Kevin Le
CSE 803
Hw3

In [ ]:
```
import numpy as np
import cv2
from matplotlib import pyplot as plt
from common import *
from starter_file import *
```

RANSAC [30 pts]

1.1 Fitting a Line [10 pts]

In this section, each question depends on the previous one. By putative model, we mean the model that is fit in the inner loop of RANSAC

Suppose we are fitting a line (e.g., y = mx + b). How many points do we need to sample in an iteration to compute a putative model?

We need at least 2 points to compute the model since at minimum, 2 points makes a line.

In the previous setting, suppose the outlier ratio of the data set is 0.1. What is the probability that a putative model fails to get a desired model?

Given $r = 0.1$ such that it is 10% to get an outlier
N = number of trials
s = Number of points for the subset

$$Failure = (1 - (1 - r)^s)^N = (1 - 0.9^s)^N$$

Since we state 2 is the minimum, $s = 2$ thus:

$$Failure = (1 - 0.9^2)^N = (1 - 0.81)^N = 0.19^N$$

Therefore the probability to fail to get a desired model is 19% for a trial

In the previous settings, to exceed a probability level of 95% for success, how many trials should we attempt to fit putative models?

Given that $Success = 1 - Failure = 1 - 0.19^N$
We need to have a probability level of >0.95 for a success.

$$\begin{gathered} 0.95 < 1 - 0.19^N \\ -0.05 < -0.19^N \\ log_{-0.19}(-0.05) < N \\ \sim 1.8 < N \end{gathered}$$

Thus we need at least 2 trials to exceed the probability level of 95%

1.2 Fitting Transformations [20 pts]

We'll begin by reviewing fitting transformations from 2D points to 2D points. You will need to use these results to solve the subsequent sections

Recall that a matrix $M \in \mathbb{R}^{2\times 2}$ is a linear transformation: $\mathbb{R}^2 \to \mathbb{R}^2$. How many degrees of freedom does M have? How many samples are required to find M?

Since we have a 2x2 matrix, that means we have 4 elements that are changeable, thus there are 4 degrees of freedom in M. Each row in the M matrix is a sample, thus with a 2x2 matrix, we have 2 samples for M.

Suppose we have a dataset $(x_i, y_i)_{i=1}^N$ and want to fit $y = Mx$. Formulate the fitting problem into the form of a least squares problem:

$$argmin_m ||Am - b||^2$$

where m is some vector that has all the parameters of M. Write out the form of A.

Given:

$$y = Mx \to \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Since $(x_i, y_i)_{i=1}^N$:

$$A^{2N\times 4} * m^{4\times 1} = b^{2N\times 1}$$

where m is the vertical collapse of M. 2N due to the 2 points for a sample. Thus:

$$\begin{pmatrix} y_1^1 \\ y_2^1 \\ y_1^2 \\ y_2^2 \\ \dots \\ y_1^N \\ y_2^N \end{pmatrix} = \begin{pmatrix} x_1^1 & x_2^1 & 0 & 0 \\ 0 & 0 & x_1^1 & x_2^1 \\ \dots & \dots & \dots & \dots \\ x_1^N & x_2^N & 0 & 0 \\ 0 & 0 & x_1^N & x_2^N \end{pmatrix} \begin{pmatrix} M_{11} \\ M_{12} \\ M_{21} \\ M_{22} \end{pmatrix}$$

Therefore:

$$A = \begin{pmatrix} x_1^1 & x_2^1 & 0 & 0 \\ 0 & 0 & x_1^1 & x_2^1 \\ \dots & \dots & \dots & \dots \\ x_1^N & x_2^N & 0 & 0 \\ 0 & 0 & x_1^N & x_2^N \end{pmatrix}$$

Use numpy.load() to load p1/transform.npy. Each row contains two points x and y, represented in the form $[x_i^T, y_i^T]_{1\times 4}$. Then fit a transformation:

$$y = Sx + t$$

where $S \in \mathbb{R}^{2\times2}, t \in \mathbb{R}^{2\times1}$. Solve the problem by setting up an optimization of the form of $argmin_v||Am - b||^2$. Report S and t.

Given:

$$y = Sx + t \rightarrow \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}$$

Since m in ||Am-b|| is our unknown, collapse S and t into a vertical vector

$$m = \begin{pmatrix} S_{11} \\ S_{12} \\ S_{21} \\ S_{22} \\ t_1 \\ t_2 \end{pmatrix}$$

Thus:

$$\begin{pmatrix} y_1^1 \\ y_2^1 \\ y_1^2 \\ y_2^2 \\ \dots \\ y_1^N \\ y_2^N \end{pmatrix} = \begin{pmatrix} x_1^1 & x_2^1 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_1^1 & x_2^1 & 0 & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ x_1^N & x_2^N & 0 & 0 & 1 & 0 \\ 0 & 0 & x_1^N & x_2^N & 0 & 1 \end{pmatrix} \begin{pmatrix} S_{11} \\ S_{12} \\ S_{21} \\ S_{22} \\ t_1 \\ t_2 \end{pmatrix}$$

Given: $m = (A^T A)^{-1} A^T b$

In [ ]:
```
data_set = np.load("p1/transform.npy")
A = np.zeros((2* data_set.shape[0], 6))
b = np.zeros((2* data_set.shape[0], 1))
x = np.zeros((2* data_set.shape[0], 1))
for point in data_set:
    A = np.vstack((A, np.array([point[0], point[1], 0, 0, 1, 0])))
    A = np.vstack((A, np.array([0, 0, point[0], point[1], 0, 1])))

    b = np.vstack((b, point[2]))
    b = np.vstack((b, point[3]))

    x = np.vstack((x, point[0]))
    x = np.vstack((x, point[1]))

m = np.linalg.inv(A.T @ A) @ A.T @ b
s = m[0:4].reshape(2,2)
t = m[4:]
print(s)
print(t)
```

```
[[ 2.04098127 -1.01644528]
 [-3.0696843   0.94335775]]
[[-1.87154926]
 [-3.05145812]]
```

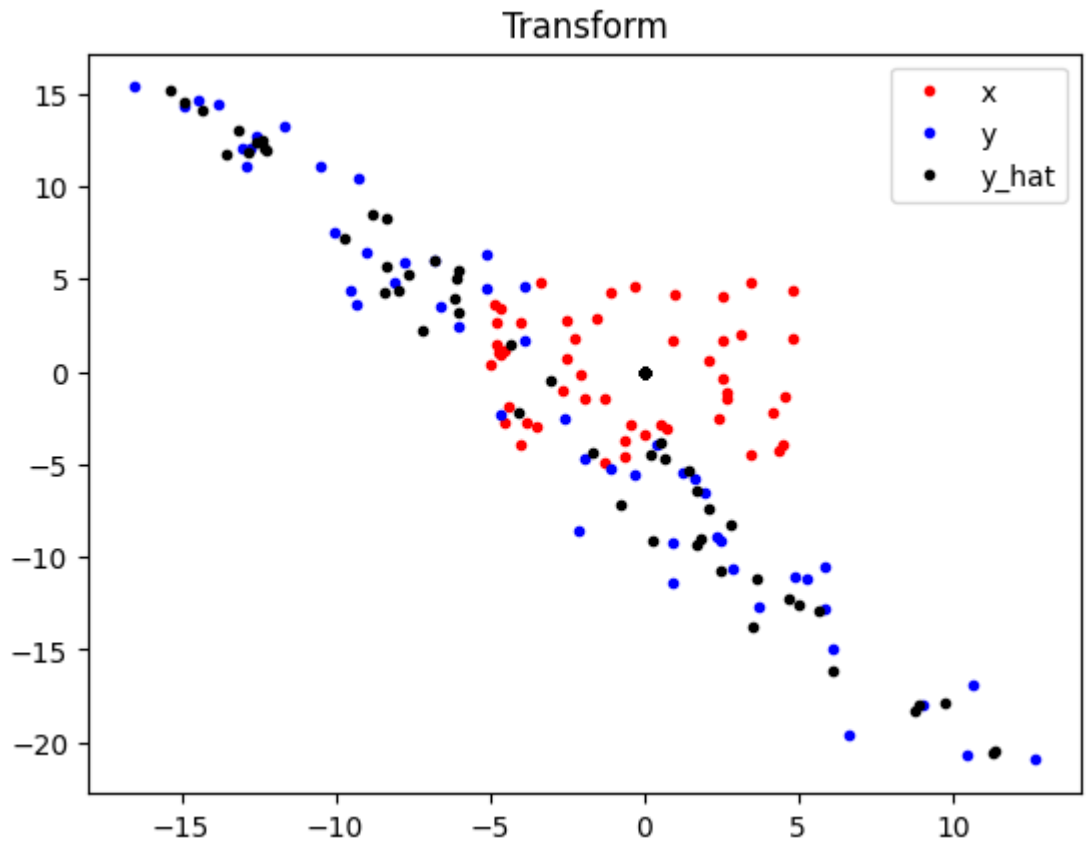
Thus:

$$S = \begin{pmatrix} 2.0498127 & -1.01644528 \\ -3.0696843 & 0.94335775 \end{pmatrix}$$

$$t = \begin{pmatrix} -1.87154926 \\ -3.05145812 \end{pmatrix}$$

Plot the points x, y and $\hat{y} = Sx + t$ in one figure with different colors. Display the figure

In [ ]:
```
y_hat = A @ m
x_plot = x.reshape(-1, 2)
y_hat_plot = y_hat.reshape(-1, 2)
y_plot = b.reshape(-1, 2)

plt.plot(x_plot[:, 0], x_plot[:, 1], 'r.', \
    y_plot[:, 0], y_plot[:, 1], 'b.', \
    y_hat_plot[:, 0], y_hat_plot[:, 1], 'k.')
plt.title('Transform')
plt.legend(['x', 'y', 'y_hat'])
plt.show()
```



Explain how you transform the points in words based on S and t. Is the fitting result good?

