Assignment 2

IMAGE MOSAICS

Done by:

Jailan Hussein 5733

Salma ElBanna 5734

Jolie Sameh 5865

Nourena Nabil 5510

Faculty of Engineering, Alexandria University

Table of Contents

Explanation of the code

1. Getting Correspondences

To get the correspondences we will choose four distinctive points in the image that appear in both views. The four points from each image will be used later as matches in the homography function.

2. Compute the Homography Parameters

Once we have obtained correspondences between the images, we calculate homography matrix. The homography matrix will use these four matching points, to estimate a relative orientation transform within the two images. i.e. it'll solve for the equation

$$Ix = H \times Iy$$

Hence, it solves for the matrix H

We use a system of linear equations Ax = b, where the 8 unknowns
of H are stacked into an 8-vector x , the 2n-vector b contains image points from one view, and the 2n × 8 matrix A is filled appropriately so that the full system gives us λp = Hp.

$$\mathbf{A}_i = \begin{bmatrix} -x & -y & -1 & 0 & 0 & 0 & xx' & yx' & x' \\ 0 & 0 & 0 & -x & -y & -1 & xy' & yy' & y' \end{bmatrix}$$

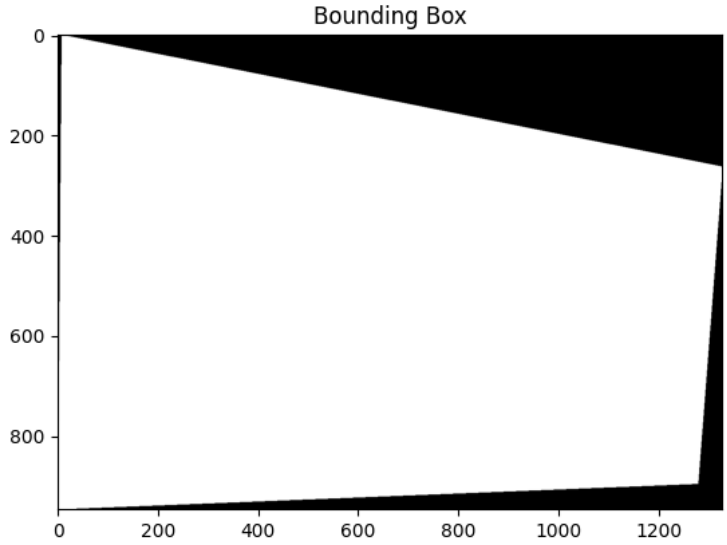
There are only 8 unknowns in H because we set H3,3 = 1 and we solve for the unknown homography matrix parameters.

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

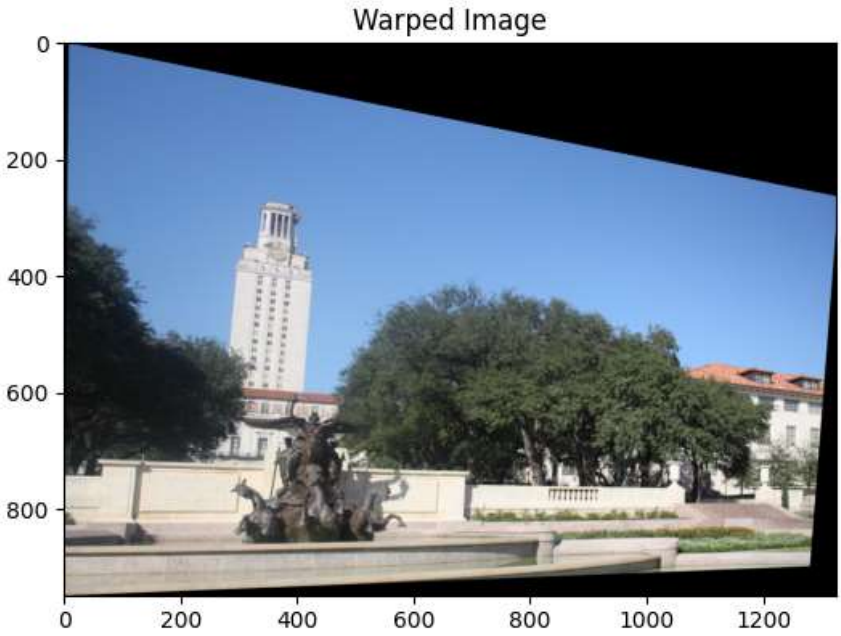
```
Homography matrix  [[ 2.03282543e-03 -8.75992408e-05 -9.42789597e-01]
 [ 3.35494710e-04  1.85537455e-03 -3.33374289e-01]
 [ 6.29990783e-07  9.56624594e-08  1.28999805e-03]]
```

3. Warping Between Image Planes

So, to warp, essentially change the field of view, we apply the homography matrix to the image. First, we determine where each of the four corners of the source image will land after warping it into the destination image using **forward warping**. After warping the points from the source image into the reference frame of the destination, and compute the **bounding box** in that new reference frame.

