_detection(gray_I1, sigma);
137     #locs2 = corner_detection(gray_I2, sigma);
138
139     orb = cv2.ORB_create(nfeatures=20000)
140     orb_locs1, orb_desc1 = orb.detectAndCompute(gray_I1, None);
141     orb_locs2, orb_desc2 = orb.detectAndCompute(gray_I2, None);
142
143     orb_locs1_coord = np.array([(int(x.pt[1]), int(x.pt[0])) for x in orb_locs1])
144     orb_locs2_coord = np.array([(int(x.pt[1]), int(x.pt[0])) for x in orb_locs2])
145
146     #print(f"locs1 = {locs1}")
147     #print(f"locs2 = {locs2}")
148     #print(f"locs1.shape = {locs1.shape}")
149     #print(f"locs2.shape = {locs2.shape}")
150
151     #print(f"orb_locs1_coord = {orb_locs1_coord}")
152     #print(f"orb_locs2_coord = {orb_locs2_coord}")
153     #print(f"orb_locs1_coord.shape = {orb_locs1_coord.shape}")
154     #print(f"orb_locs2_coord.shape = {orb_locs2_coord.shape}")
155
156     #-----Descriptor-----#
157     # TODO: Obtain descriptors for the computed feature locations
158     #desc1, locs1 = computeBrief(gray_I1, locs1)
159     #desc2, locs2 = computeBrief(gray_I2, locs2)
160
161
162     #-----Matching-----#
163     # TODO: Match features using the descriptors
164     #matches = briefMatch(desc1, desc2, ratio)
165
166     index_params = dict(algorithm=6, # FLANN_INDEX_LSH (for ORB)

```

Figure 37: Q4. Code Snippet 3