S and t takes the values from $X \in [-5, 5]$ and maps it to $\approx [-20, 15]$ "y_hat" is a good fit since there is a close corrolation to y shown in the plot

We have 8 cases of homography transformation of letter M. In each case, there are two sets of 2D points X and Y, which are represented in the same format as 1.2.3 (a N×4 array with each row in the form $[x_i^T, y_i^T]_{1\times 4}$, we have loaded the data for you). For each case, fit a homography

$$\begin{bmatrix} y \\ 1 \end{bmatrix} \equiv H \begin{bmatrix} x \\ 1 \end{bmatrix}$$

where $H \in \mathbb{R}^{3\times 3}$. Solve the problem by dealing with the optimization of the form of $argmin_h ||Ah||^2$ with $||h|| = 1$ where h has all the parameters of H. Report matrix H of each case.

Given:

$$y = Hx \rightarrow \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} \equiv \begin{bmatrix} x_2' \\ y_2' \\ z_2' \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Note from Szeliski:

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31} + h_{32} + h_{33}} \rightarrow x'(h_{31} + h_{32} + h_{33}) = h_{11}x + h_{12}y + h_{13}$$

$$y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31} + h_{32} + h_{33}} \rightarrow y'(h_{31} + h_{32} + h_{33}) = h_{21}x + h_{22}y + h_{23}$$

Collapse and create a vertical vector for H:

$$h = \begin{bmatrix} H_{11} \\ H_{12} \\ H_{13} \\ H_{21} \\ H_{22} \\ H_{23} \\ H_{31} \\ H_{32} \\ H_{33} \end{bmatrix}$$

Thus, with $N \times 4$ arrays,

$$A = \begin{bmatrix} 0 & 0 & 0 & -X_1^1 & -X_2^1 & -X_3^1 & (Y_2^1 * X_1^1) & (Y_2^1 * X_2^1) & (Y_2^1 * X_3^1) \\ X_1^1 & X_2^1 & X_3^1 & 0 & 0 & 0 & (-Y_1^1 * X_1^1) & (-Y_1^1 * X_2^1) & (-Y_1^1 * X_3^1) \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & -X_1^N & -X_2^N & -X_3^N & (Y_2^N * X_1^N) & (Y_2^N * X_2^N) & (Y_2^N * X_3^N) \\ X_1^N & X_2^N & X_3^N & 0 & 0 & 0 & (-Y_1^N * X_1^N) & (-Y_1^N * X_2^N) & (-Y_1^N * X_3^N) \end{bmatrix}_{2N\times 9}$$

Getting eigen vector for $A^T A$ with the smallest eigen value gets H

In [ ]:
```
def homography_transform(X, H):
    # Perform homography transformation on a set of points X
    # using homography matrix H
    # Input - a set of 2D points in an array with size (N,2)
    #         a 3*3 homography matrix
    # Output - a set of 2D points in an array with size (N,2)
    X = np.hstack((X, np.ones((X.shape[0],1))))
    Y = X @ H.T
    Y /= Y[:,2:3]
    return Y[:,:2]


def fit_homography(XY):
    # Given two set of points X, Y in one array,
    # fit a homography matrix from X to Y
    # Input - an array with size(N,4), each row contains two
    #         points in the form[x^T_i,y^T_i]1×4
    # Output - a 3*3 homography matrix
    X = XY[:,0:2]
    Y = XY[:,2:4]
    rows = XY.shape[0]
    o = np.ones((rows,1))
    X = np.hstack((X, o))
    A = np.zeros((rows*2, 9))
    for i in range(rows):
        A = np.vstack((A, np.array([0,0,0, \
            -X[i][0], -X[i][1], -X[i][2], \
            Y[i][1] * X[i][0], Y[i][1] * X[i][1], Y[i][1] * X[i][2]])))
        A = np.vstack((A, np.array([X[i][0], X[i][1], X[i][2], \
            0,0,0, \
            -Y[i][0] * X[i][0], -Y[i][0] * X[i][1], -Y[i][0] * X[i][2]])))
    val, vec = np.linalg.eig(A.T @ A)
    H = vec[:, np.argmin(val)].reshape((3, 3))
    return H
```

$$H0 = \begin{pmatrix} -5.84619486e^{-01} & -9.38188545e^{-04} & 7.85709945e^{-02} \\ -1.48861804e^{-03} & -3.61934740e^{-01} & 4.27826653e^{-01} \\ -2.62615121e^{-05} & -2.09196954e^{-05} & -5.81387269e^{-01} \end{pmatrix}$$

$$H1 = \begin{pmatrix} 3.68911272e^{-01} & 8.92258702e^{-04} & -3.93043451e^{-01} \\ 2.75059829e^{-05} & 5.96020476e^{-01} & -2.41643576e^{-02} \\ 2.17658960e^{-07} & 1.34347784e^{-05} & 5.94639553e^{-01} \end{pmatrix}$$

$$H2 = \begin{pmatrix} -2.53193239e^{-01} & -3.99057800e^{-03} & 5.69038181e^{-01} \\ -5.45228988e^{-03} & -2.53238879e^{-01} & 6.28011972e^{-01} \\ -1.21872179e^{-04} & -1.08513265e^{-04} & -3.91808736e^{-01} \end{pmatrix}$$

$$H3 = \begin{pmatrix} 3.09846350e^{-12} & 5.77350269e^{-01} & -2.27213490e^{-10} \\ 5.77350269e^{-01} & 9.18668681e^{-13} & -7.49060732e^{-11} \\ 3.81114701e^{-14} & 2.27553847e^{-14} & 5.77350269e^{-01} \end{pmatrix}$$

$$H4 = \begin{pmatrix} -7.91063985e^{-03} & 2.28491079e^{-13} & 6.01208628e^{-01} \\ -5.35800683e^{-14} & -7.91063985e^{-03} & 7.98974625e^{-01} \\ 2.89392053e^{-15} & -1.06881468e^{-15} & 7.91063985e^{-03} \end{pmatrix}$$

$$H5 = \begin{pmatrix} -1.38048111e^{-01} & -2.19005570e^{-01} & 6.17163607e^{-01} \\ -2.20297434e^{-01} & -1.36451811e^{-01} & 6.05518007e^{-01} \\ -7.60197606e^{-05} & -4.81267787e^{-05} & -3.43927668e^{-01} \end{pmatrix}$$

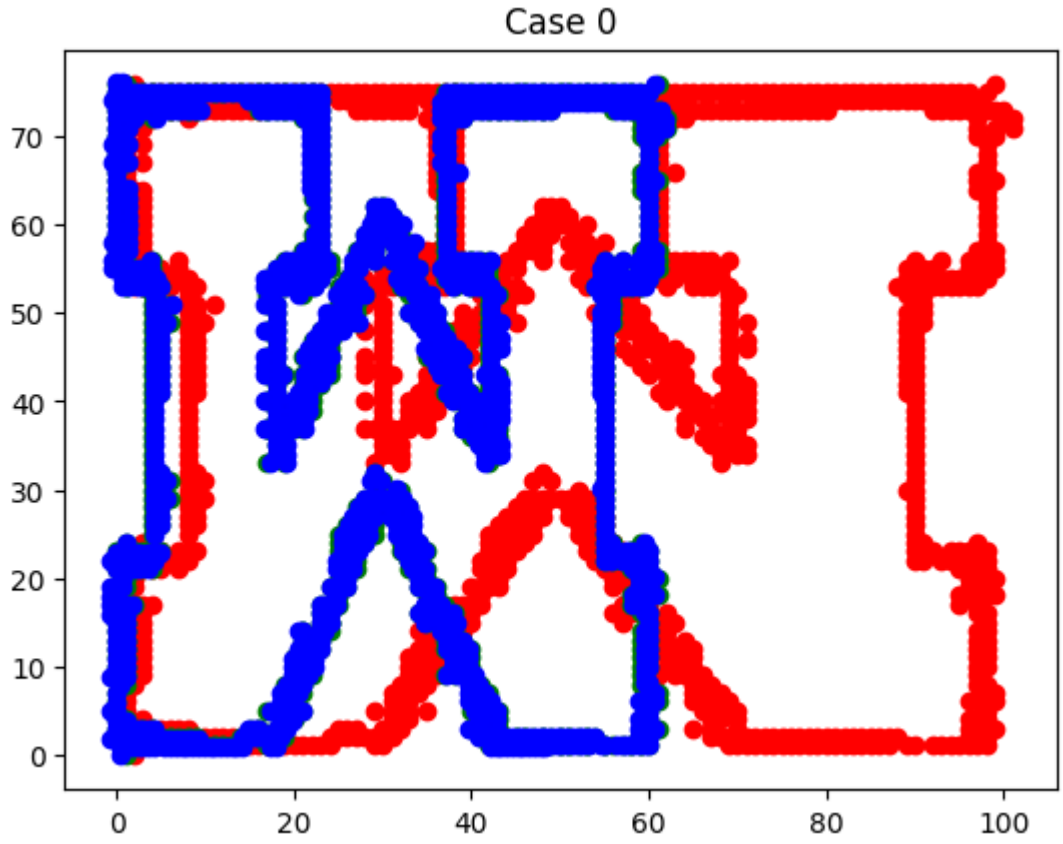
$$H6 = \begin{pmatrix} 1.34941272e^{-01} & 7.96016480e^{-03} & -1.11241055e^{-02} \\ -1.37954817e^{-02} & 1.51370758e^{-01} & 9.63220032e^{-01} \\ -5.34941898e^{-05} & -1.11646326e^{-05} & 1.75235524e^{-01} \end{pmatrix}$$

$$H7 = \begin{pmatrix} -1.44485744e^{-01} & 8.35368093e^{-03} & -7.39389853e^{-01} \\ 8.19996491e^{-03} & -1.69535496e^{-01} & -6.11590373e^{-01} \\ 7.82502801e^{-05} & -1.85804532e^{-05} & -1.71768495e^{-01} \end{pmatrix}$$

