Assignment – 3

2018701022

Question – 1:

Description:

- In Dense SIFT will get a SIFT descriptor at every location in image. Dense SIFT collects more features at each location and scale in an image, increasing recognition accuracy accordingly.
- In this larger set of local image descriptors computed over a dense grid. It provides more information than sparse SIFT.

Procedure:

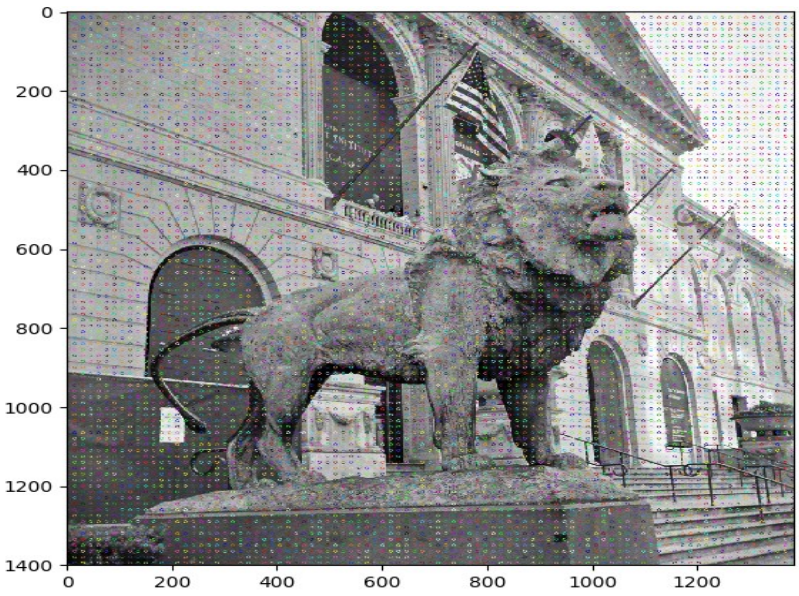
- Import the stereo image pair. Divide it into two parts. Now we need to apply Dense based SIFT on this two parts and find out the matching points.
- Dense SIFT is not available freely in openCV. So by using SIFT only applying Dense SIFT.
- First calculated the Keypoints over dense grid (image) using SIFT. Here considering grid size **5** × **5**.
- Passing the this grid entire image calculate the descriptor for each grid by using **sift.compute()** function.

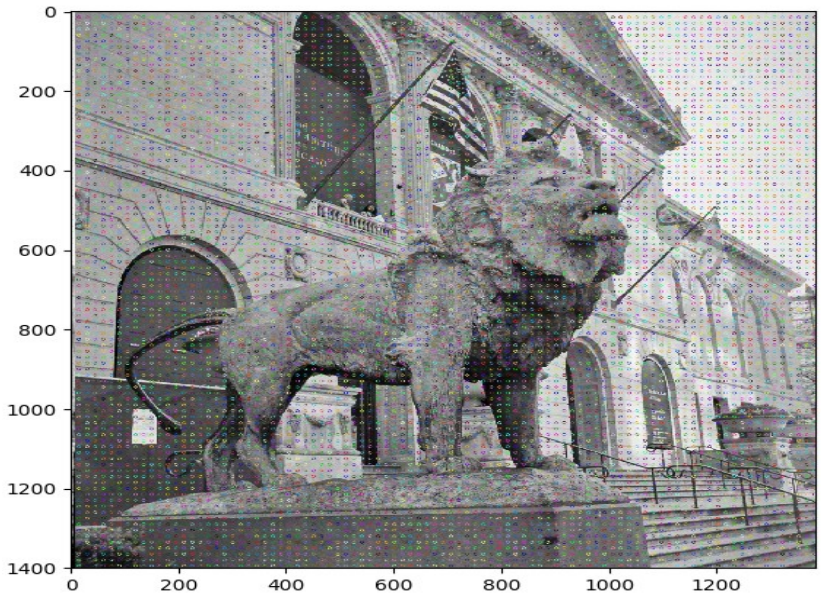
Code:

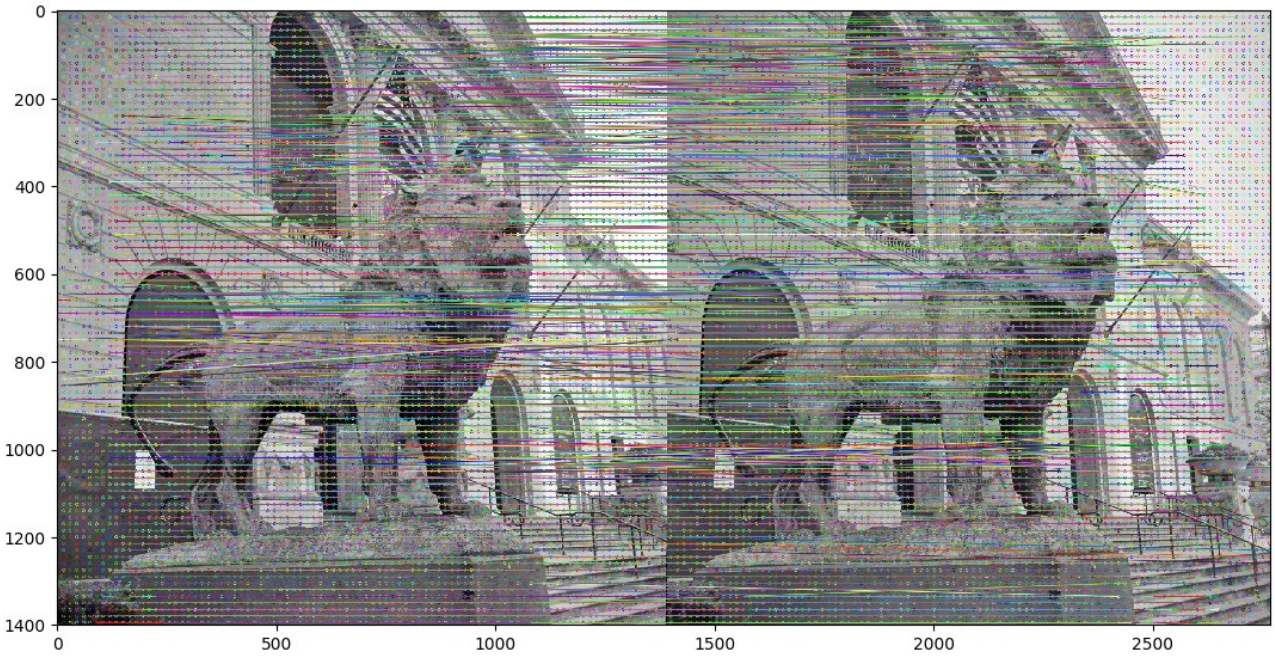
```
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('StereoImages/Stereo_Pair1.jpg')          #inputimage
w, h = int(gray.shape[0]), int(gray.shape[1]/2)

#Dividing stereo image into two parts

left_img = u[:w][:,:h]
right_img = u[:w][:,h:]

#RGBtoGRAY

gray1= cv2.cvtColor(left_img ,cv2.COLOR_BGR2GRAY)
gray2= cv2.cvtColor(right_img ,cv2.COLOR_BGR2GRAY)
```

```
#SIFT object creation
sift = cv2.xfeatures2d.SIFT_create()
grid_size = 5

#Finding the keypoints

kp1 = [cv2.KeyPoint(x, y, grid_size) for y in range(0, gray1.shape[0],
grid_size) for x in range(0, gray1.shape[1], grid_size)]
kp2 = [cv2.KeyPoint(x, y, grid_size) for y in range(0, gray3.shape[0],
grid_size) for x in range(0, gray2.shape[1], grid_size)]

#Drawing Keypoints

img1=cv2.drawKeypoints(gray1, kp1, left_img)
img2=cv2.drawKeypoints(gray2,kp2, rghit_img)

#Computing the dense descriptors using sift.computr()
_, dense_feat1 = sift.compute(gray1, kp1)
_, dense_feat2 = sift.compute(gray2, kp2)

#Finding the matches between pair of images
bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
matches = bf.match(dense_feat1, dense_feat2)
matches = sorted(matches, key = lambda x:x.distance)

#Draw the matching windows in stereo images

img3 = cv2.drawMatches(img1,kp1,img2,kp2,matches[:100],None,
flags=2)
plt.imshow(img3), plt.show()
```

