Assignment 3

CMSC 691 — Computer Vision

Faisal Rasheed Khan

VB02734

vb02734@umbc.edu

Question 1--

1. Homogeneous 3x3 matrix with angle θ at point (a,b)
   Deriving homogeneous transformation for rotating a point (x,y) around specific point (a,b) with angle θ, it involves 3 transformations.
   i. Translation of (x,y) that move (a,b) to origin
   ii. Rotation around the origin w.r.t θ
   iii. Translation back to original position

   i. Translate point (x,y) such that (a,b) moves to origin

   $$T_1 = \begin{bmatrix} 1 & 0 & -a \\ 0 & 1 & -b \\ 0 & 0 & 1 \end{bmatrix}$$

   ii. Rotation around the origin w.r.t θ

   $$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

   iii. Translation back to original position

   $$T_2 = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix}$$

   Overall,

   $M = T_2 * R * T_1 =$

   $$M = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -a \\ 0 & 1 & -b \\ 0 & 0 & 1 \end{bmatrix}$$

   $$M = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & -a\cos\theta + b\sin\theta \\ \sin\theta & \cos\theta & -a\sin\theta - b\cos\theta \\ 0 & 0 & 1 \end{bmatrix}$$

   $$M = \begin{bmatrix} \cos\theta & -\sin\theta & -a\cos\theta + b\sin\theta + a \\ \sin\theta & \cos\theta & -a\sin\theta - b\cos\theta + b \\ 0 & 0 & 1 \end{bmatrix}$$

   $$\begin{bmatrix} x^1 \\ y^1 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & -a\cos\theta + b\sin\theta + a \\ \sin\theta & \cos\theta & -a\sin\theta - b\cos\theta + b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2. P1 = (1,1), P2 = (2,1), P3 = (2,2), P4 = (1,2)
   At point P2 = (2,1) rotation with 45 degrees.
   The matrix form can be written as,

$$\begin{bmatrix} x^1 \\ y^1 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\ \theta & -sin\,\theta & -a\,cos\,\theta + b\sin\theta + a \\ sin\,\theta & cos\,\theta & -a\sin\theta - b\cos\theta + b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

   We have (a,b) = (2,1),

$$\begin{bmatrix} x^1 \\ y^1 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\ 45 & -sin\,45 & -2\,cos\,45 + \sin 45 + 2 \\ sin\,45 & cos\,45 & -2\sin 45 - \cos 45 + 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x^1 \\ y^1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & -2\ \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}} + 2 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & -2\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}} + 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x^1 \\ y^1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & \frac{-1+2\sqrt{2}}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{-3+\sqrt{2}}{\sqrt{2}} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x^1 \\ y^1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{x - y - 1 + 2\sqrt{2}}{\sqrt{2}} \\ \frac{x + y - 3 + \sqrt{2}}{\sqrt{2}} \\ 1 \end{bmatrix}$$

   Substitute all the points in the above matrix form to get the new points,
   $P1$ = (1,1)

$$P_1' = \begin{bmatrix} \frac{1 - 1 - 1 + 2\sqrt{2}}{\sqrt{2}} \\ \frac{1 + 1 - 3 + \sqrt{2}}{\sqrt{2}} \\ 1 \end{bmatrix}$$

$$P_1' = \begin{bmatrix} \frac{2\sqrt{2} - 1}{\sqrt{2}} \\ \frac{\sqrt{2} - 1}{\sqrt{2}} \\ 1 \end{bmatrix}$$

$$P_1' = \left(\frac{2\sqrt{2} - 1}{\sqrt{2}}, \frac{\sqrt{2} - 1}{\sqrt{2}} \right)$$

   $P2$ = (2,1)

$$P_2' = \begin{bmatrix} \frac{2 - 1 - 1 + 2\sqrt{2}}{\sqrt{2}} \\ \frac{2 + 1 - 3 + \sqrt{2}}{\sqrt{2}} \\ 1 \end{bmatrix}$$

$$P_2' = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$$

$P_2' = (2,1)$

$P3 = (2,2)$

$$P_3' = \begin{bmatrix} \frac{2-2-1+2\sqrt{2}}{\sqrt{2}} \\ \frac{2+2-3+\sqrt{2}}{\sqrt{2}} \\ 1 \end{bmatrix}$$

$$P_3' = \begin{bmatrix} \frac{2\sqrt{2}-1}{\sqrt{2}} \\ \frac{\sqrt{2}+1}{\sqrt{2}} \\ 1 \end{bmatrix}$$

$P_1' = \left(\frac{2\sqrt{2}-1}{\sqrt{2}}, \frac{\sqrt{2}+1}{\sqrt{2}} \right)$

P4 = (1,2)

$$P_4' = \begin{bmatrix} \frac{1-2-1+2\sqrt{2}}{\sqrt{2}} \\ \frac{1+2-3+\sqrt{2}}{\sqrt{2}} \\ 1 \end{bmatrix}$$

$P_4' = (2-\sqrt{2}\ , 1)$

3. $x^1 = a\,x + b\,y + t_x + \propto x^2 + \beta\, y^2$

   $y^1 = c\,x + d\,y + t_y + \gamma\, x^2 + \theta\, y^2$

   To find point of correspondence we need to do " Ah = 0 "

   The above equations can be written in the matrix form as,

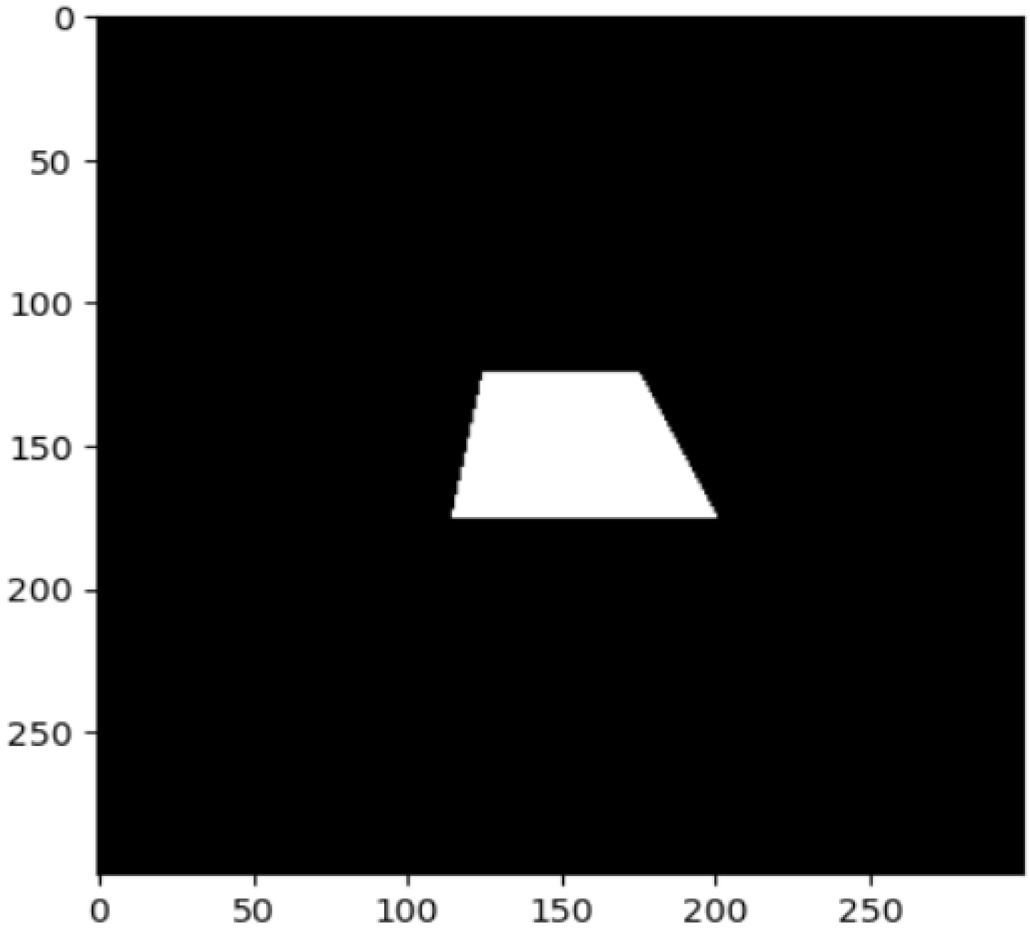
   $$\begin{bmatrix} x & y & 0 & 0 & 1 & 0 & x^2 & y^2 & 0 & 0 \\ 0 & 0 & x & y & 0 & 1 & 0 & 0 & x^2 & y^2 \end{bmatrix} * [\,a \quad b \quad c \quad d \quad t_x \quad t_y \quad \alpha \quad \beta \quad \gamma \quad \theta\,]^T$$

   A * h = 0

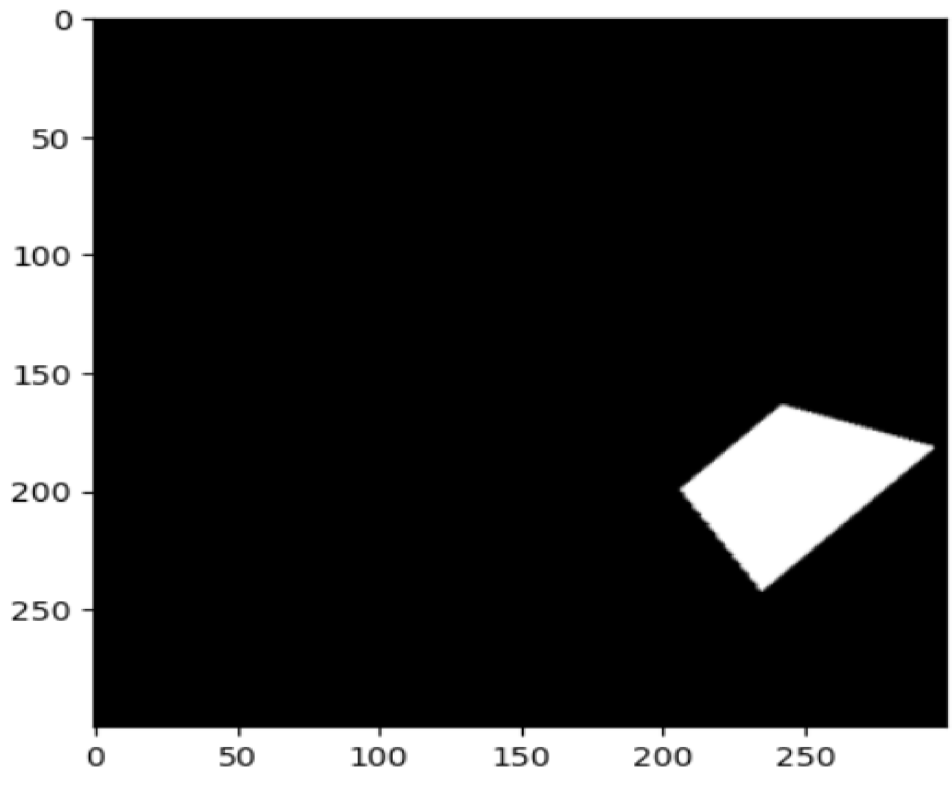
   Solve for A*h = 0,

   We have 10 unknowns to solve and we need atleast 5 correspondence points to solve the problem.