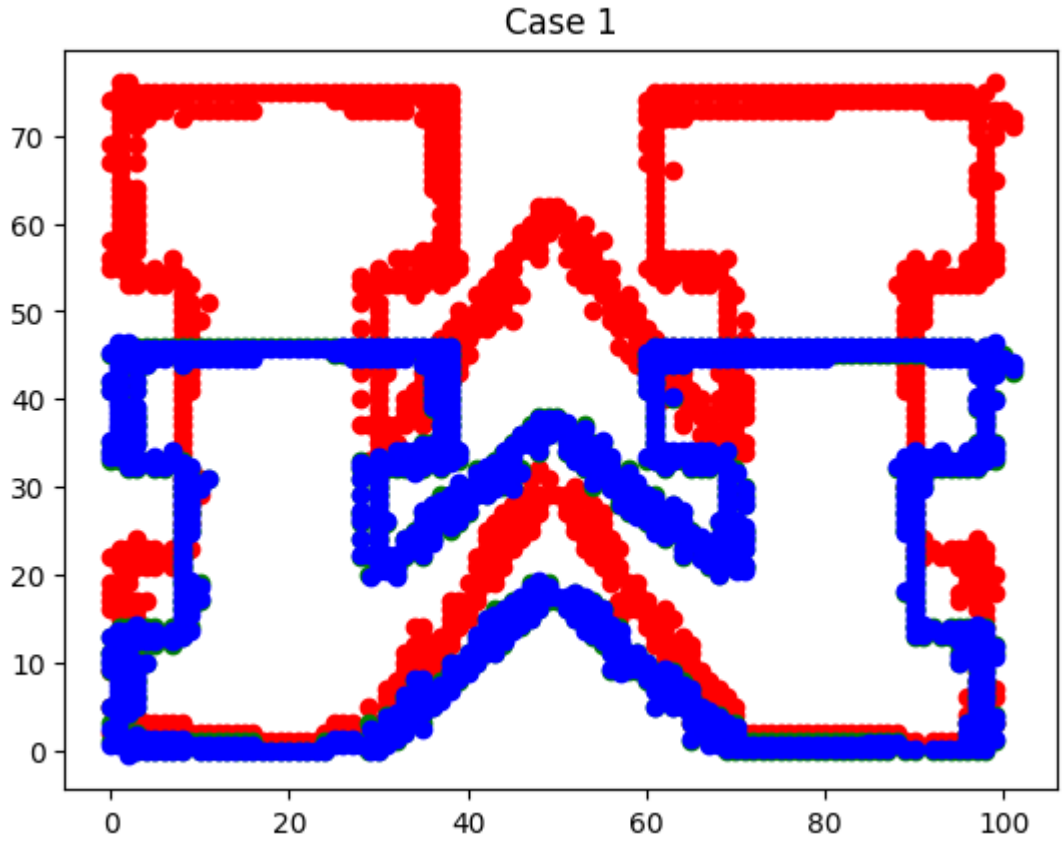
Visualize the original points, the target points and the points after homography transformation in one figure (three separate images in one figure). Display the figure of each case.

In [ ]:
```
case = 8# you will encounter 8 different transformations
for i in range(case):
    XY = np.load('p1/points_case_'+str(i)+'.npy')
    # 1. generate your Homography matrix H using X and Y
    #
    #    specifically: fill function fit_homography()
    #    such that H = fit_homography(XY)
    H = fit_homography(XY)
    # 2. Report H in your report
    print(H)
    # 3. Transform the points using H
    #
    #    specifically: fill function homography_transform
    #    such that Y_H = homography_transform(X, H)
    Y_H = homography_transform(XY[:,:2], H)
    # 4. Visualize points as three images in one figure
    # the following codes plot figure for you
    plt.scatter(XY[:,1],XY[:,0],c="red") #X
    plt.scatter(XY[:,3],XY[:,2],c="green") #Y
    plt.scatter(Y_H[:,1],Y_H[:,0],c="blue") #Y_hat
    plt.title(f'Case {i}')
    plt.show()
```

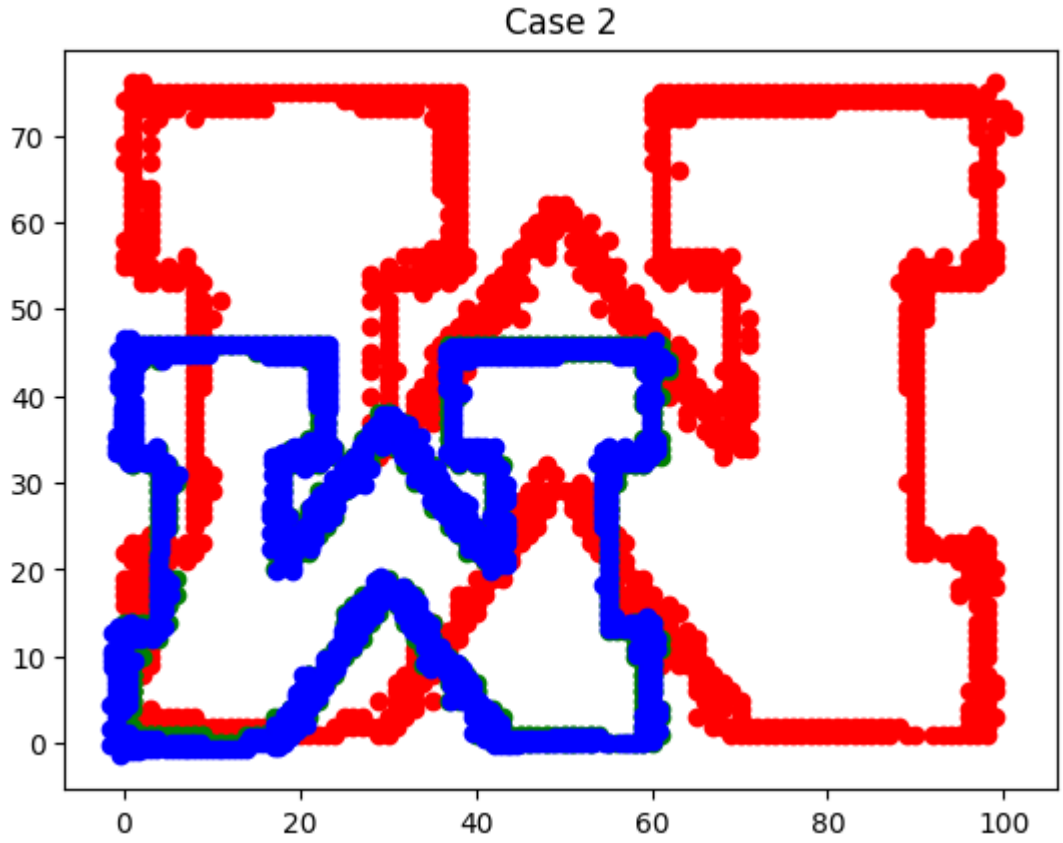
```
[[-5.84619486e-01 -9.38188545e-04  7.85709945e-02]
 [-1.48861804e-03 -3.61934740e-01  4.27826653e-01]
 [-2.62615121e-05 -2.09196954e-05 -5.81387269e-01]]
```



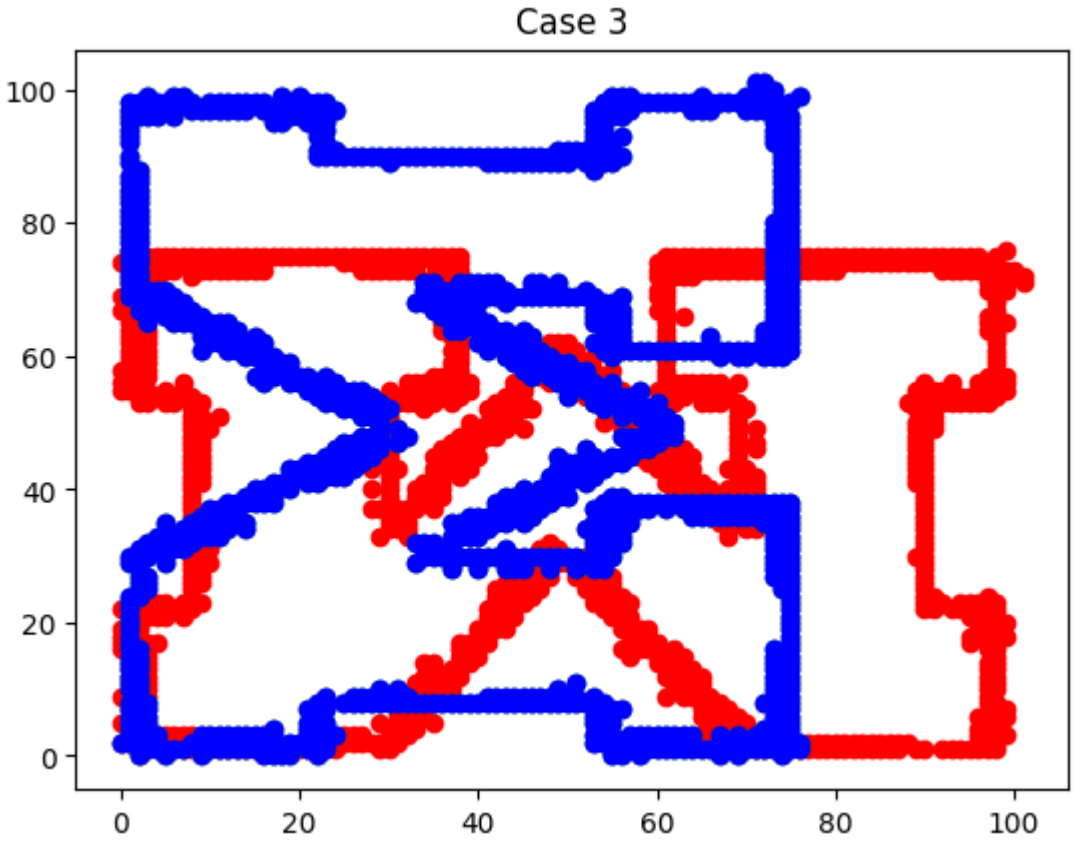

```
[[ 3.68911272e-01  8.92258702e-04 -3.93043451e-01]
 [ 2.75059829e-05  5.96020476e-01 -2.41643576e-02]
 [ 2.17658960e-07  1.34347784e-05  5.94639553e-01]]
```