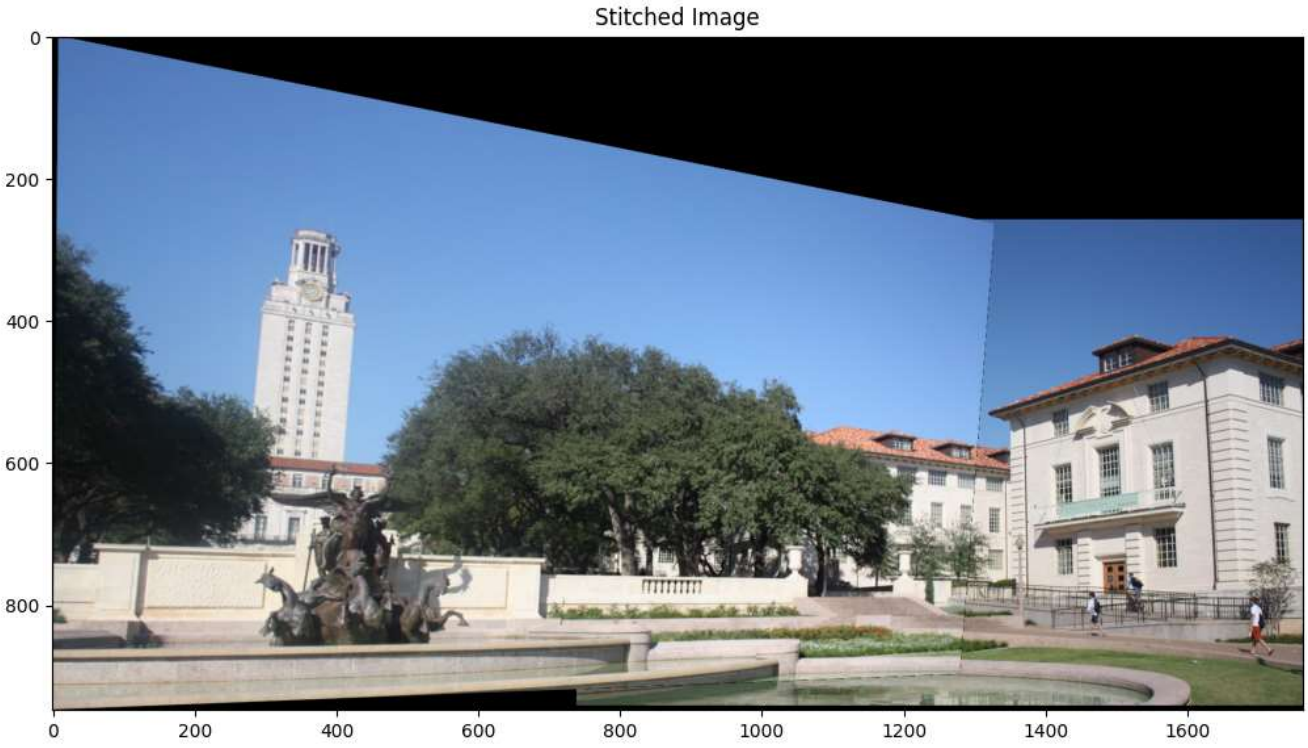


Then to avoid holes, we use **inverse warping**. Each pixel in the obtained bounding box will undergo a multiplication with the inverse of the homography. In case we landed between two pixels in the actual image, we use **linear interpolation** to determine the intensity.



4. Create the output mosaic

Once we have the source image warped into the destination images frame of reference, we can create a merged image showing the mosaic. We create a new image large enough to hold both views and then overlay one view onto the other. Then we crop the merged image to remove the black regions where no data is available.