4. Qudrilateral:



Quadrilateral After applying transformations:



```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# a black image
image_b = np.zeros((300, 300, 3), dtype=np.uint8)

# points for an irregular quadrilateral
pts = np.array([[125, 125], [115, 175], [200, 175], [175, 125]], np.int32)
pts = pts.reshape((-1, 1, 2))

# quadrilateral in white
cv2.fillPoly(image_b, [pts], (255, 255, 255))

plt.imshow(image_b)

# Translate
M_translate = np.float32([[1, 0, 30], [0, 1, 100]])
translated = cv2.warpAffine(image_b, M_translate, (300, 300))

# Rotate
center = (150, 150)
M_rotate = cv2.getRotationMatrix2D(center, 45, 1.0)
rotated = cv2.warpAffine(translated, M_rotate, (300, 300))

# Display
plt.imshow(rotated)
```

Question 2---

2.1 ,2.2, 2.3, 2.4---

Image A keypoints:

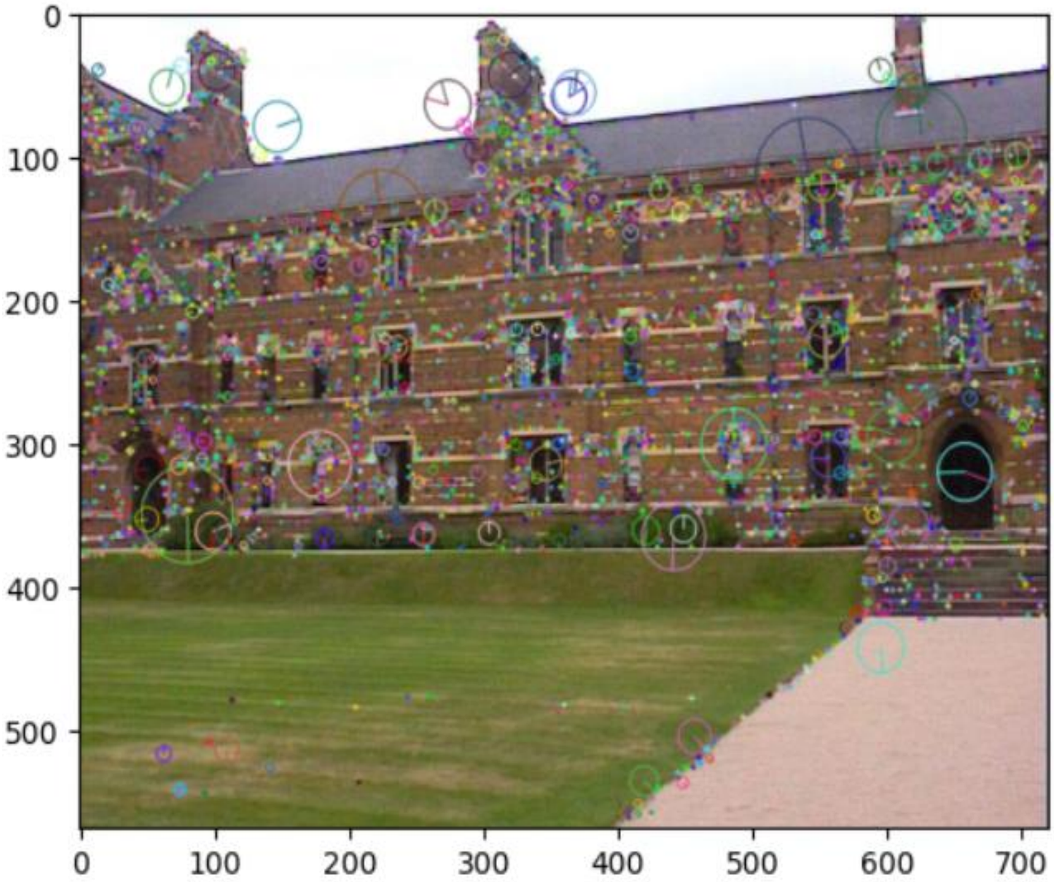
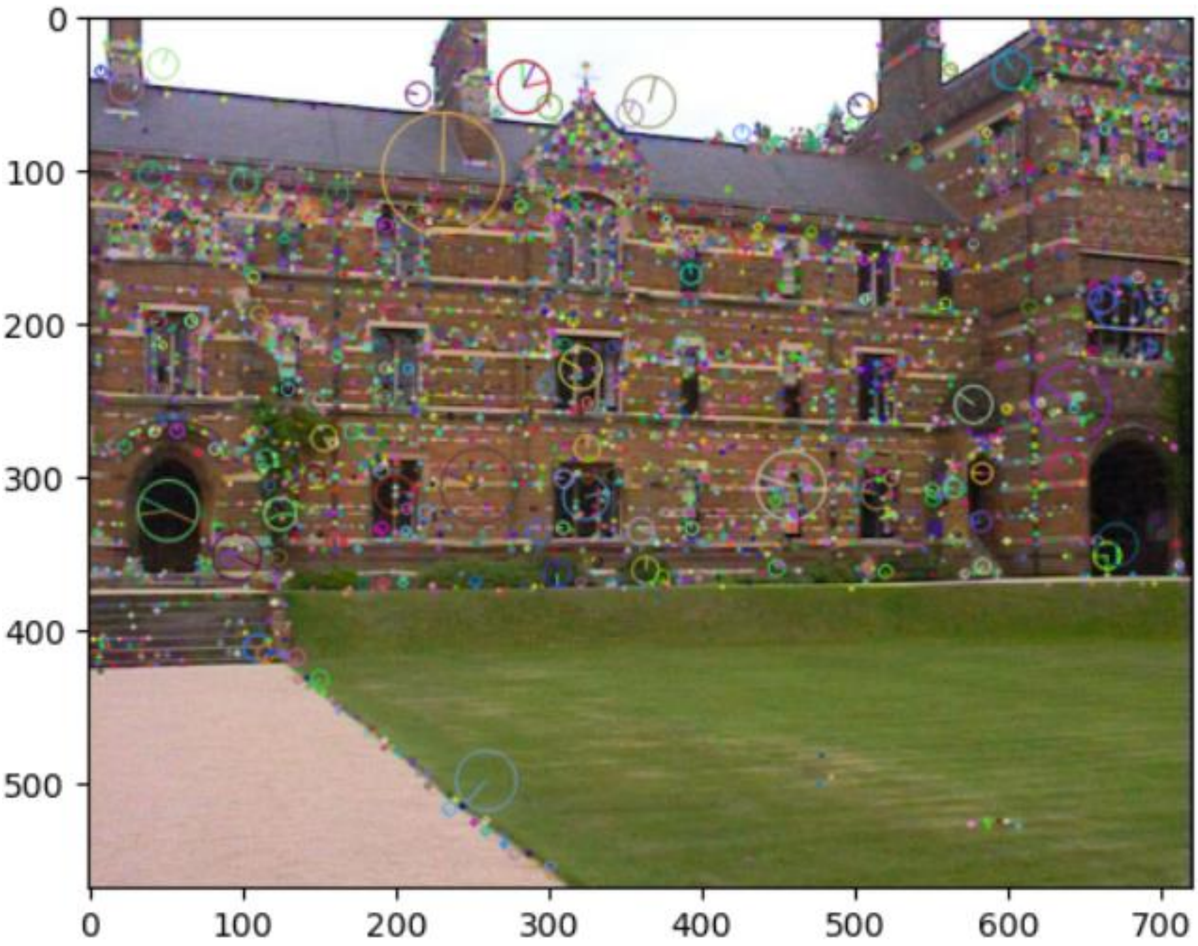
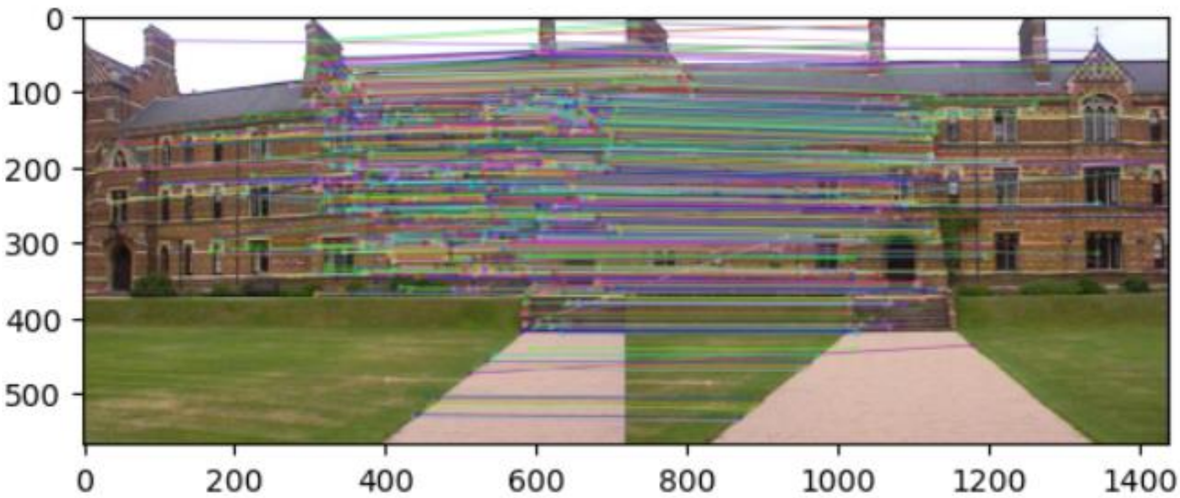