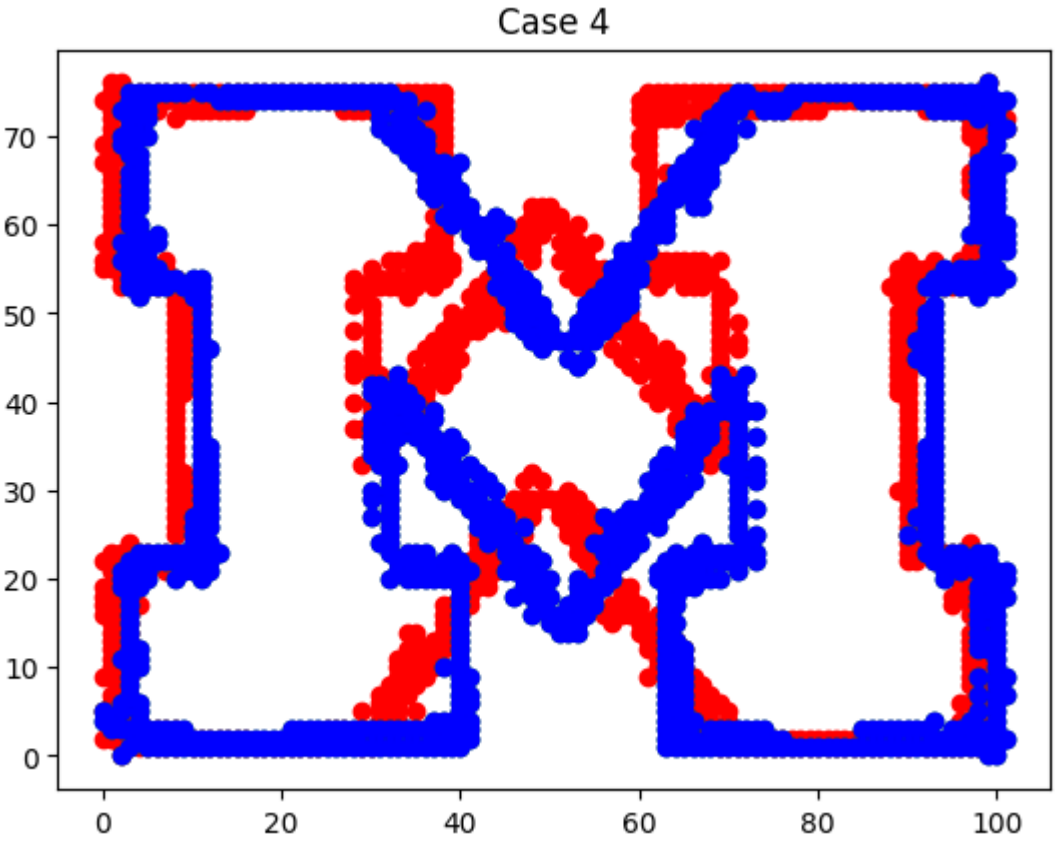
```
[[-2.53193239e-01 -3.99057800e-03  5.69038181e-01]
 [-5.45228988e-03 -2.53238879e-01  6.28011972e-01]
 [-1.21872179e-04 -1.08513265e-04 -3.91808736e-01]]
```



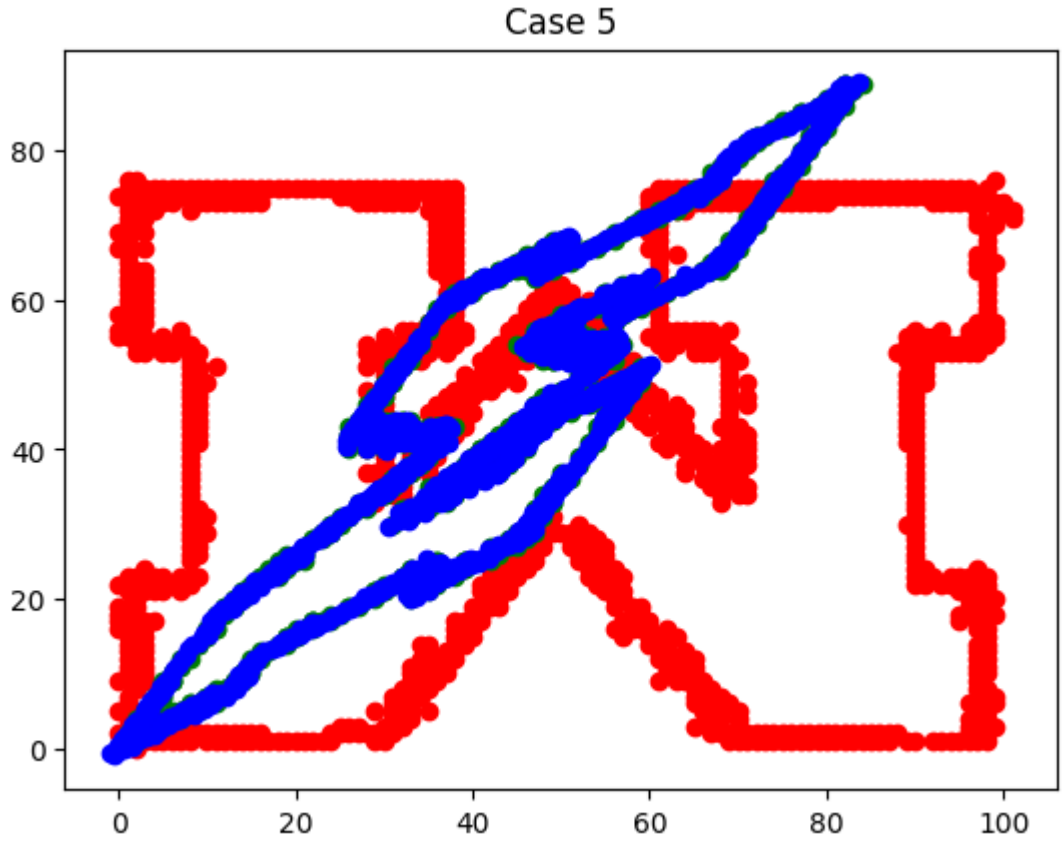

```
[[ 3.09846350e-12  5.77350269e-01 -2.27213490e-10]
 [ 5.77350269e-01  9.18668681e-13 -7.49060732e-11]
 [ 3.81114701e-14  2.27553847e-14  5.77350269e-01]]
```



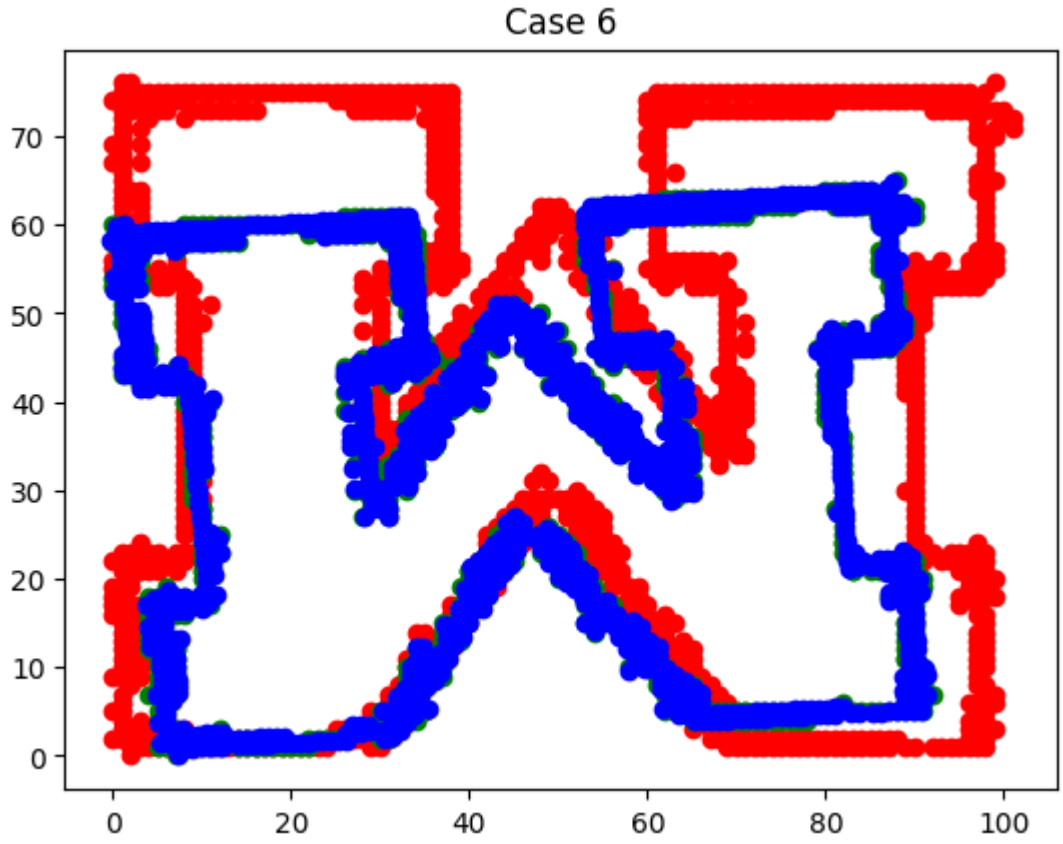

```
[[-7.91063985e-03  2.28491079e-13  6.01208628e-01]
 [-5.35800683e-14 -7.91063985e-03  7.98974625e-01]
 [ 2.89392053e-15 -1.06881468e-15  7.91063985e-03]]
```