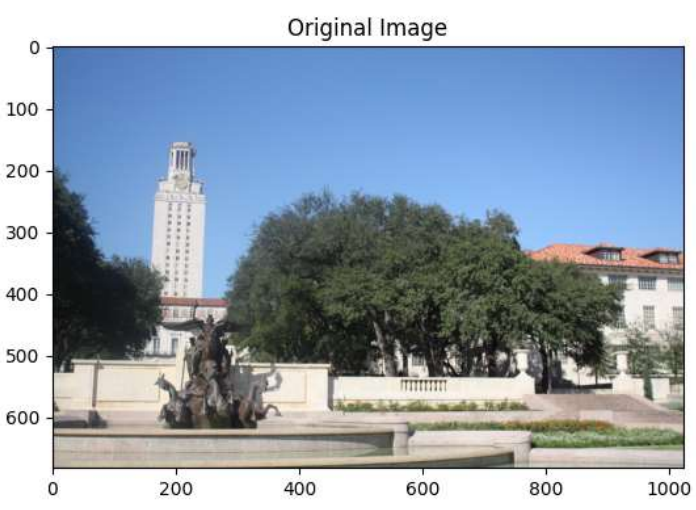


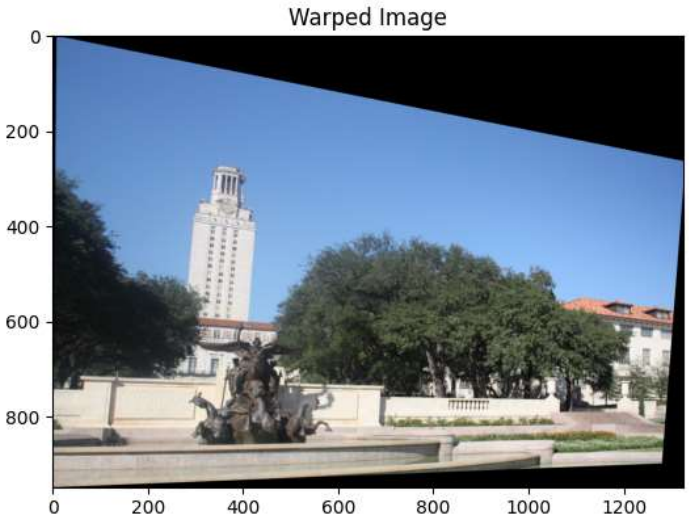
Output for test images

1) Sample Run

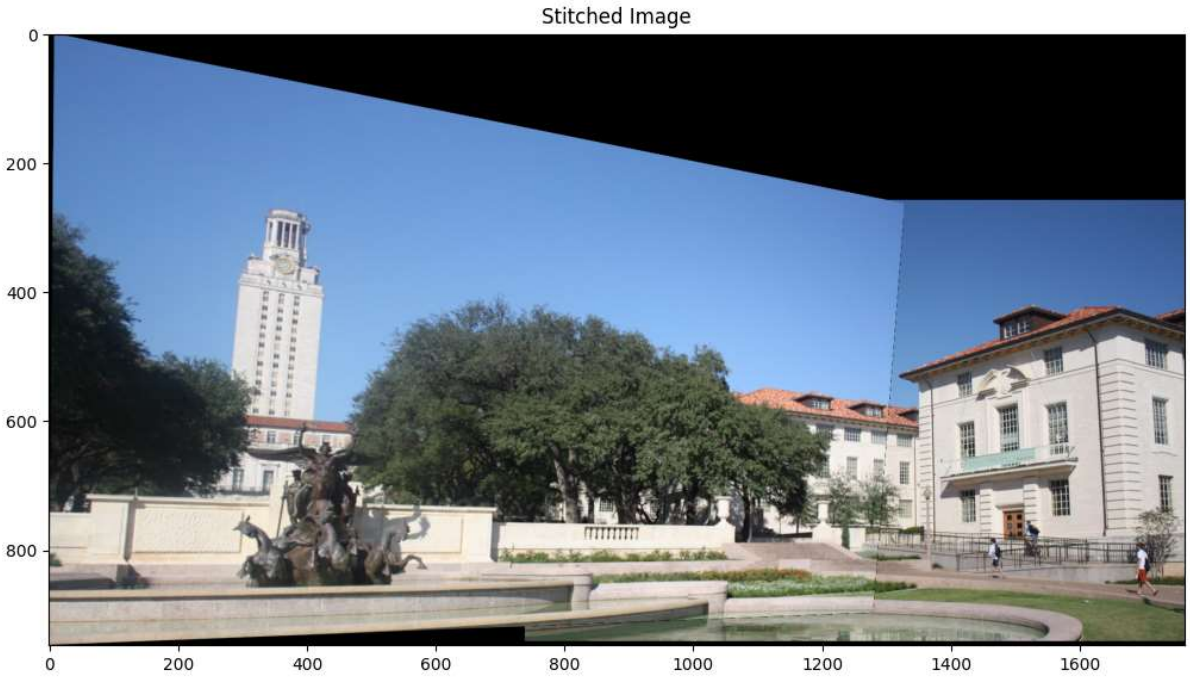
1. Test images:

2. Warped image:



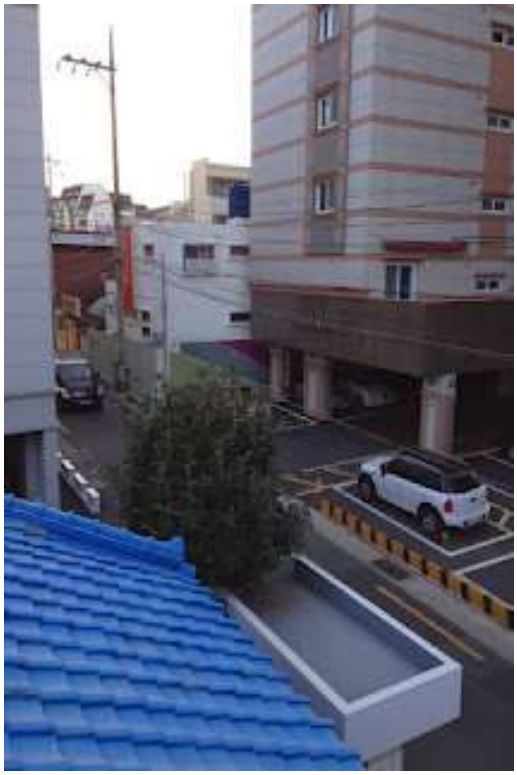


3. Stitched image:

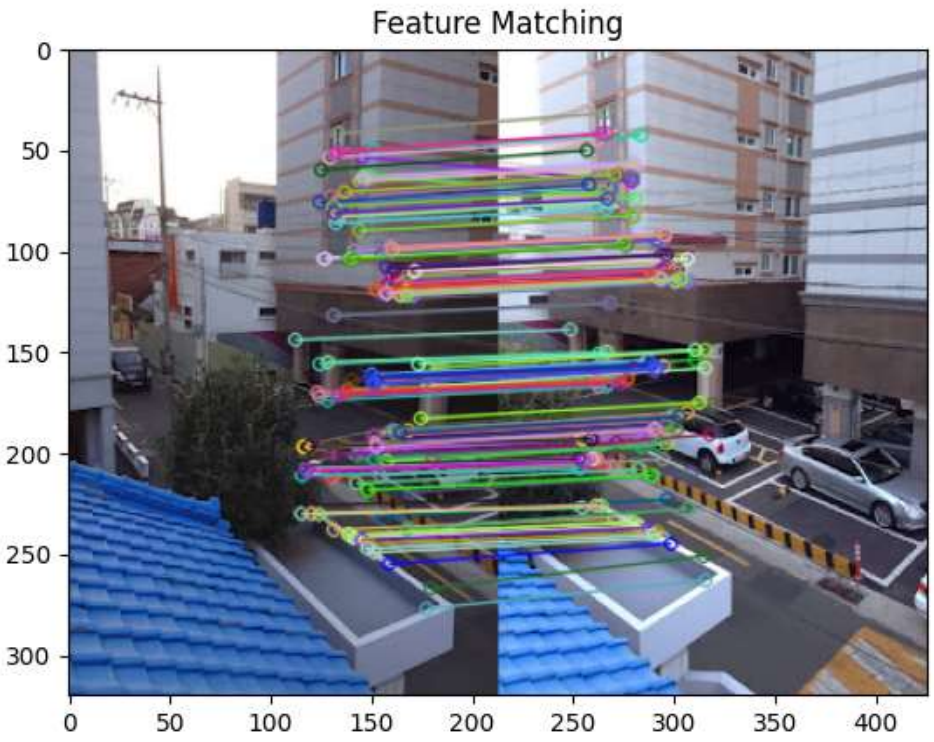


2) Sample Run

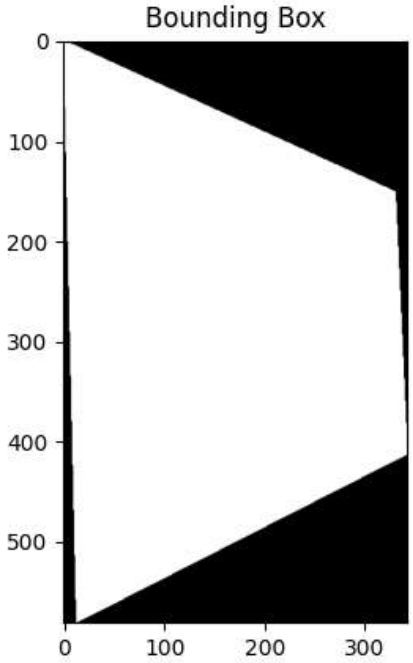
1. Test images:



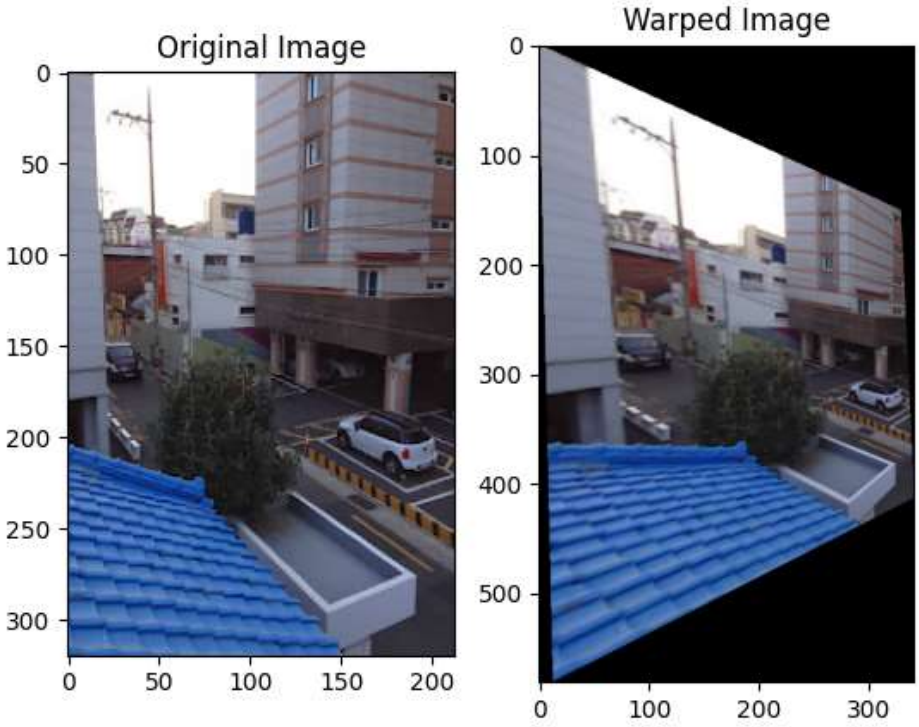
2. Feature matching:



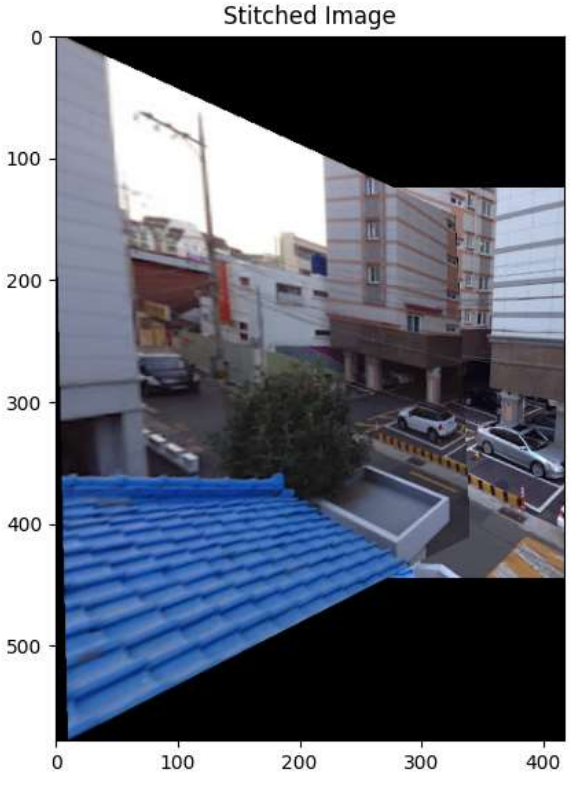
3. Bounding Box



4. Warped image:



5. Stitched image:



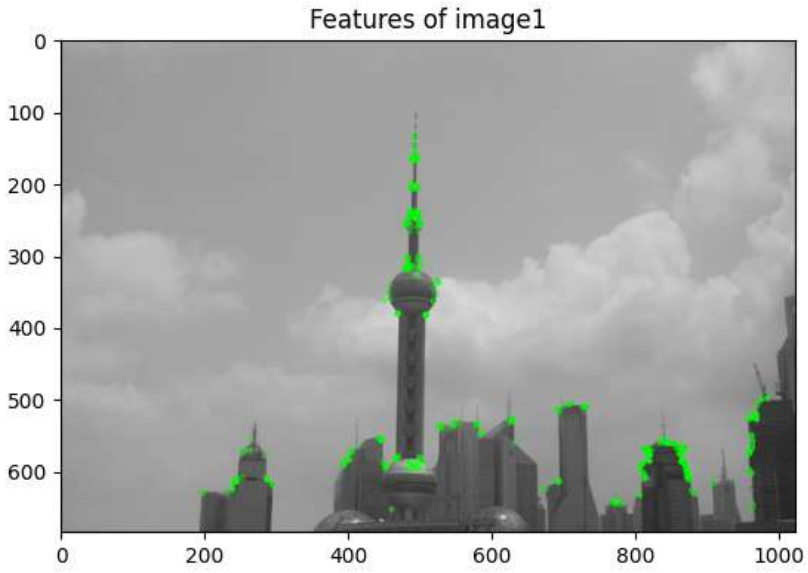
Bonus

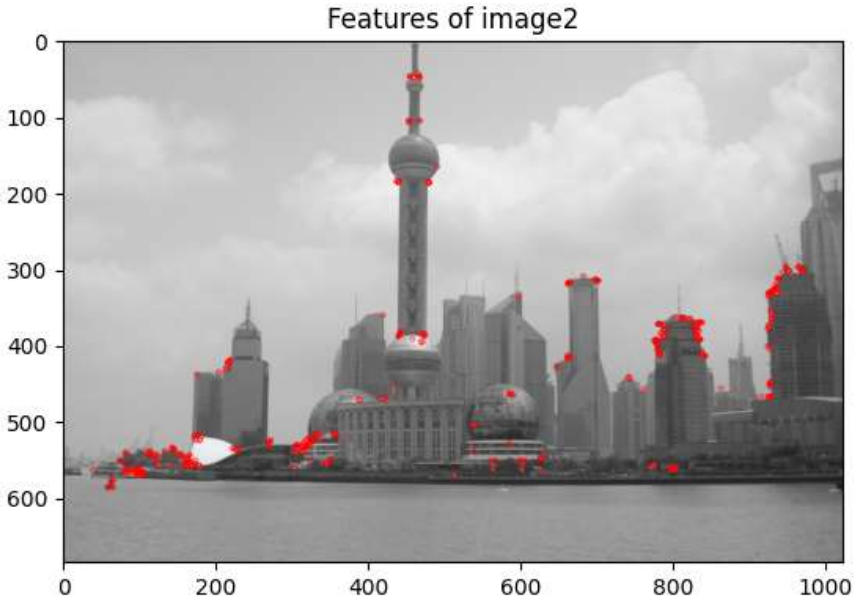
Step 1: Feature extraction

For the feature extraction we use the ORB detector since it is a good alternative to SIFT and SURF in computation cost, matching performance and mainly the patents. It is fusion of fast keypoint detector and brief descriptor with many modifications to enhance the performance.

```
orb = cv2.ORB_create()
descriptors,keypoints=orb.detectAndCompute(image_gray,None)
```

The features of the images are displayed in the plots below:



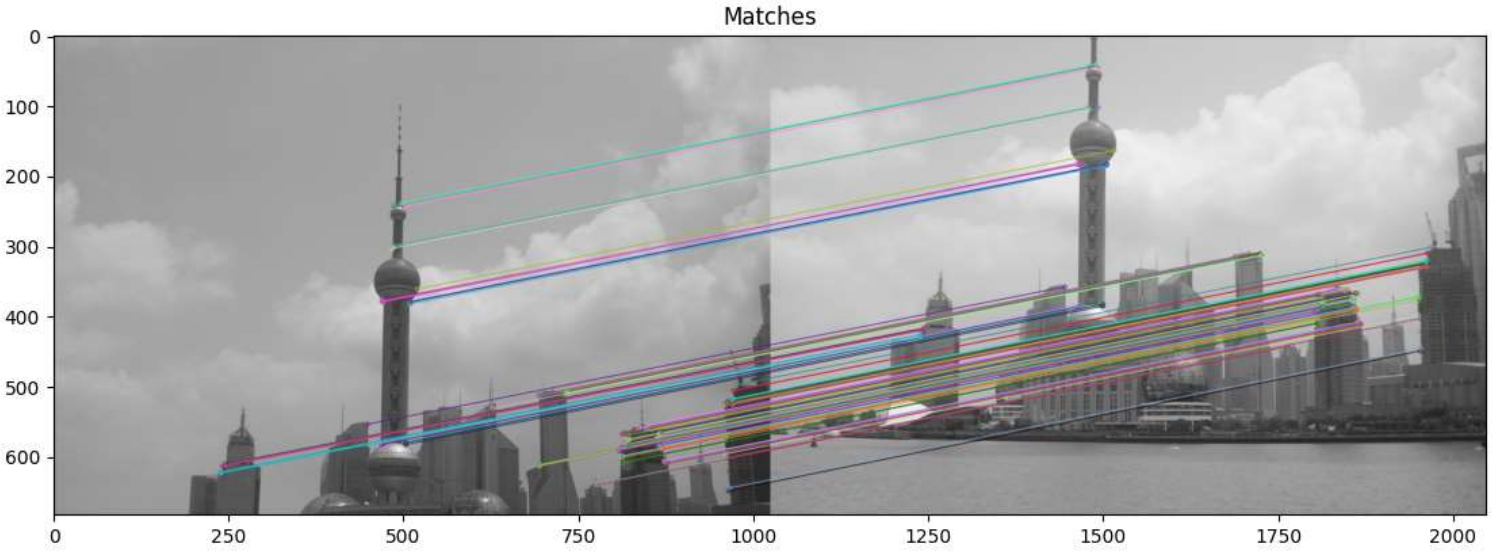


Step 2: Matching correspondences between images

Once we have got the descriptors and keypoints of the 2 images we find the correspondences between them to obtain the overlapping points. These overlapping points will give us an idea of the orientation of the second image w.r.t to the other one. And based on these common points, we get an idea whether the second image has just slid into the bigger image or has it been rotated and then overlapped, or maybe scaled down/up and then fitted. All such information is yielded by establishing correspondences.

For matching, we use either FLANN or BFMatcher, that is provided by OpenCV.

```
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck = True)
matches = bf.match(descriptor1, descriptor2)
```

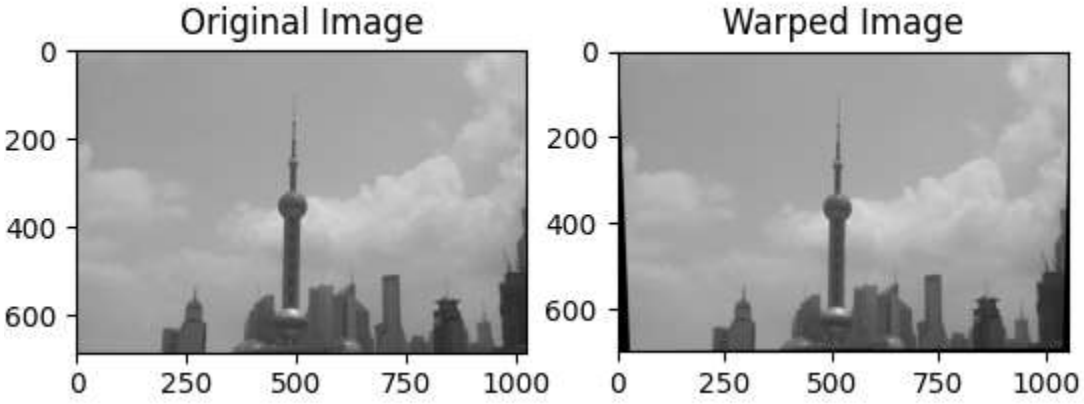


Step 3: Compute Homography

Once we have obtained matches between the images, our next step is to calculate the homography matrix. The homography matrix will use these matching points, to estimate a relative orientation transform within the two images.

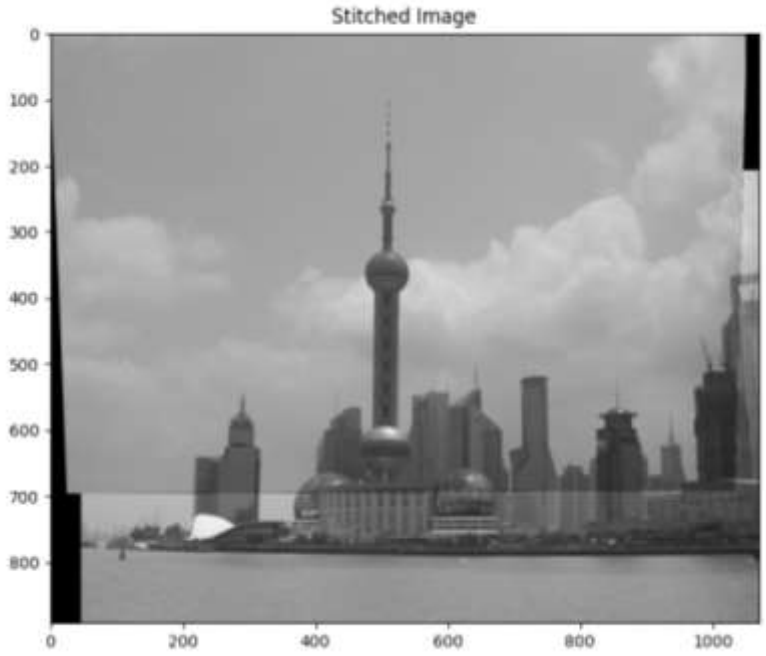
Step 4: Warping

After calculating the homography matrix we apply the same warping function that was mentioned before.