Image B keypoints:

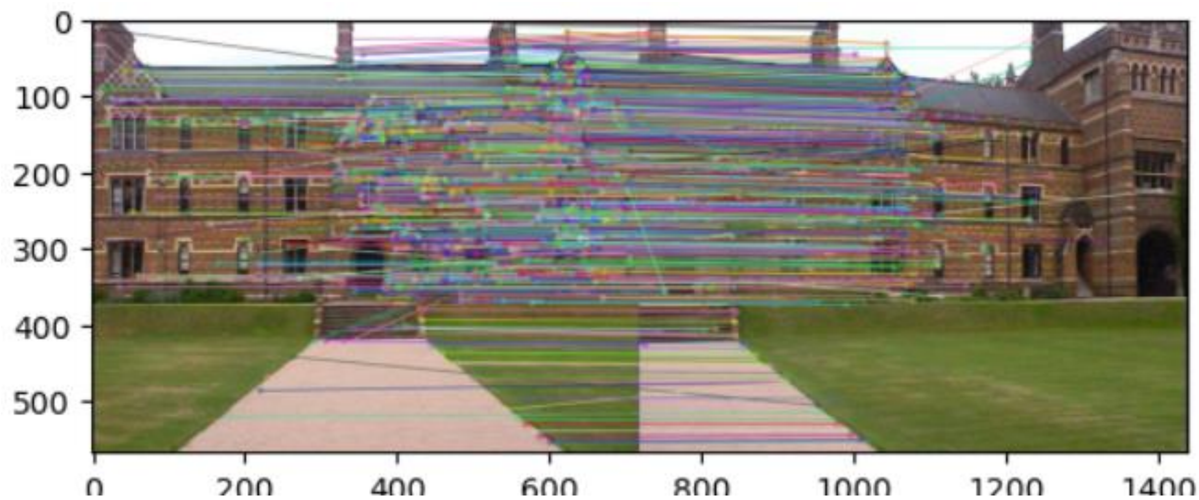
Image C keypoints:



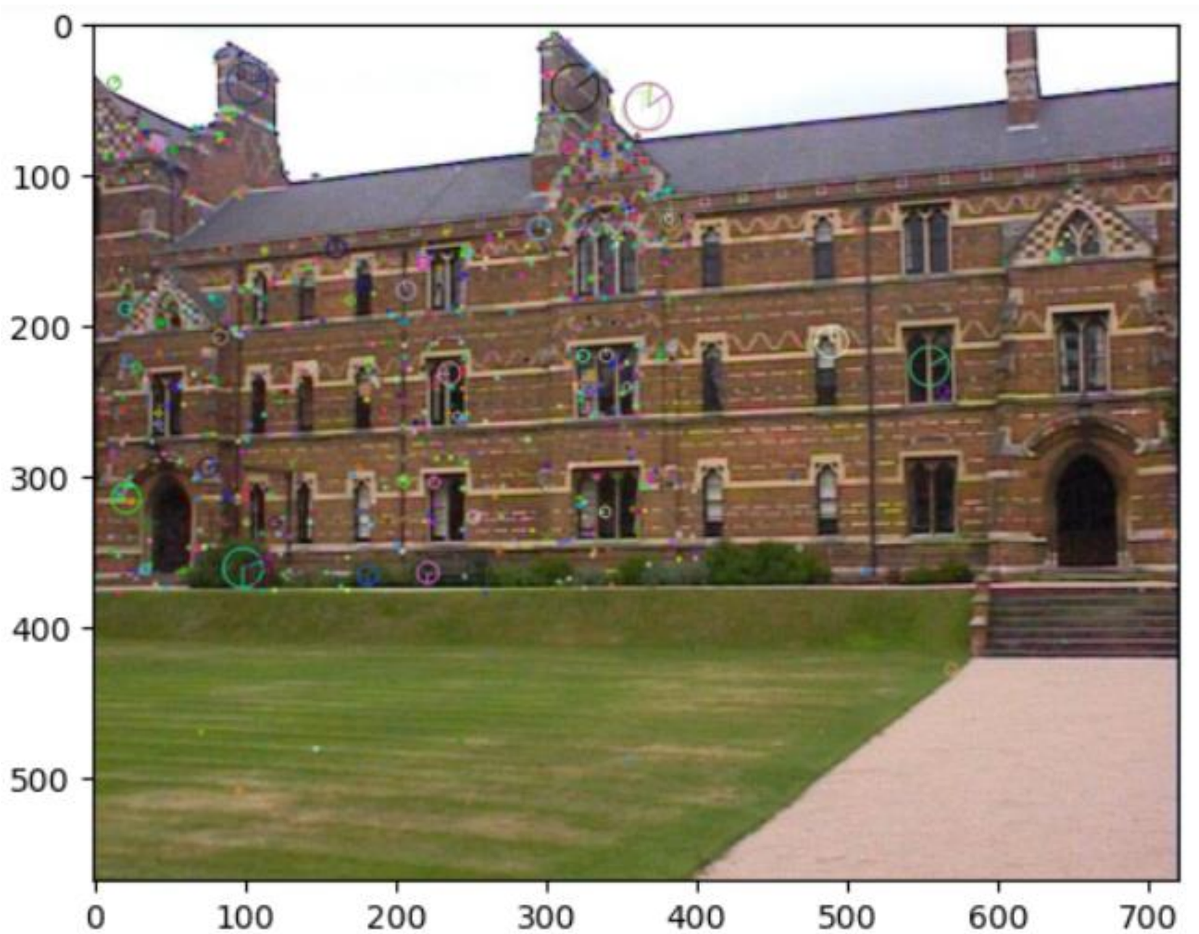
Matching features between Image A and Image B:



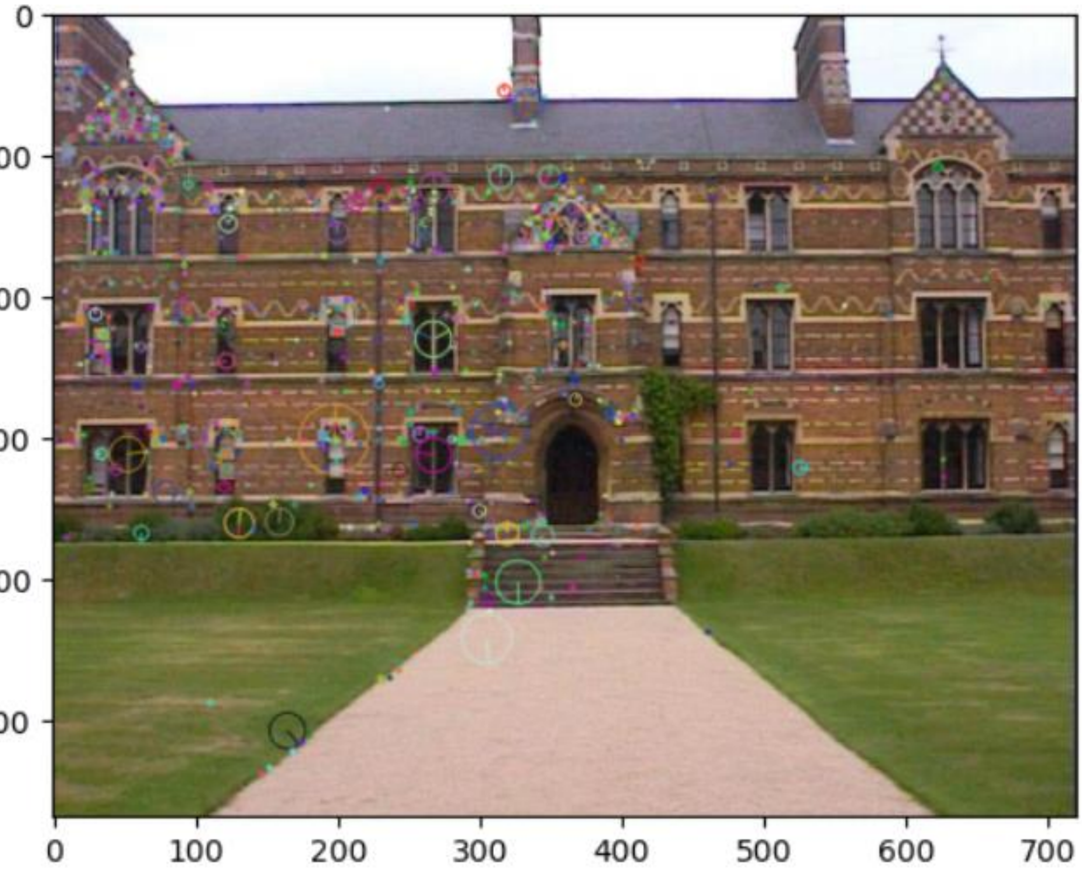
Matching features between Image B and Image C:



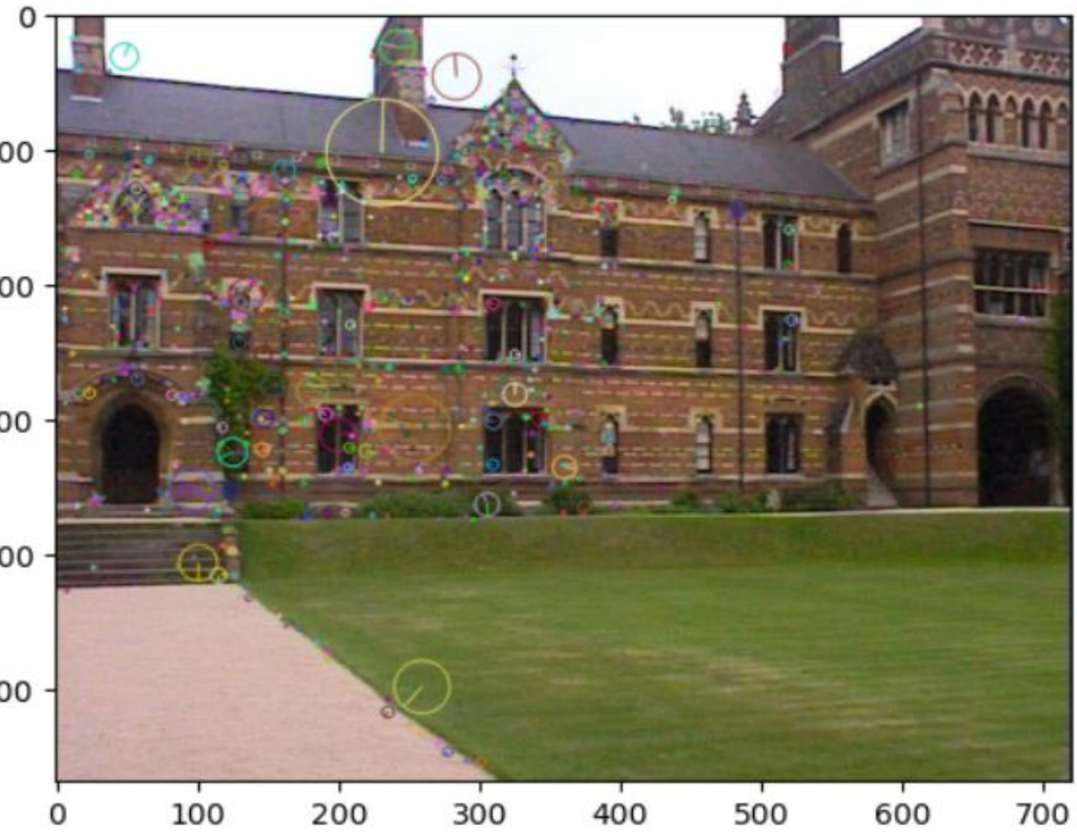
Keypoints of image A after similar matches:



Keypoints of image B after similar matches:



Keypoints of image C after similar matches:



```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Load images
image1 = cv2.imread('C:/Users/juver/Downloads/hw3/keble_a.jpg')
image2 = cv2.imread('C:/Users/juver/Downloads/hw3/keble_b.jpg')
image3 = cv2.imread('C:/Users/juver/Downloads/hw3/keble_c.jpg')
#SIFT
sift = cv2.SIFT_create()
# Method 2: Directly finding keypoints and descriptors
keypoints1, descriptors1 = sift.detectAndCompute(image1, None)
keypoints2, descriptors2 = sift.detectAndCompute(image2, None)
keypoints3, descriptors3 = sift.detectAndCompute(image3, None)
img=cv2.drawKeypoints(image1,keypoints1,None,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
# BGR to RGB
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img_rgb)
def match_feature_descriptors(des1,des2):
    # BFMatcher with default params
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des1,des2,k=2)
    # Apply ratio test
    good = []
    for m,n in matches:
        if m.distance < 0.75*n.distance:
            good.append([m])
    return good
good12 = match_feature_descriptors(descriptors1, descriptors2)
good23 = match_feature_descriptors(descriptors2, descriptors3)

img3 = cv2.drawMatchesKnn(image1,keypoints1,image2,keypoints2,good12,None,flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
img3 = cv2.cvtColor(img3, cv2.COLOR_BGR2RGB)
plt.imshow(img3),plt.show()
```

```python
# threshold
threshold = 4054
# Extracting keypoints from cv2.DMatch objects
if isinstance(good12[0], cv2.DMatch):
    keypoints1_to_draw = [keypoints1[match.trainIdx] for match in good12 if match.trainIdx <= threshold]
else:
    keypoints1_to_draw = [keypoints1[match[0].trainIdx] for match in good12 if match[0].trainIdx <= threshold]
# Draw keypoints
img = cv2.drawKeypoints(image1, keypoints1_to_draw, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img_rgb)
plt.show()


# Extracting keypoints from cv2.DMatch objects
keypoints2_to_draw = [keypoints2[match[0].trainIdx] for match in good12]
# Draw keypoints
img = cv2.drawKeypoints(image2, keypoints2_to_draw, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

plt.imshow(img_rgb)
plt.show()
```

Image A warped perspective with the Homograph matrix:

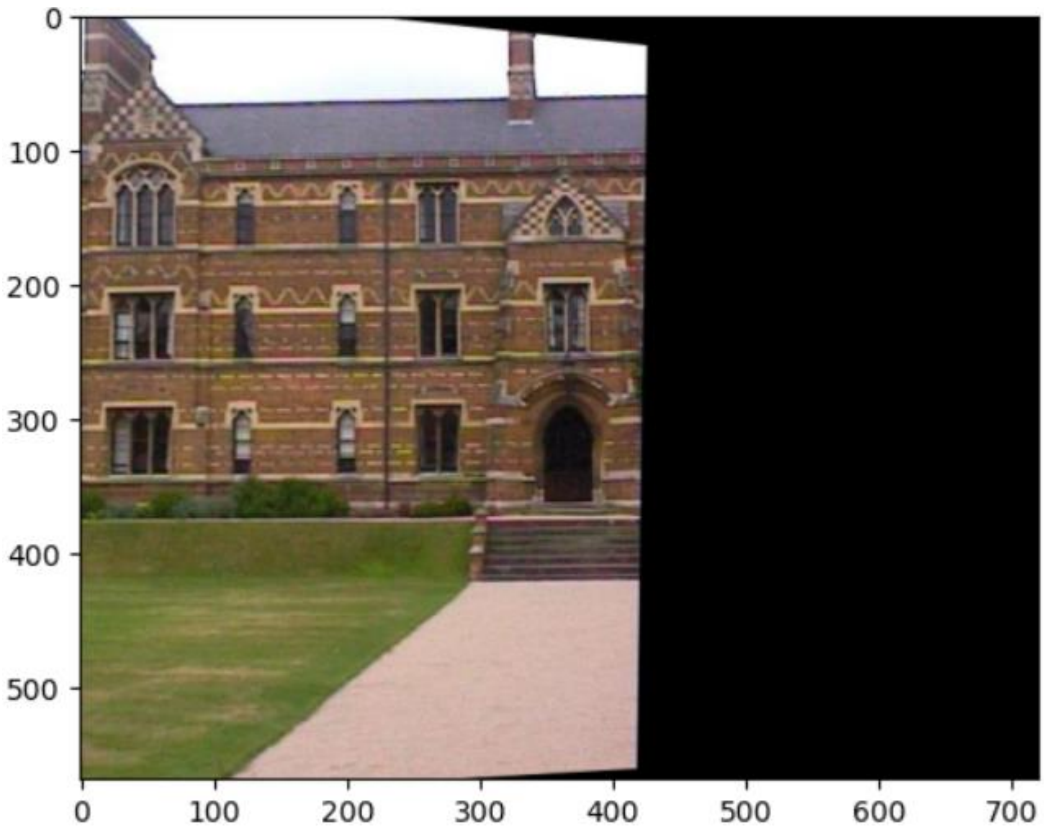
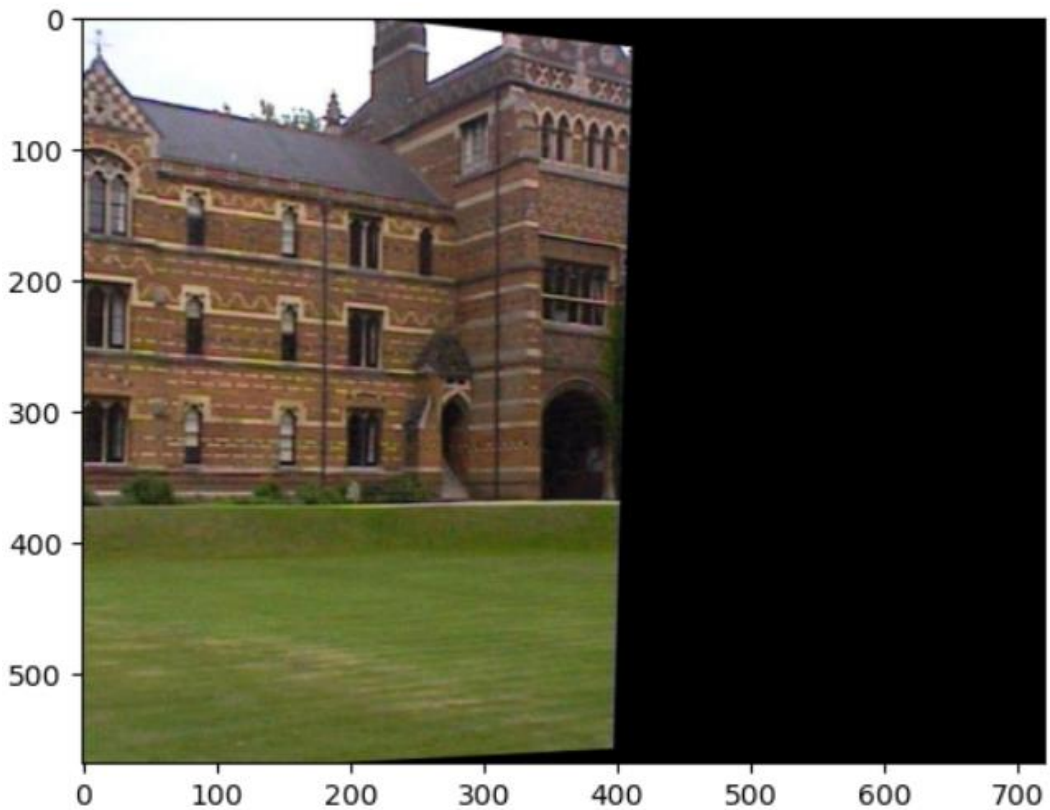
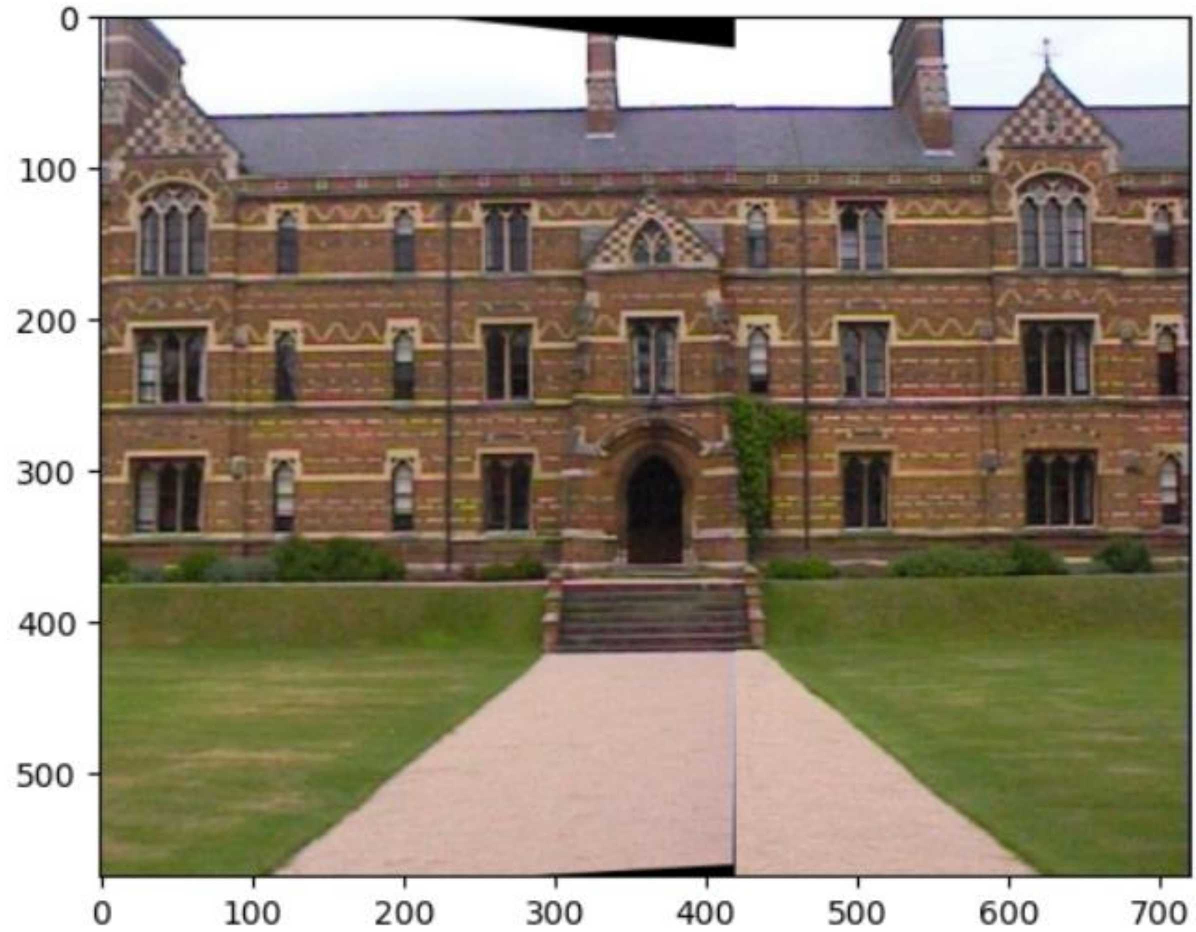