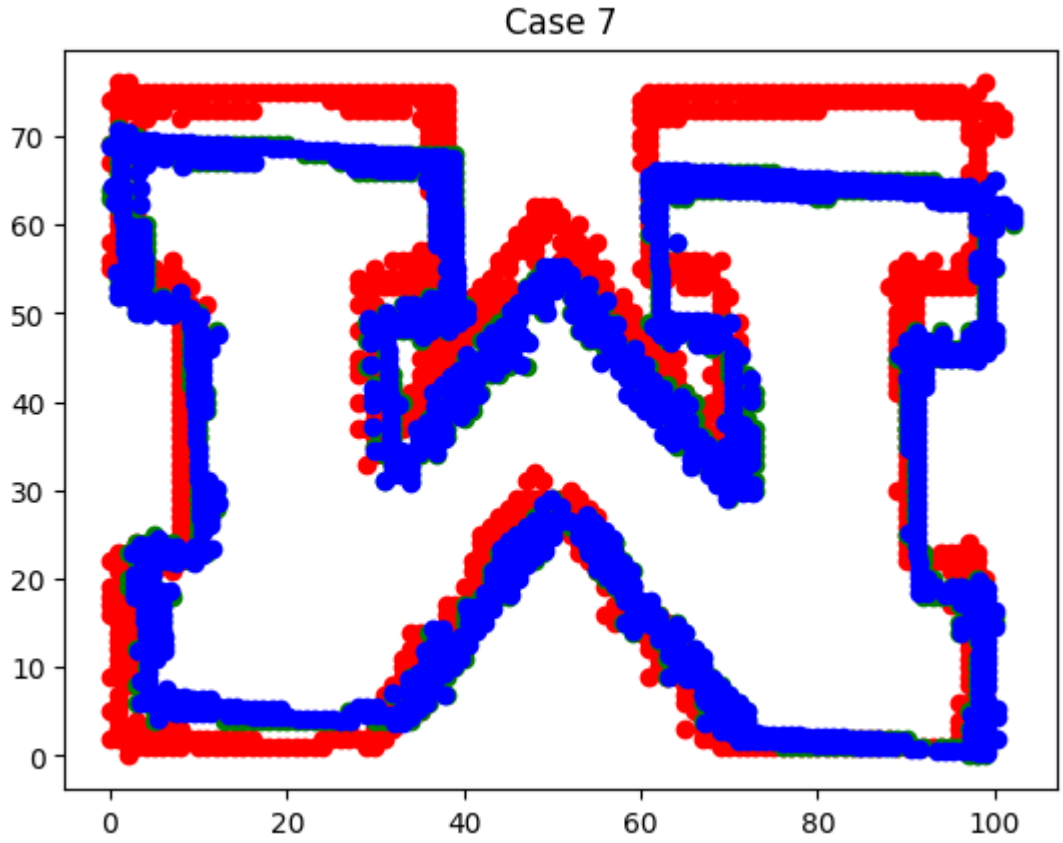
```
[[-1.38048111e-01 -2.19005570e-01  6.17163607e-01]
 [-2.20297434e-01 -1.36451811e-01  6.05518007e-01]
 [-7.60197606e-05 -4.81267787e-05 -3.43927668e-01]]
```




```
[[ 1.34941272e-01  7.96016480e-03 -1.11241055e-02]
 [-1.37954817e-02  1.51370758e-01  9.63220032e-01]
 [-5.34941898e-05 -1.11646326e-05  1.75235524e-01]]
```




```
[[-1.44485744e-01  8.35368093e-03 -7.39389853e-01]
 [ 8.19996491e-03 -1.69535496e-01 -6.11590373e-01]
 [ 7.82502801e-05 -1.85804532e-05 -1.71768495e-01]]
```



Discuss the transformations your observed in visualization with the homography matrix H. Do they make sense to you?

The transformations are accurate. The blue scatter plot represents the results of the homography transformation, and as shown, is covering the green scatter plot which is the target.

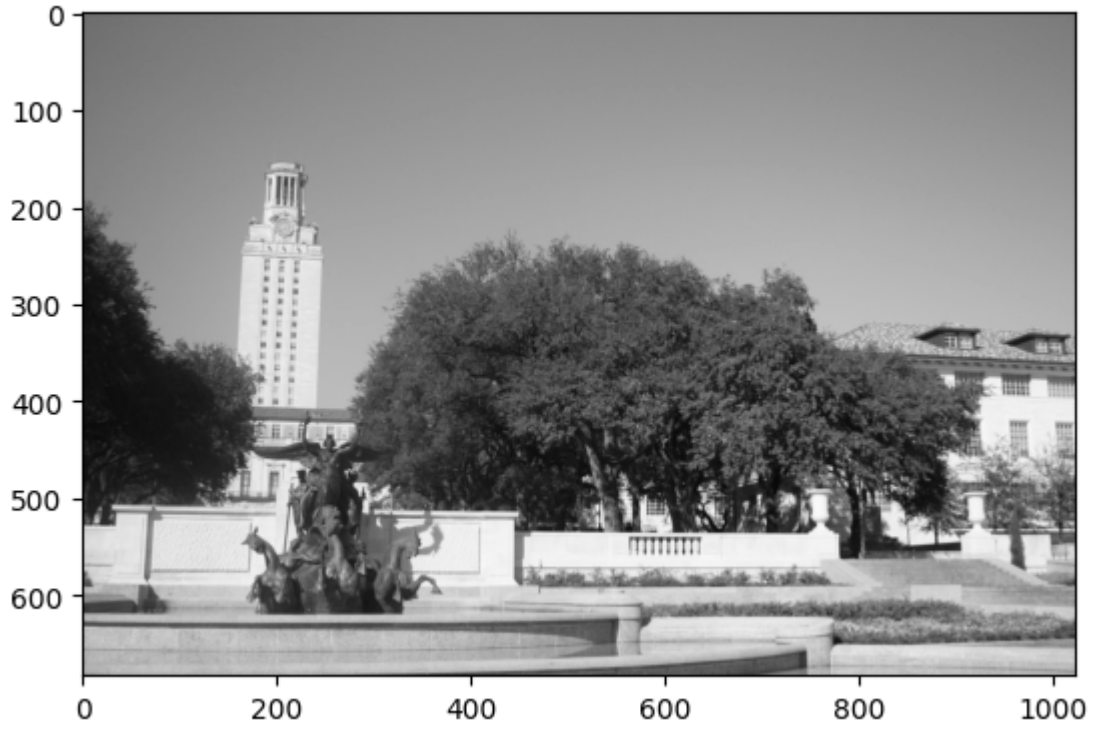
Image Stitching [55 pts]

Image stitching or photo stitching combines multiple photographic images that have overlapping fields of view to produce a segmented panorama or high-resolution image. You'll be able to do this by the end of this problem (which is derived from an assignment developed by Svetlana Lazebnik at UIUC).

Load both images: p2/uttower_left.jpg and p2/uttower_right.jpg. Convert them to double and to grayscale. Display the grayscale images.