Step 5: Stitching

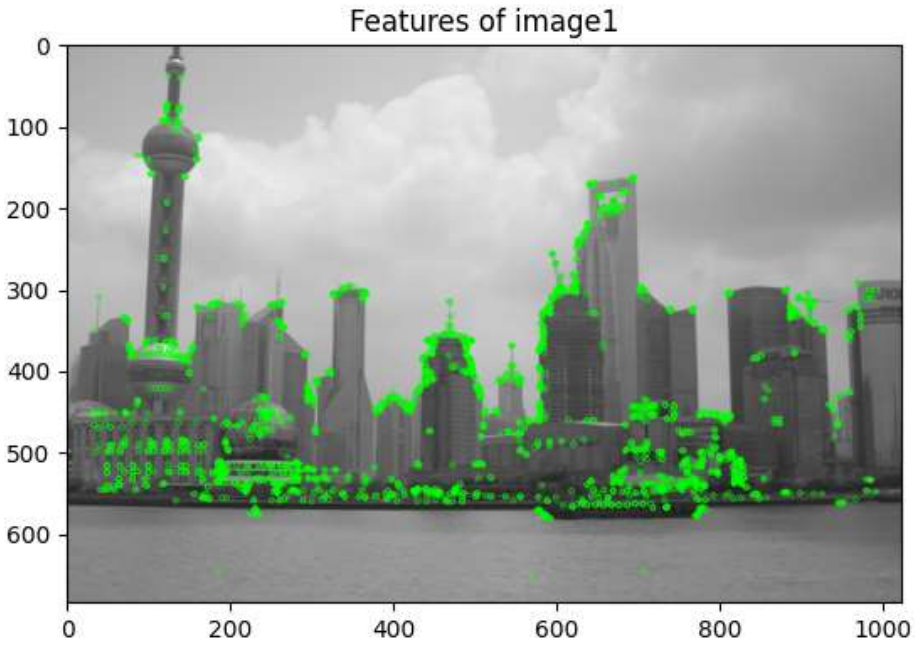
Once, we have obtained a warped image, we simply add the warped image along with the second image. We create a new image large enough to hold both views and then overlay one view onto the other. Then we crop the merged image to remove the black regions where no data is available.

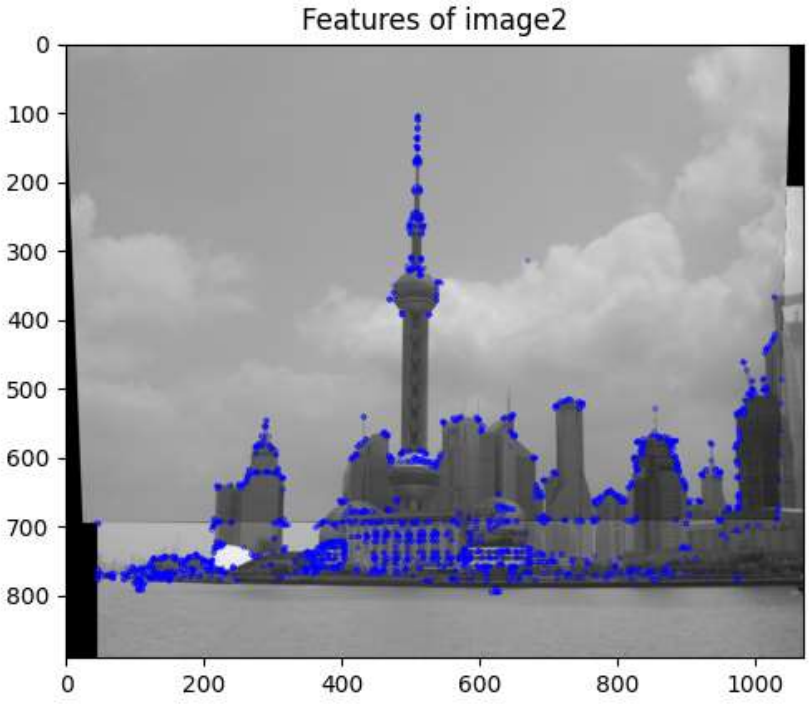


Step 6: Repeat

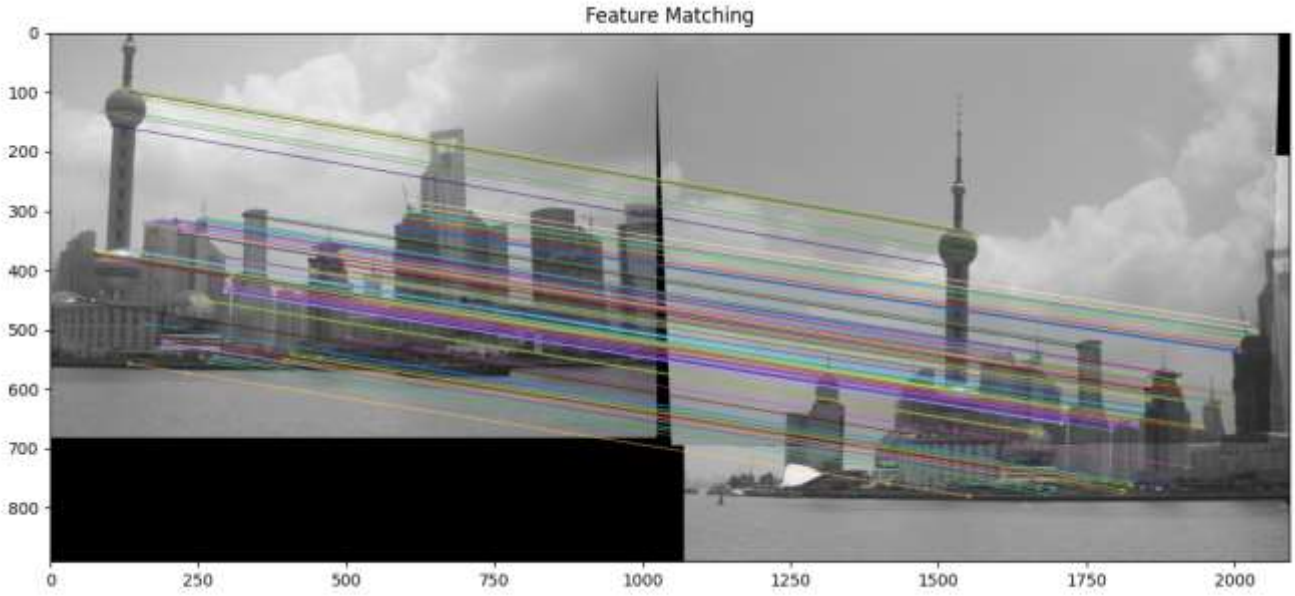
We repeat steps one through five between the stitched image and the third image.