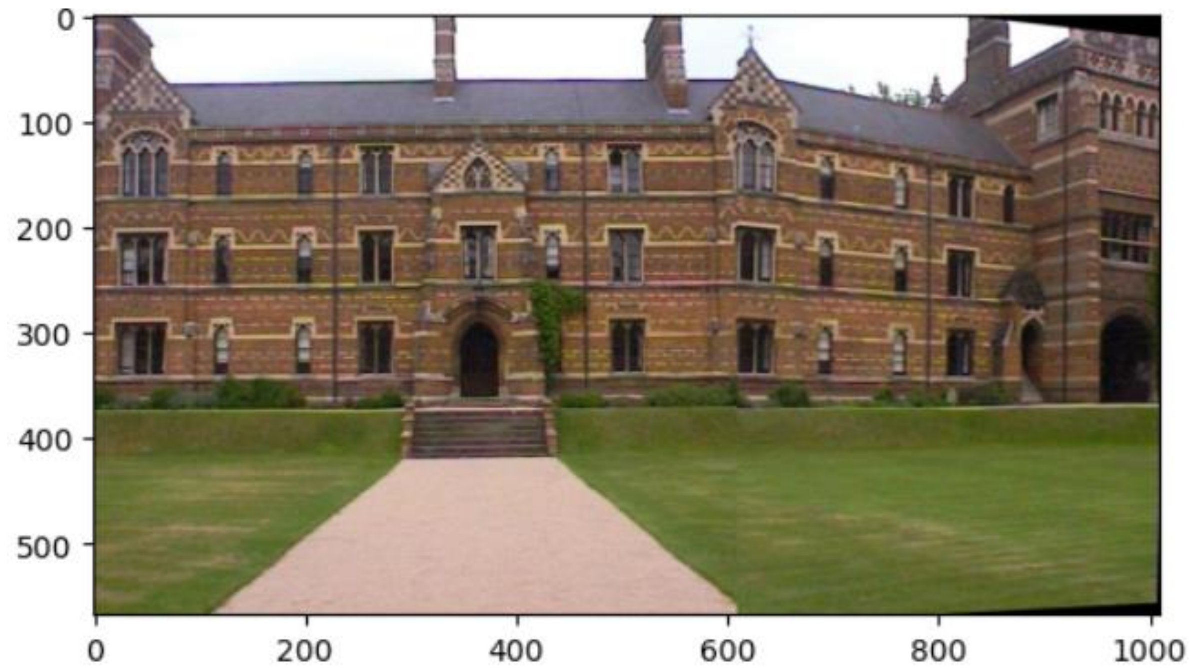
Image C warped perspective with the Homograph matrix:



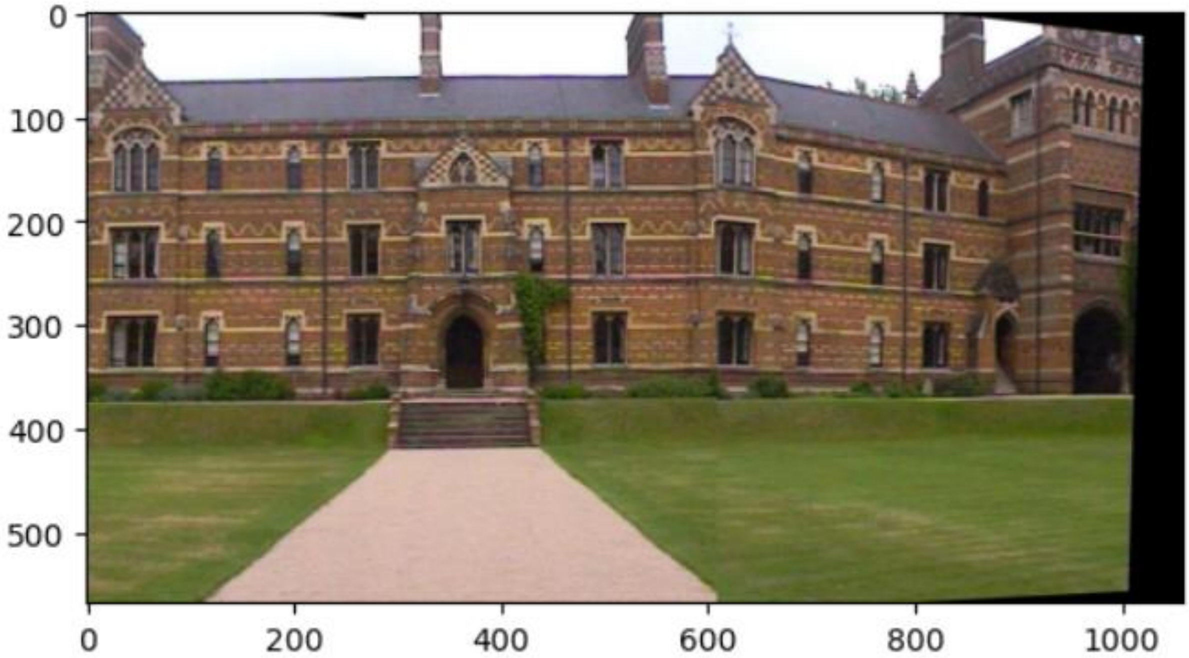
Stitching Image A and Image B:



Stitching Image B and Image C:



Creating Mosaic by stitching Image A,B,C:

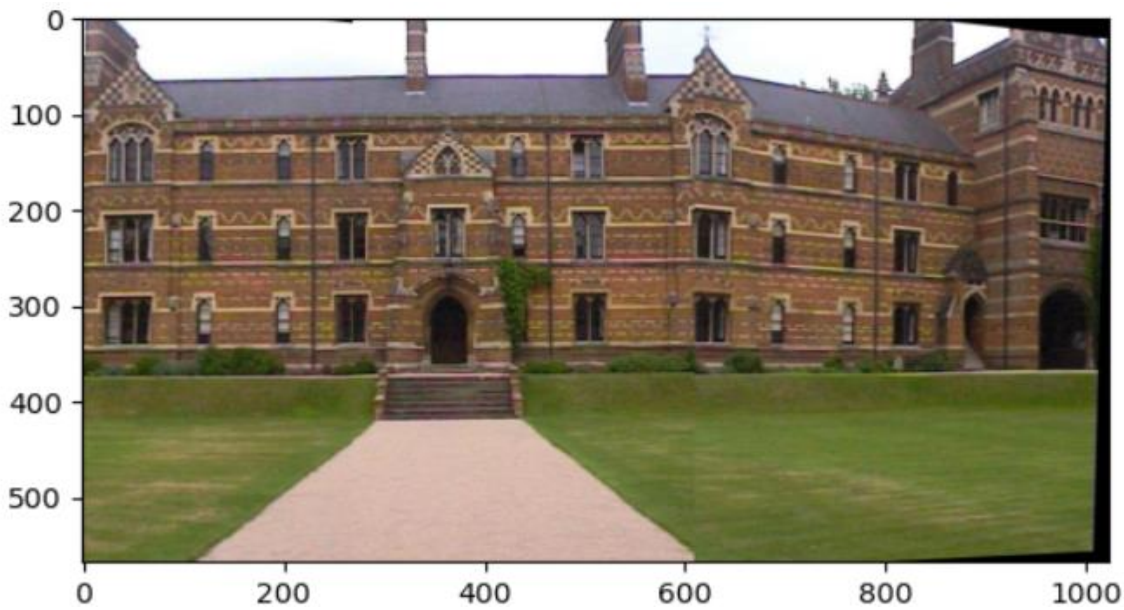

0
100
200
300
400
500
0
200
400
600
800
1000

```python
def computeH(im1_pts, im2_pts):
    #for solving Ah = 0
    A = []
    for i in range(im1_pts.shape[1]):
        x, y = im1_pts[:, i]
        u, v = im2_pts[:, i]
        A.append([-x, -y, -1, 0, 0, 0, x*u, y*u, u])
        A.append([0, 0, 0, -x, -y, -1, x*v, y*v, v])
    A = np.array(A)
    # Perform SVD on A
    _, _, V = np.linalg.svd(A)
    H = V[-1, :].reshape(3, 3)
    return H
def ransac(corr, thresh, iter):
    max_inliers = []
    H_max = None
    for _ in range(iter):
        # Randomly sample 4 point correspondences
        random_samples = np.random.choice(len(corr), 4, replace=False)
        random_sample_corr = [corr[i] for i in random_samples]
        # Estimate homography using the sampled correspondences
        im1_pts = np.array([pt[0] for pt in random_sample_corr]).T
        im2_pts = np.array([pt[1] for pt in random_sample_corr]).T
        H = computeH(im1_pts, im2_pts)
        # Compute inliers using the estimated homography
        inliers = []
        for pt1, pt2 in corr:
            pt1_hom = np.append(pt1, 1)
            pt2_est_hom = np.dot(H, pt1_hom)
            pt2_est = pt2_est_hom[:2] / pt2_est_hom[2]
            if np.linalg.norm(pt2 - pt2_est) < thresh:
                inliers.append((pt1, pt2))
        # Update maximum inliers
        if len(inliers) > len(max_inliers):
            max_inliers = inliers
            H_max = H
    return H_max, max_inliers
```

```python
# corresponding keypoints
ref_pts = np.float32([keypoints2[match[0].trainIdx].pt for match in good12])
img_pts = np.float32([keypoints1[match[0].queryIdx].pt for match in good12])
# (point in image1, point in image2)
corr = list(zip(img_pts, ref_pts))
H_max, _ = ransac(corr, thresh=7, iter=750)

np.save('C:\\Users\\juver\\good12.npy', H_max)

# corresponding keypoints
ref_pts = np.float32([keypoints3[match[0].trainIdx].pt for match in good23])
img_pts = np.float32([keypoints2[match[0].queryIdx].pt for match in good23])
# (point in image1, point in image2)
corr = list(zip(img_pts, ref_pts))
H_max, _ = ransac(corr, thresh=7, iter=750)

np.save('C:\\Users\\juver\\good23.npy', H_max)

# Load the homography matrix from the file
H_max1 = np.load('C:\\Users\\juver\\good12.npy')
H_max2 = np.load('C:\\Users\\juver\\good23.npy')

warped_image1 = cv2.warpPerspective(image1, H_max1, (image2.shape[1], image2.shape[0]))
stitch2 = np.zeros((568, 720, 3), dtype=np.uint8)

stitch2[:, :420] = warped_image1[:, :420]

stitch2[:, 420:] = image2[:, 420:]
plt.imshow(cv2.cvtColor(stitch2, cv2.COLOR_BGR2RGB))
warped_image2 = cv2.warpPerspective(image3, H_max2, (image2.shape[1], image2.shape[0]))
plt.imshow(cv2.cvtColor(warped_image2, cv2.COLOR_BGR2RGB))
```

```python
stitch3 = np.zeros((568, 1010, 3), dtype=np.uint8)

stitch3[:, :610] = image2[:, :610]

stitch3[:, 610:] = warped_image2[:, :400]

plt.imshow(cv2.cvtColor(stitch3, cv2.COLOR_BGR2RGB))

mosaic = np.zeros((568, 1060, 3), dtype=np.uint8)
mosaic[:, :270] = warped_image1[:, :270]
mosaic[:, 270:610] = image2[:, 270:610]
mosaic[:, 610:] = warped_image2[:, :450]

plt.imshow(cv2.cvtColor(mosaic, cv2.COLOR_BGR2RGB))
```

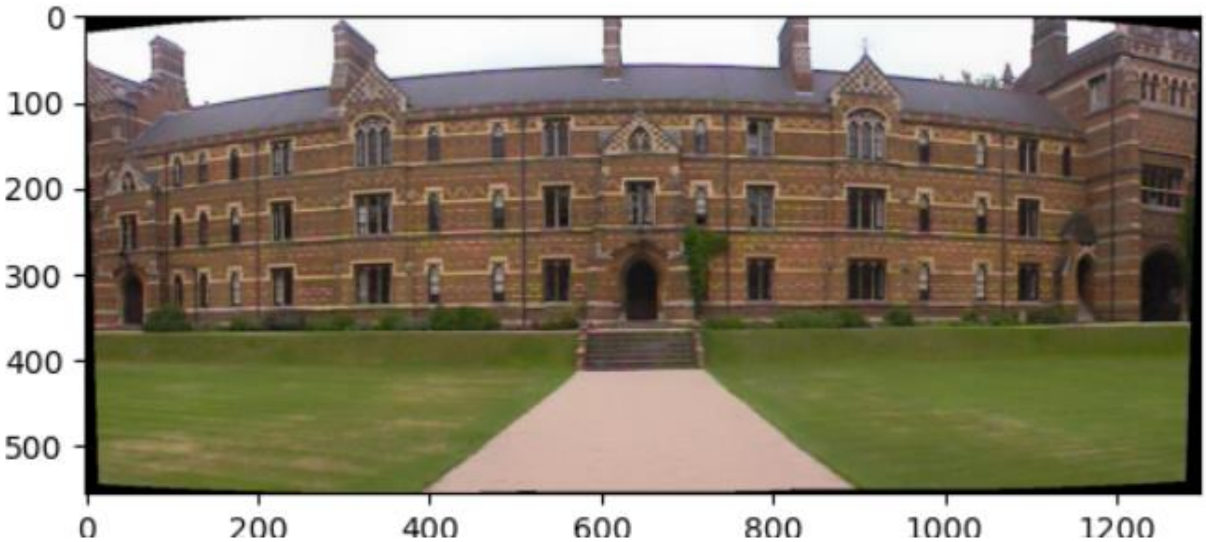
2.5--




```
# 2.5 Extra credit
import cv2
import numpy as np
image1 = cv2.imread('C:/Users/juver/Downloads/hw3/keble_a.jpg')
image2 = cv2.imread('C:/Users/juver/Downloads/hw3/keble_b.jpg')
image3 = cv2.imread('C:/Users/juver/Downloads/hw3/keble_c.jpg')
def non_black_rectangle(image):
    # grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # a binary image
    _, binary = cv2.threshold(gray, 1, 255, cv2.THRESH_BINARY)

    # contours
    contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # initial largest area and the rectangle coordinates
    max_area = 0
    best_rect = (0, 0, 0, 0)

    # best bound rectangle, update max area and its rectangle coordinates
    for contour in contours:
        x, y, w, h = cv2.boundingRect(contour)
        area = w * h
        if area > max_area:
            max_area = area
            best_rect = (x, y, w, h)

    # Crop the image to the largest found rectangle
    x, y, w, h = best_rect
    cropped_image = image[y:y+h, x:x+w]
    return cropped_image
non_black = non_black_rectangle(mosaic)
plt.imshow( cv2.cvtColor( non_black, cv2.COLOR_BGR2RGB))
```

2.6--

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Load images
image1 = cv2.imread('C:/Users/juver/Downloads/hw3/keble_a.jpg')
image2 = cv2.imread('C:/Users/juver/Downloads/hw3/keble_b.jpg')
image3 = cv2.imread('C:/Users/juver/Downloads/hw3/keble_c.jpg')

def stitch_images(images):
    # Create a Stitcher object
    stitcher = cv2.Stitcher_create()