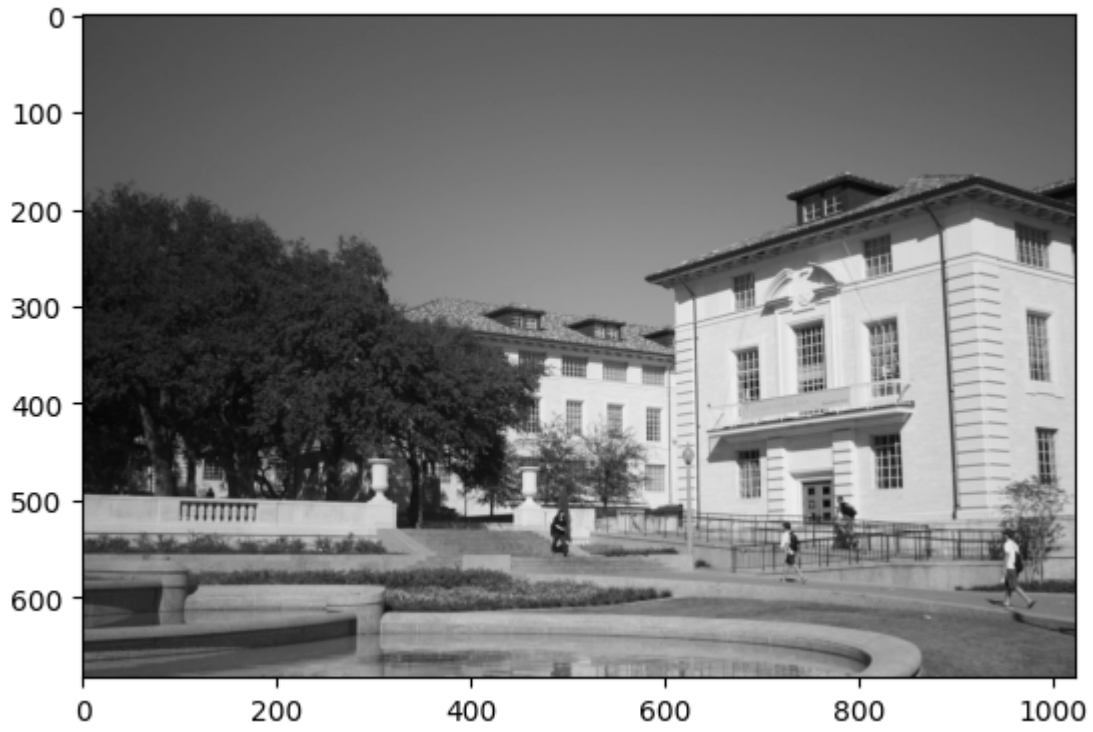
In [ ]:
```
ut_L = read_img("p2/uttower_left.jpg")
plt.imshow(ut_L, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x24e5022d748>
```



In [ ]:
```
ut_R = read_img("p2/uttower_right.jpg")
plt.imshow(ut_R, cmap='gray')
```

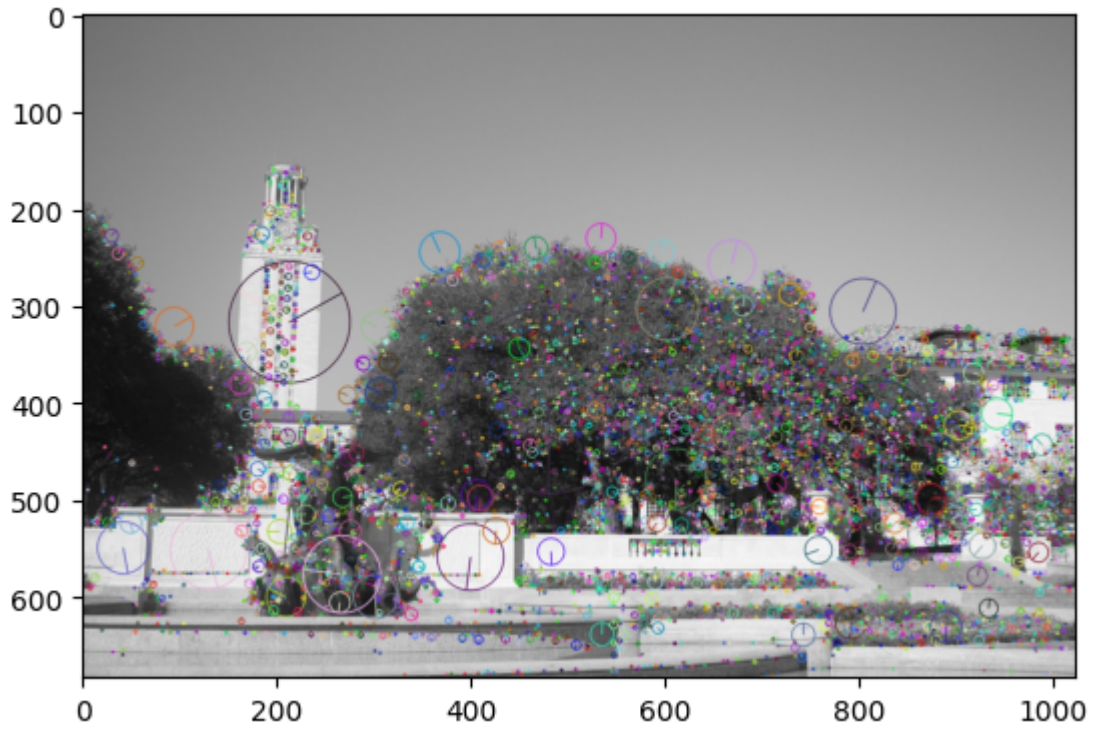
Out[ ]:
```
<matplotlib.image.AxesImage at 0x24e502b7e48>
```



Use SIFT/SURF descriptors in OpenCV to detect feature points in both images. Display both the images along with the feature points

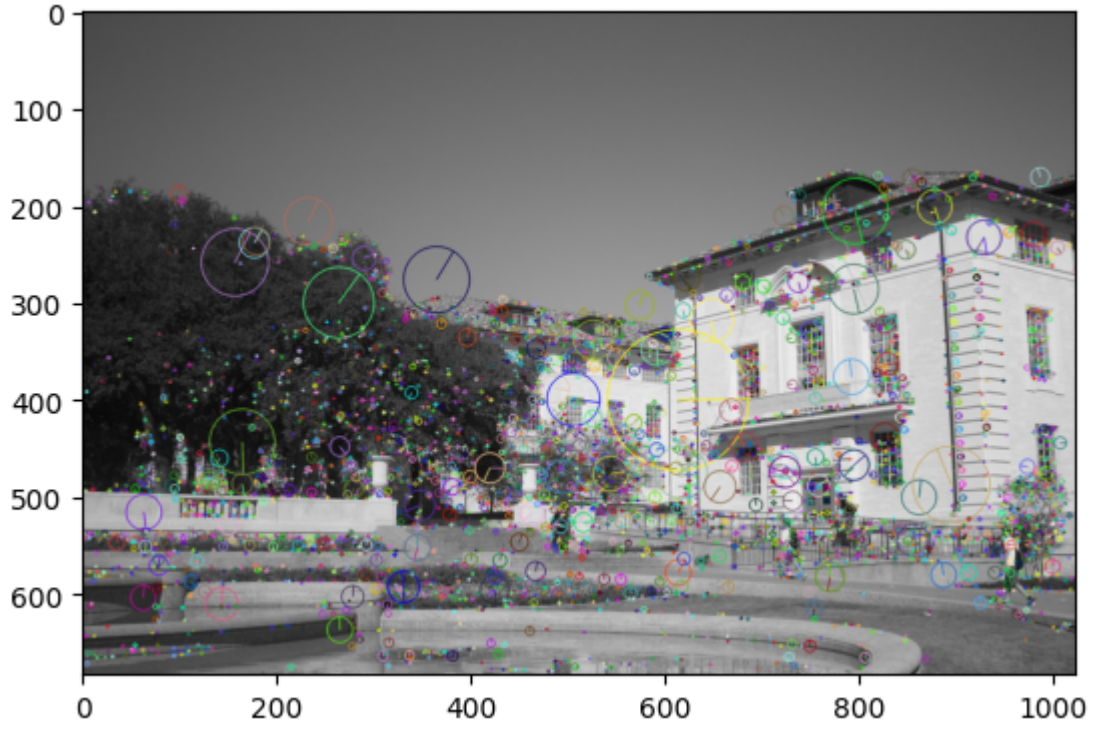
In [ ]:
```
sift = cv2.xfeatures2d.SIFT_create()

ut_L = cv2.imread("p2/uttower_left.jpg")
gray_L = cv2.cvtColor(ut_L,cv2.COLOR_BGR2GRAY)
kp_L, ds_L= sift.detectAndCompute(gray_L, None)

ut_R = cv2.imread("p2/uttower_right.jpg")
gray_R = cv2.cvtColor(ut_R,cv2.COLOR_BGR2GRAY)
kp_R, ds_R = sift.detectAndCompute(gray_R, None)

left_draw_kp = cv2.drawKeypoints(gray_L, kp_L, ut_L,\
    flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
right_draw_kp= cv2.drawKeypoints(gray_R, kp_R, ut_R, \
    flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
plt.imshow(left_draw_kp, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x24e5039d148>
```



In [ ]:
```
plt.imshow(right_draw_kp, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x24e50423ac8>
```



Compute distances between every descriptor in one image and every descriptor in the other image. Alternatively, experiment with computing normalized correlation, or Euclidean distance after normalizing all descriptors to have zero mean and unit standard deviation. Report your choices. Note: You are not allowed

to use built-in functions to match features for you, including but not limit to cv2.BFMatcher. However, you can use them to debug your code and compare to your implementation.