1. Feature extraction





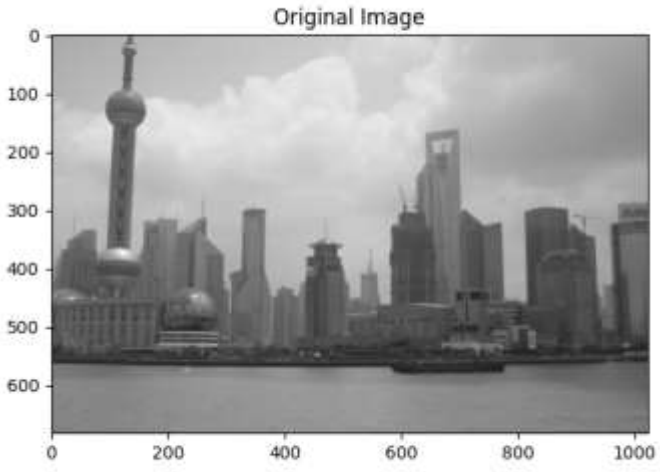
2. Matching correspondences between images

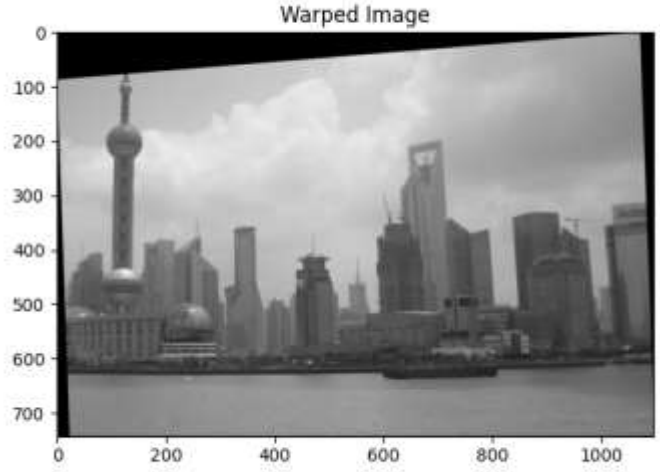


3. Compute Homography

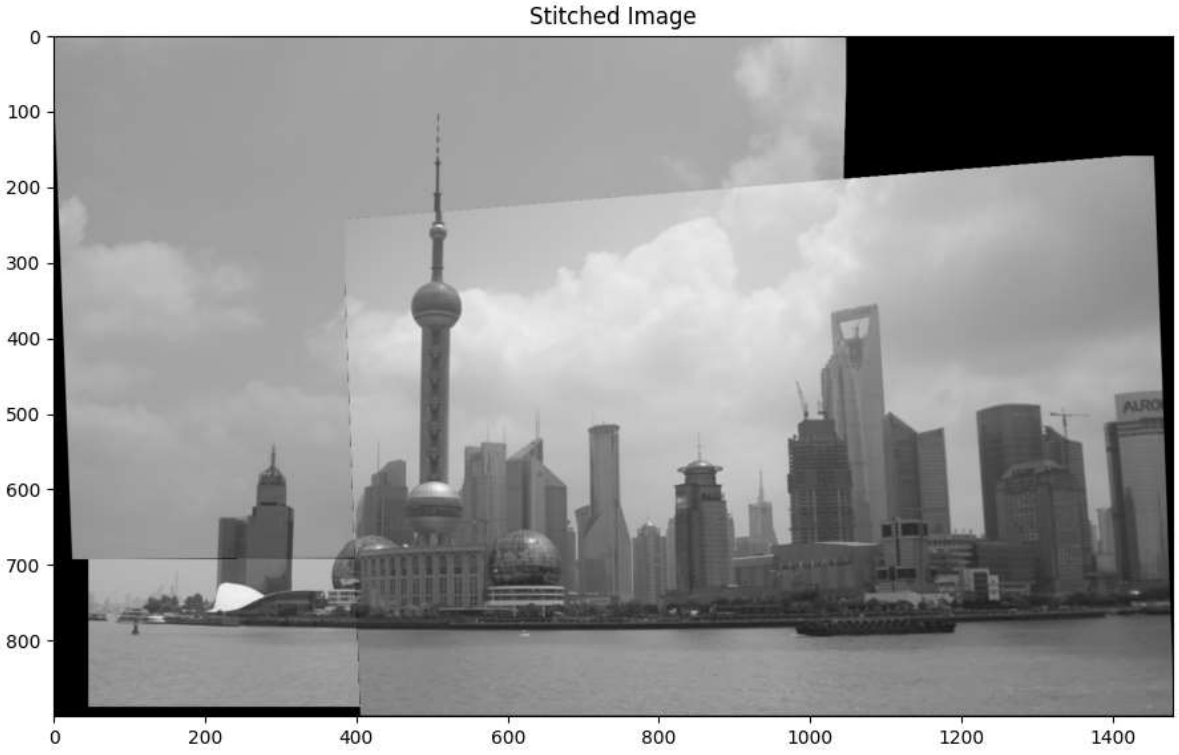
   We apply the same homography function that was mentioned before.

4. Warping





5. Stitching


Stitched Image
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
1200
1400