    # Attempt to stitch the images
    status, stitched_image = stitcher.stitch(images)
    if status == cv2.Stitcher_OK:
        return stitched_image
    return None


stitched_image = stitch_images([image1, image2, image3])

plt.imshow(cv2.cvtColor(stitched_image, cv2.COLOR_BGR2RGB))
```



Technical Differences:

For my approach, I have used SIFT for keypoints and descriptors. The OpenCV Stitcher uses built-in Stitcher algorithms which can be ORB and these robusts methods are tuned to get fine results which is not the case where I implemented.

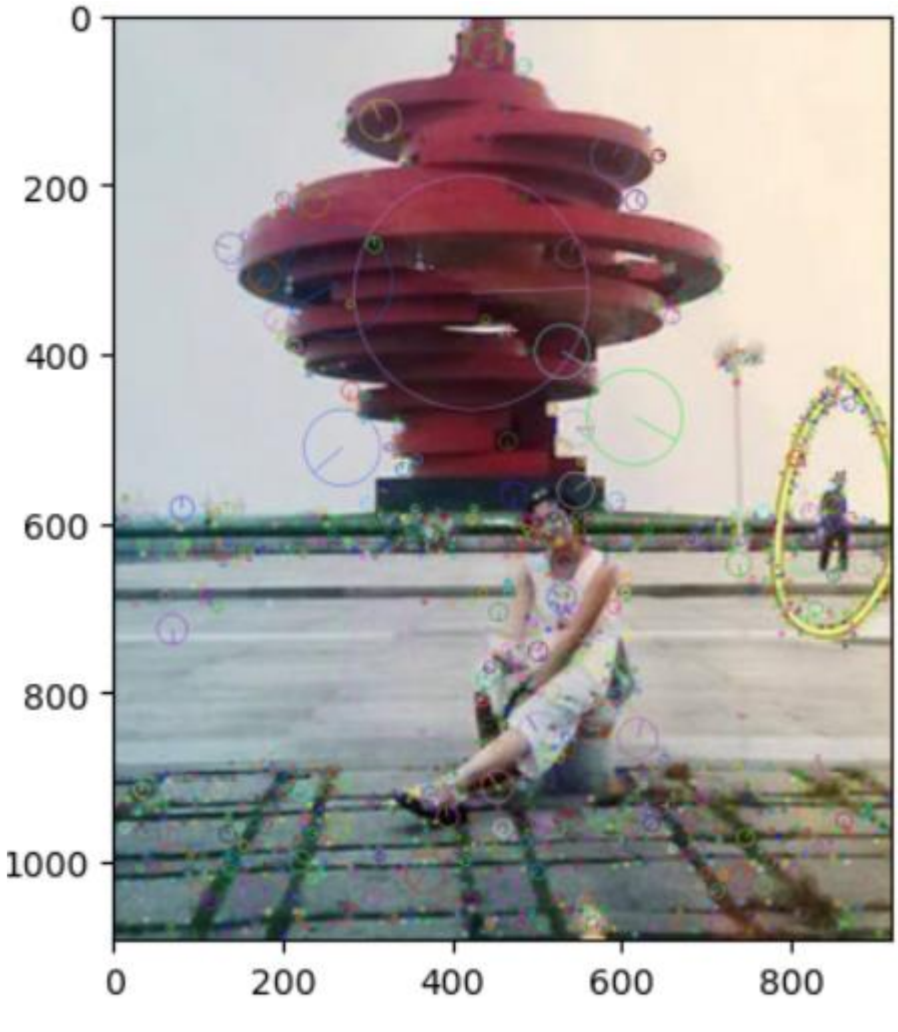
Theres a difference in using the feature matching too. I have used default BFMatcher whereas OpenCV Stitcher uses multi-band blending for smooth transitions.

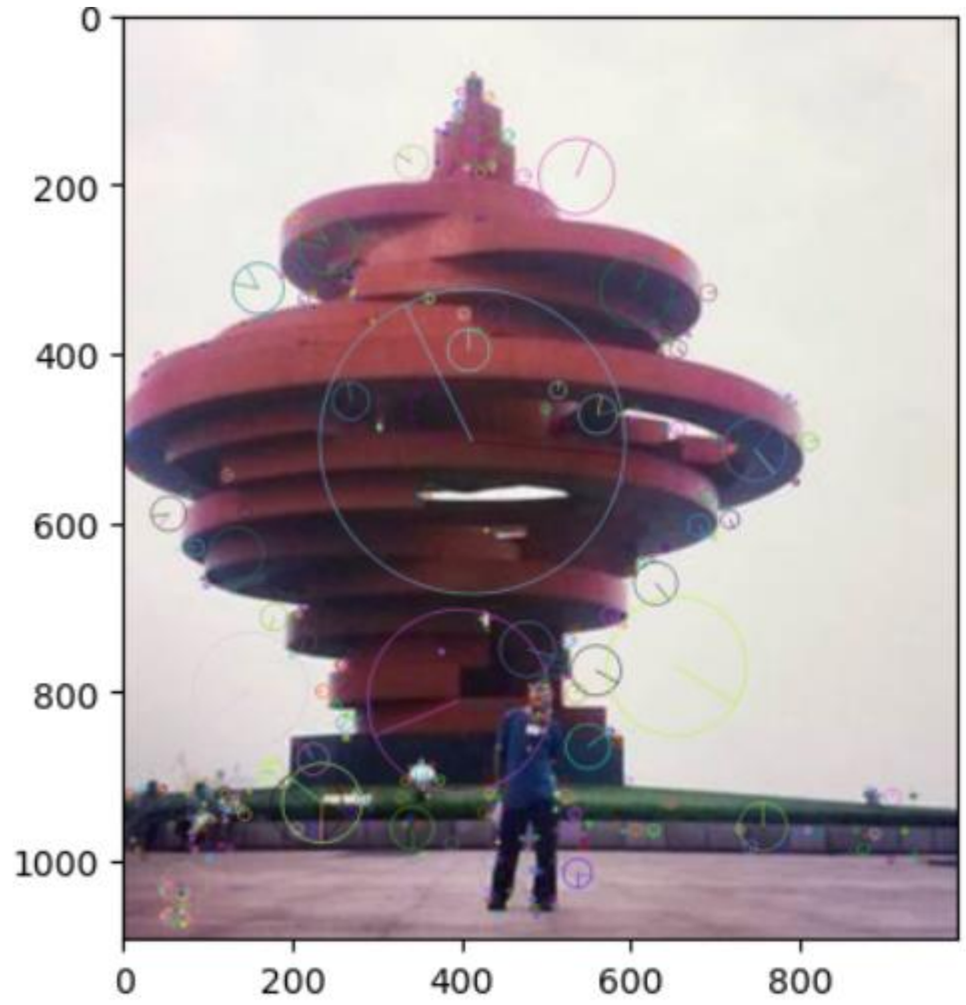
For the Homography estimation too, I have limited the iterations and threshold, the in-built optimizes and uses the parameters efficiently.

The image blending techniques are advanced in the OpenCV stitcher compared to my approach because of the usage of multi-band blending.

2.7---

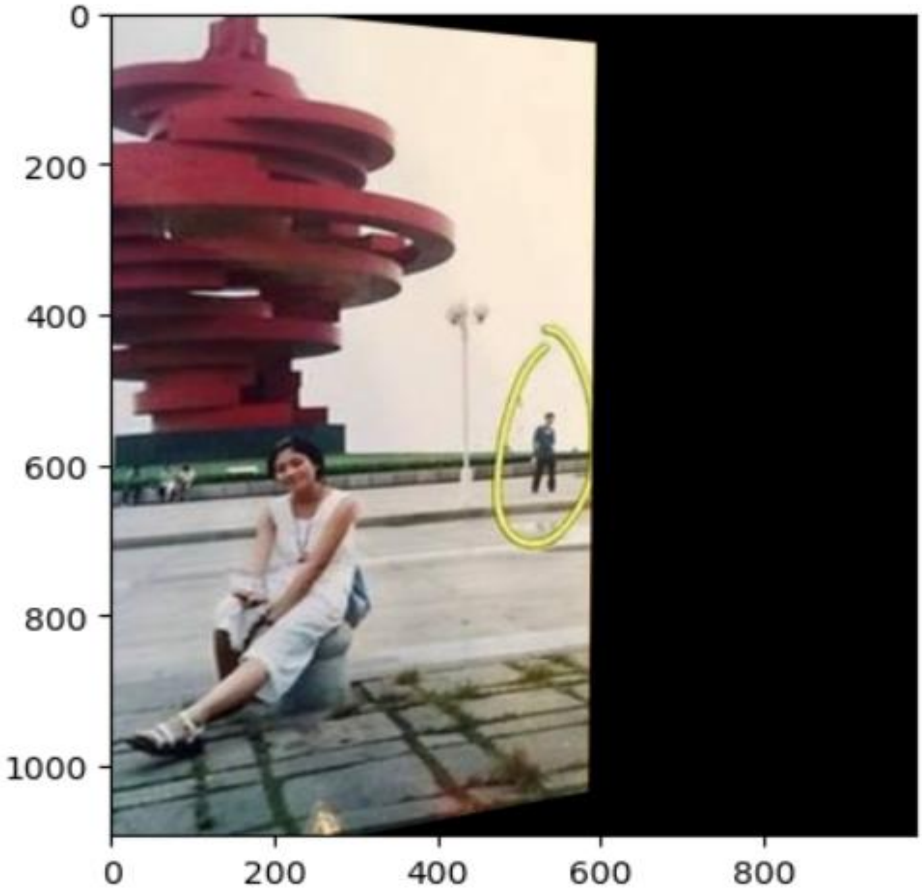
Keypoints:

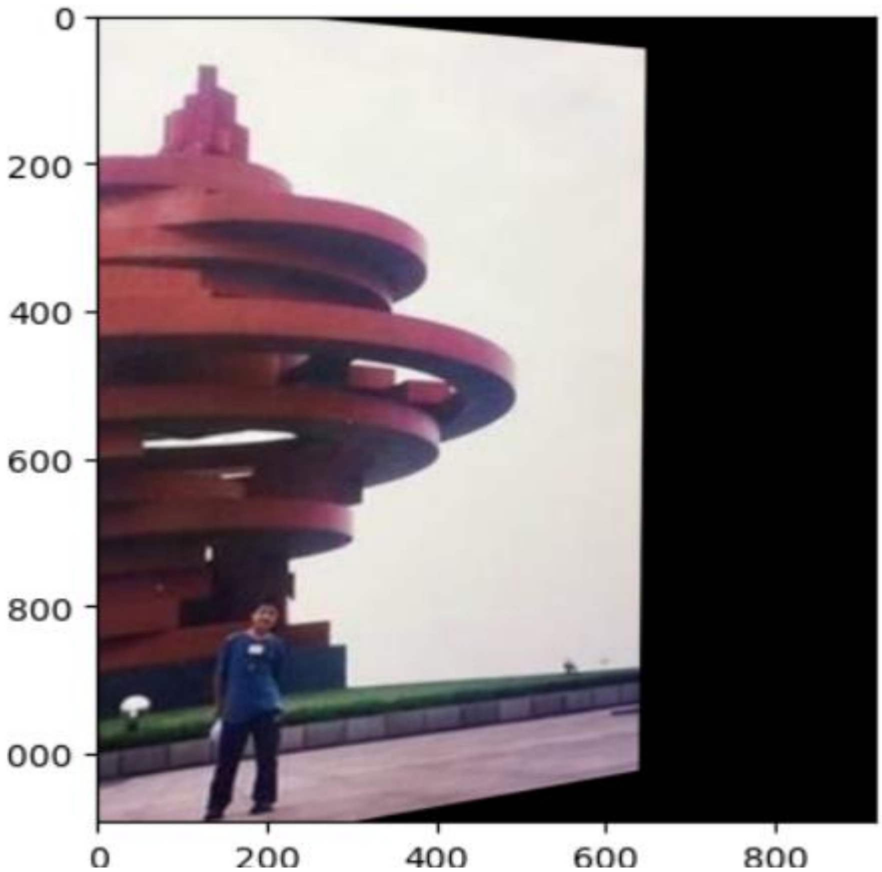




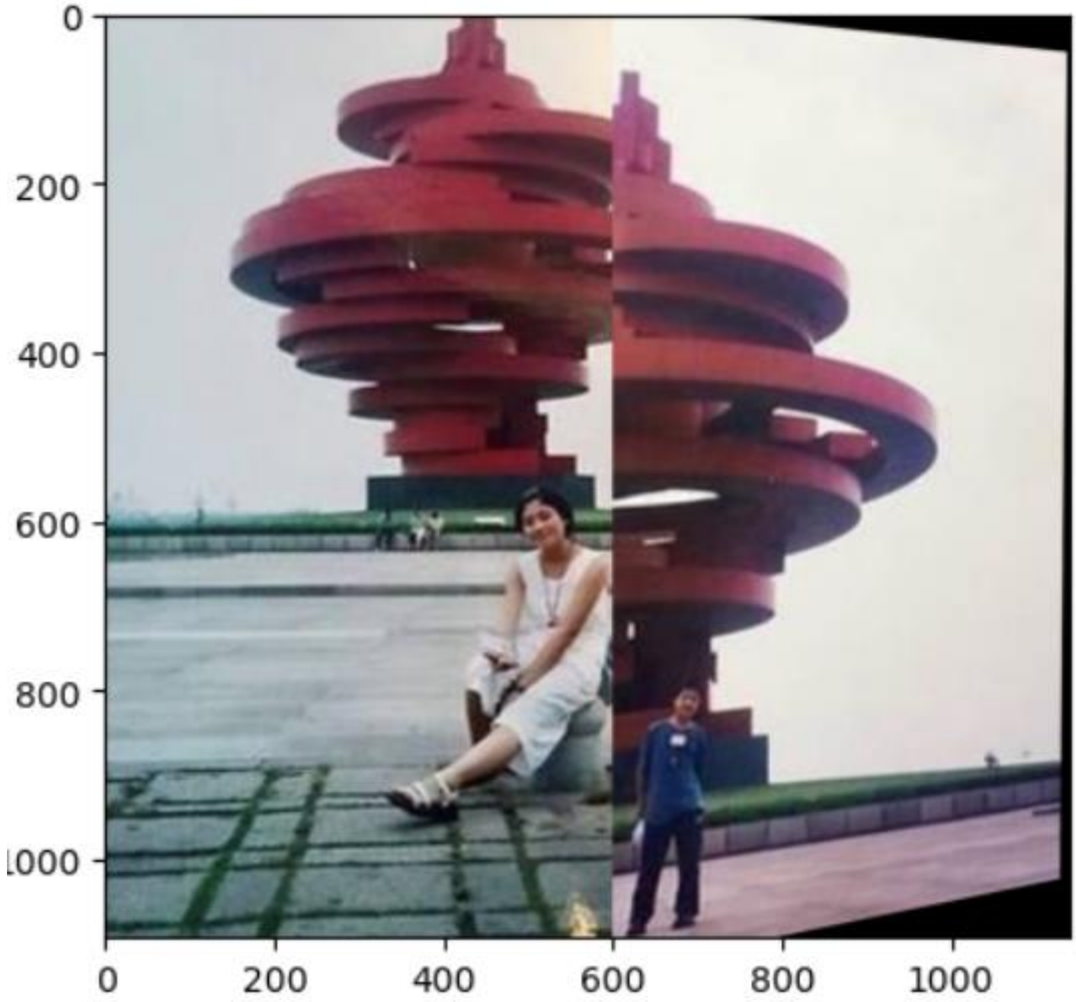
Match Features:

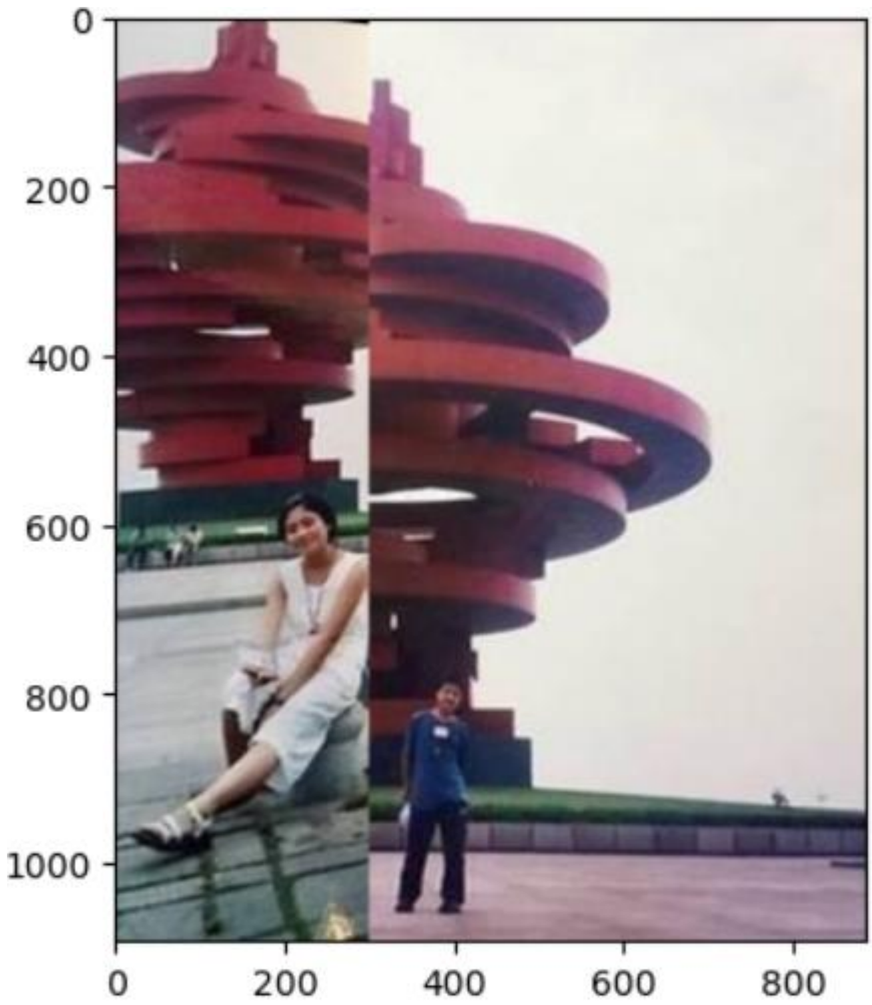
Homographies:


0
200
400
600
800
1000
0
200
400
600
800


0
200
400
600
800
000
0
200
400
600
800

Stitching:






```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Load images
image4 = cv2.imread('C:/Users/juver/Downloads/hw3/xue.png')
image5 = cv2.imread('C:/Users/juver/Downloads/hw3/ye.png')
#SIFT
sift = cv2.SIFT_create()
# Method 2: Directly finding keypoints and descriptors
keypoints11, descriptors11 = sift.detectAndCompute(image4, None)
keypoints22, descriptors22 = sift.detectAndCompute(image5, None)
img=cv2.drawKeypoints(image4,keypoints11,None,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img_rgb)

good122 = match_feature_descriptors(descriptors11, descriptors22)
img3 = cv2.drawMatchesKnn(image4,keypoints11,image5,keypoints22,good122,None,flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
img3 = cv2.cvtColor(img3, cv2.COLOR_BGR2RGB)
plt.imshow(img3),plt.show()

# Get corresponding keypoints
ref_pts = np.float32([keypoints22[match[0].trainIdx].pt for match in good122])
img_pts = np.float32([keypoints11[match[0].queryIdx].pt for match in good122])
# (point in image1, point in image2)
corr = list(zip(img_pts, ref_pts))

H_max_x, _ = ransac(corr, thresh=7, iter=750)
warped_image11 = cv2.warpPerspective(image4, H_max1, (image5.shape[1], image5.shape[0]))
plt.imshow(cv2.cvtColor(warped_image11, cv2.COLOR_BGR2RGB))
stitch5 = np.zeros((1093, 886, 3), dtype=np.uint8)
stitch5[:, :300] = warped_image11[:, :300]
stitch5[:, 300:] = image5[:, 400:]

plt.imshow(cv2.cvtColor(stitch5, cv2.COLOR_BGR2RGB))
```

References:

[1] s11263-006-0002-3.pdf (springer.com)

[2] OpenCV: Introduction to SIFT (Scale-Invariant Feature Transform)

[3] OpenCV: Feature Matching

[4] homography_estimation.pdf (ucsd.edu)

[5] OpenCV: Images stitching

[6] Matthew Brown, Richard Szeliski, and Simon Winder. Multi-image matching using multi-scale oriented patches. In 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), volume 1, pages 510–517. IEEE, 2005. 2

[7] Matplotlib — Visualization with Python

[8] Python Documentation contents — Python 3.12.2 documentation

[9] NumPy –

[10] OpenCV - Open Computer Vision Library

[11] Lecture 5: Image Features (umbc.edu)

[12]12_homographies.pdf (umbc.edu)

[13]13_camera-models.pdf (umbc.edu)