In [ ]:
```
kp_L_loc = np.array([[kp.pt[0],kp.pt[1]] for kp in kp_L])
kp_R_loc = np.array([[kp.pt[0],kp.pt[1]] for kp in kp_R])
distfunc= lambda p1, p2: np.sqrt(((p1-p2)**2).sum())
dist = np.asarray([[distfunc(p1, p2) for p2 in ds_R] for p1 in ds_L])
print(dist.shape)
```

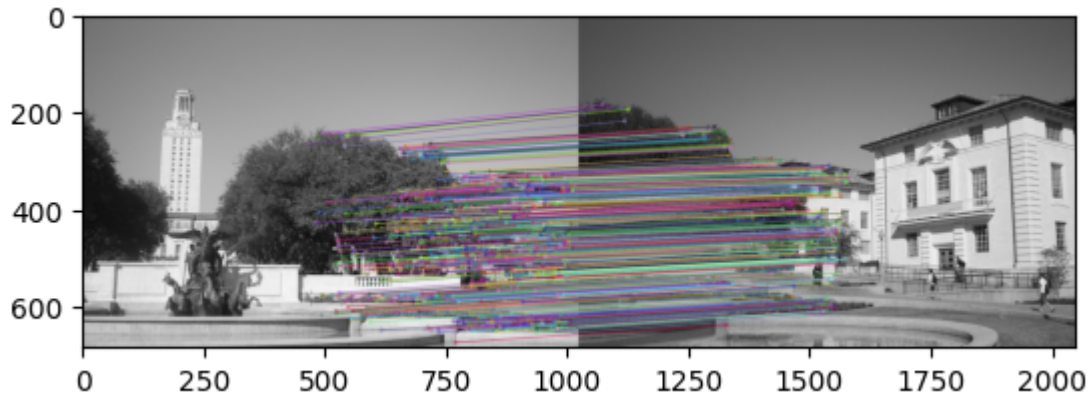
```
(5314, 4195)
```

Select putative matches based on the matrix of pairwise descriptor distances obtained above. You can (i) select all pairs whose descriptor distances are below a specified threshold; (ii) select the top few hundred descriptor pairs with the smallest pairwise distances; or (iii) as described in lecture, compute the ratio test described in lecture (nearest-neighbor to second-nearest-neighbor ratio test). Report your choices

In [ ]:
```
t = 0.5
matches = []
des_indL = []
des_indR = []

for i in range(len(dist)):
    part = dist[i]
    min_idx = np.argpartition(part, 2)
    ratio = part[min_idx[0]]/part[min_idx[1]]
    if ratio < t:
        indL = i
        indR = part.argmin()
        m = cv2.DMatch(indL, indR, dist[indL][indR])
        matches.append(m)
        des_indL.append(indL)
        des_indR.append(indR)

match = cv2.drawMatches(gray_L, kp_L, gray_R,
                        kp_R, matches, None, flags=2)

des_indL = np.array(des_indL)
des_indR = np.array(des_indR)

print(f"Number of matches {len(matches)}")
plt.imshow(match, cmap='gray')
```

```
Number of matches 724
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x24e504ee088>
```



Each row of the distance matrix represents a single descriptor and it's distance to every other descriptor. The 2 lowest distances are then ratio'd and compared with an arbitrary threshold inbetween the range of 0.3 and 0.7. In this case, 0.5 was chosen. If the ratio was below the threshold, the row index and column index was saved for their respective picture. The matches were then drawn based on those indices.

Run RANSAC to estimate a homography mapping one image onto the other. For the best fit, Report the number of inliers and the average residual for the inliers (squared distance between the point coordinates in one image and the transformed coordinates of the matching point in the other image). Also, display the locations of inlier matches in both images. (i.e. for the inliers matches, show lines between matching locations in the two images). You can use cv2.drawMatches to draw matches. Note: You need to implement RANSAC and calculate the transform matrix. You are not allowed to use functions that do RANSAC in one line, including but not limit to cv2.findHomography or cv2.getPerspectiveTransform. However, it's a good idea to use them to debug your implementation.

In [ ]:
```
kp_Lx = []
kp_Ly = []
for i in des_indL:
    kp_Lx.append([kp_L_loc[i][0]])
    kp_Ly.append([kp_L_loc[i][1]])

kp_Rx = []
kp_Ry = []
for i in des_indR:
    kp_Rx.append([kp_R_loc[i][0]])
    kp_Ry.append([kp_R_loc[i][1]])

kp_Lx = np.array(kp_Lx)
kp_Ly = np.array(kp_Ly)
kp_Rx = np.array(kp_Rx)
kp_Ry = np.array(kp_Ry)

kp_Lxy = np.hstack((kp_Lx, kp_Ly))
kp_Rxy = np.hstack((kp_Rx, kp_Ry))

iter = 250
avg_res = None
max_in = 0
bH = None
```

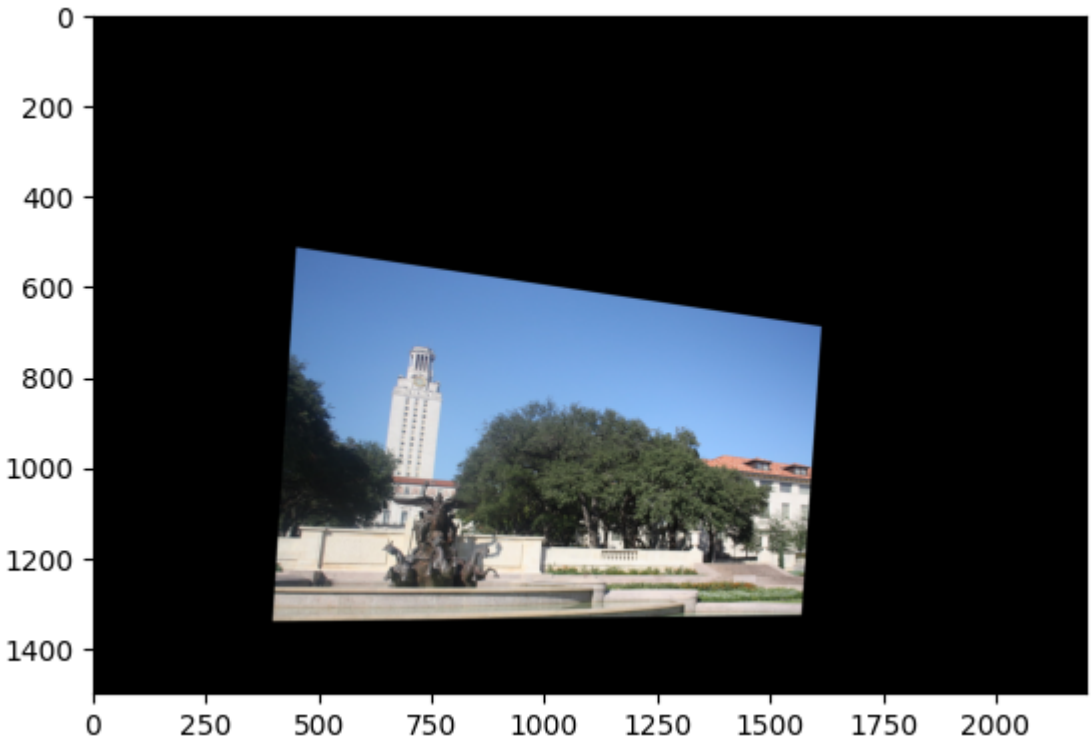
```
for i in range(iter):
    p = np.random.choice(len(des_indL), 4, replace=False)
    lx = []
    ly = []
    for i in des_indL[p]:
        lx.append([kp_L_loc[i][0]])
        ly.append([kp_L_loc[i][1]])
    rx = []
    ry = []
    for i in des_indR[p]:
        rx.append([kp_R_loc[i][0]])
        ry.append([kp_R_loc[i][1]])
    A = np.hstack((np.array(lx),np.array(ly),np.array(rx),np.array(ry)))
    H = fit_homography(A)
    trf = homography_transform(kp_Lxy, H)
    ransac_dist = np.linalg.norm(trf - kp_Rxy, axis=1)
    inl = np.sum(ransac_dist < 1)
    if inl > max_in:
        max_in = inl
        bH = H
        avg_res = np.mean(ransac_dist**2)

print(avg_res)
print(max_in)
print(bH)
```

```
3.4141084229409064
534
[[-2.17801219e-03  1.34685952e-04  9.59998901e-01]
 [-2.88654368e-04 -2.02849730e-03  2.79982786e-01]
 [-4.83192054e-07 -1.16607189e-08 -1.67021126e-03]]
```

Warp one image onto the other using the estimated transformation. To do this, you will need to learn about cv2.warpPerspective. Please read the documentation.

In [ ]:
```
hl, wl, c = ut_L.shape
ut_L = cv2.imread("p2/uttower_left.jpg")
ut_L = cv2.cvtColor(ut_L, cv2.COLOR_BGR2RGB)
trans_L = np.array([[1, 0, wl], [0, 1, hl], [0, 0, 1]])
warp_L = cv2.warpPerspective(ut_L, trans_L @ bH, (2200, 1500))

plt.imshow(warp_L)
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x24e50179308>
```

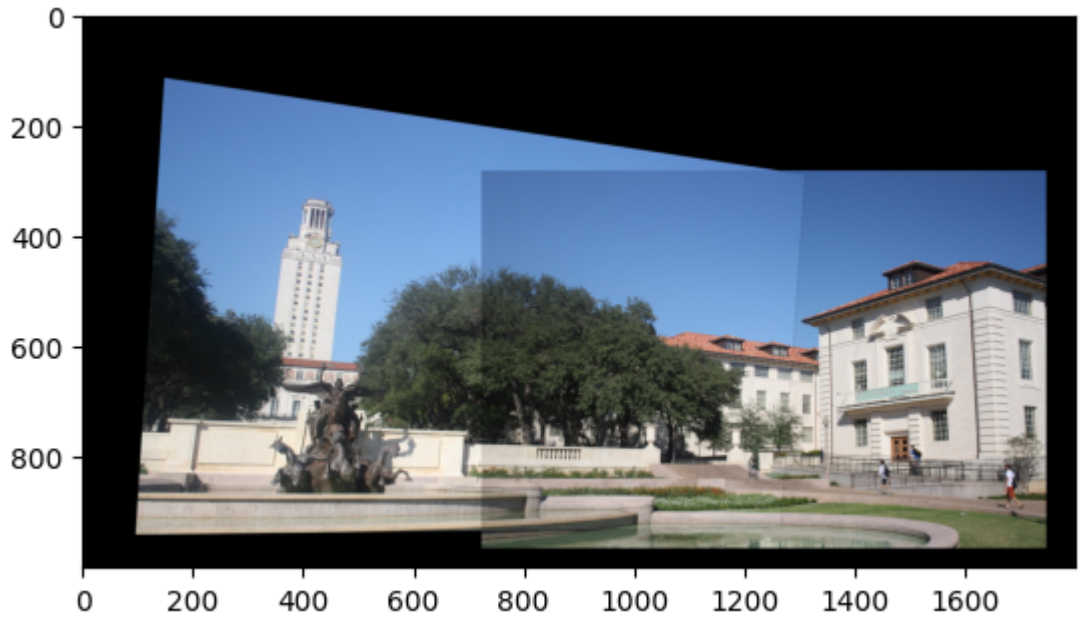


Create a new image big enough to hold the panorama and composite the two images into it. You can composite by simply averaging the pixel values where the two images overlap. Display a colored image, which might look like this:

In [ ]:
```
ut_R = cv2.imread("p2/uttower_right.jpg")
ut_R = cv2.cvtColor(ut_R, cv2.COLOR_BGR2RGB)
hr, wr, c = ut_R.shape
combined = warp_L.astype('int64')
for i in range(hr):
    for j in range(wr):
        c_val = [combined[hl + i][wl + j][0], \
            combined[hl + i][wl + j][1], \
                combined[hl + i][wl + j][2]]
        if c_val == [0, 0, 0]:
            combined[hl + i][wl + j] = ut_R[i][j]
        else:
            combined[hl + i][wl + j] = (combined[hl + i][wl + j]+ut_R[i][j])/2
```

In [ ]:
```
combined_cropped = combined[400:1400,300:2100]
plt.imshow(combined_cropped)
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x24e50207748>
```



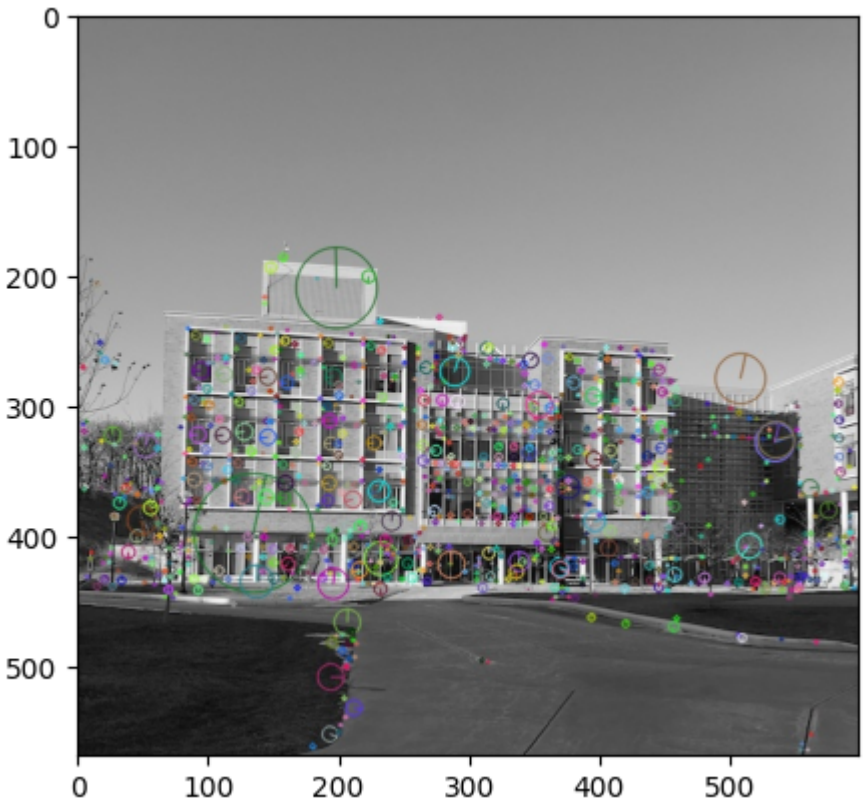
Run your code on p2/bbb left.jpg and p2/bbb right.jpg. Display the detected feature points, the matching result, the inliers after RANSAC, and the stitched image.

In [ ]:
```
bbb_L = cv2.imread("p2/bbb_left.jpg")
bbb_R = cv2.imread("p2/bbb_right.jpg")
bbb_Combined = stitchimage(bbb_L, "bbb_left", bbb_R, "bbb_right")
```

```
bbb_left_kp.jpg is saved!
bbb_right_kp.jpg is saved!
bbb_match.jpg is saved!
Max Number of Inliers: 302
```

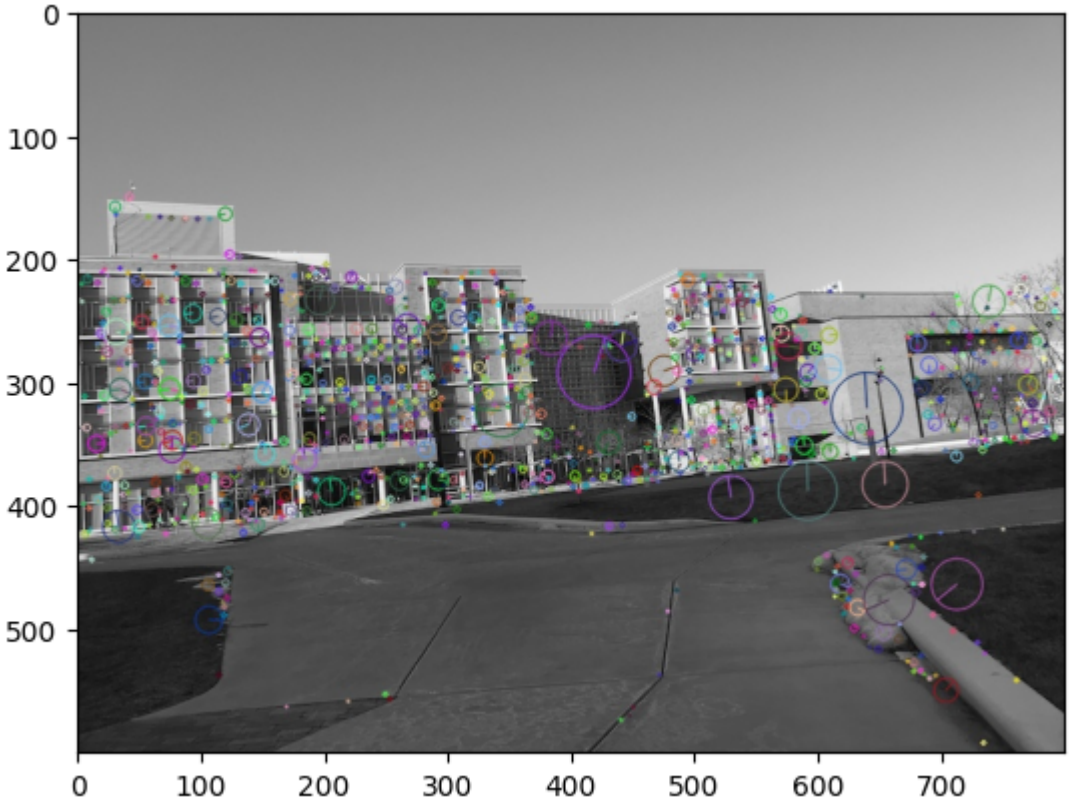
In [ ]:
```
bbb_L_kp = plt.imread("bbb_left_kp.jpg")
plt.imshow(bbb_L_kp)
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x24e4fbf13c8>
```



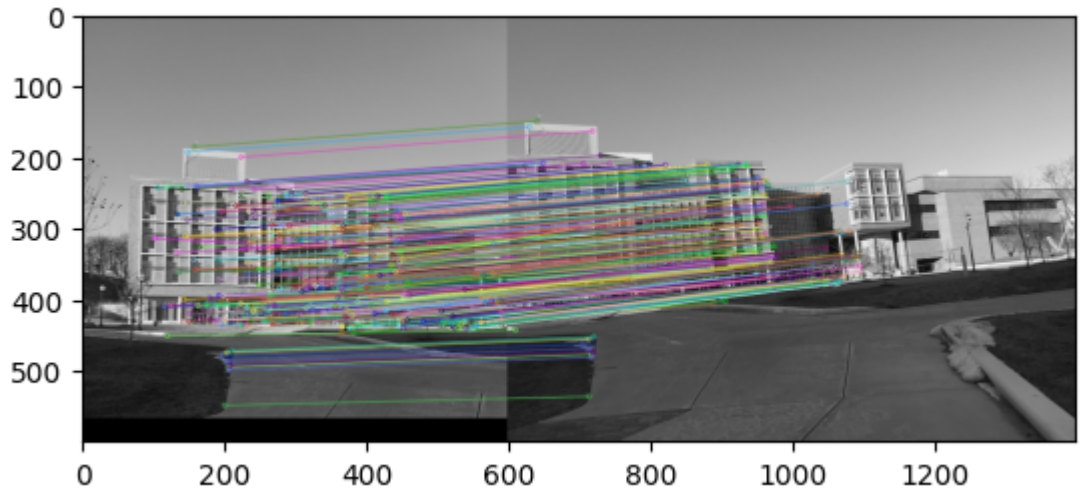
In [ ]:
```
bbb_R_kp = plt.imread("bbb_right_kp.jpg")
plt.imshow(bbb_R_kp)
```

Out[ ]: <matplotlib.image.AxesImage at 0x24e4fd7bc48>



In [ ]:
```
bbb_match = plt.imread("bbb_match.jpg")
plt.imshow(bbb_match)
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x24e4fd68988>
```



In [ ]:
```
bbb_Combined_cropped = bbb_Combined[500:1150,400:1400]
plt.imshow(bbb_Combined_cropped)
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x24e4ffcb988>
```