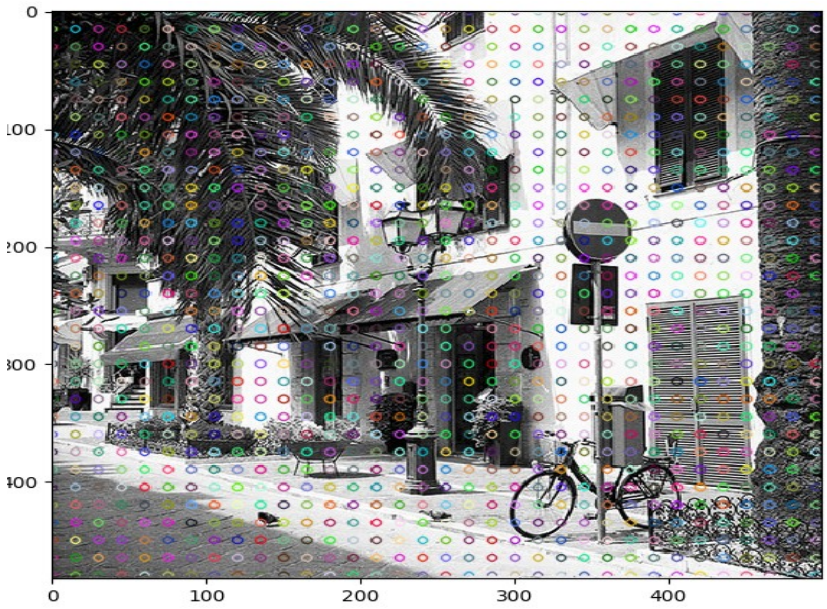
Output Images: Stereo Image1:

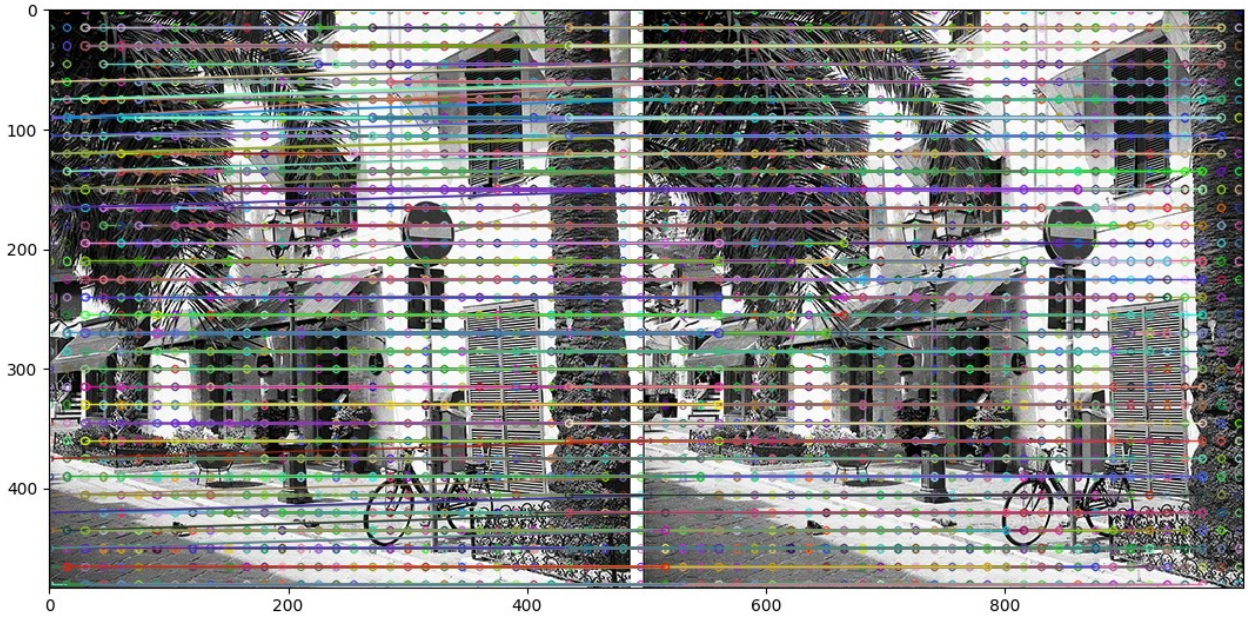




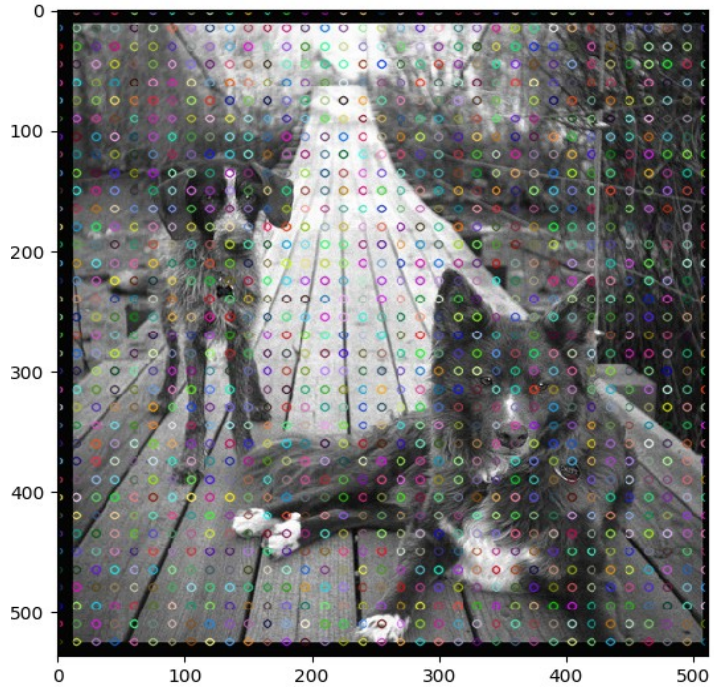


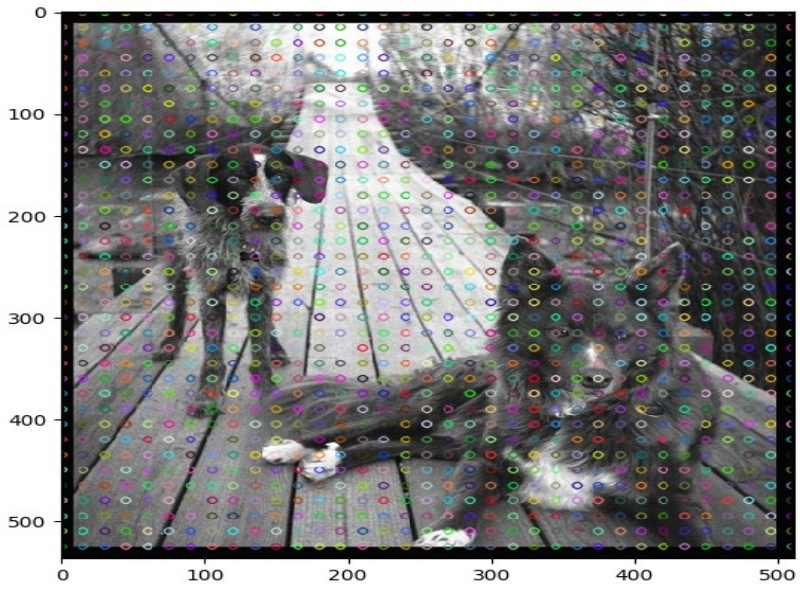
Stereo Image2:

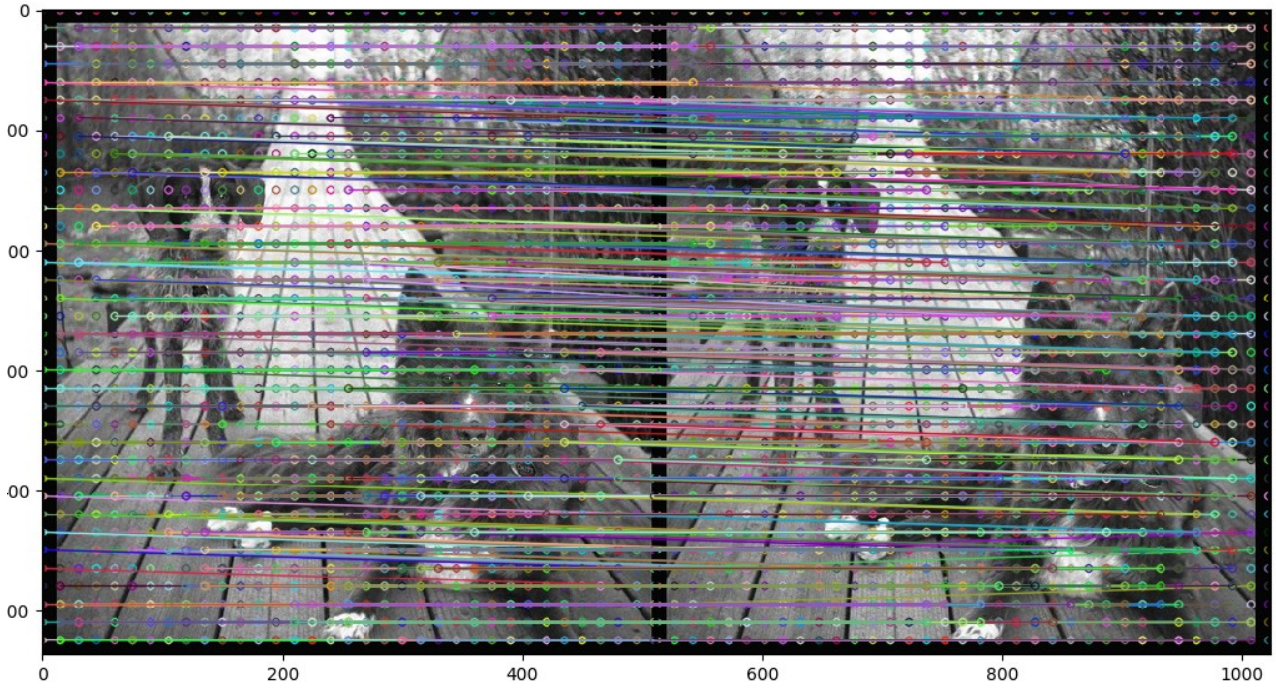




Stereo Image 3:







Question 2: Intensity Window-based correlation

- Correspondence between two images problem can be classified into two types. In that of type is **Intensity Window-based correlation.** In intensity window-based approach the matching process is applied on intensity profiles of two images.
- If we close look at the intensity profiles from the corresponding row of the image pair reveals that the two intensity profiles differ only by a horizontal shift and a local foreshortening.
- Here, we'll match the parallel stereo images using correlation window. We will calculate the correlation between two image patches and find out the matching patches.

Procedure:

- Take the stereo image pair. Divide the stereo pair image into two images. Convert these two images from RGB to Gray.
- Here I am considering the window size **5 × 5.** So the striding also **5 × 5**.
- The first step in this, take window **5 × 5** size in left image, and take same size window in right image, calculate the correlation between these two windows.

$$\text{Correlation} = \frac{\sum(\text{window1} \cdot \text{window2})}{\sqrt{(\text{window} \cdot \text{window1})} \times \sqrt{(\text{window} \cdot \text{window1})}}$$

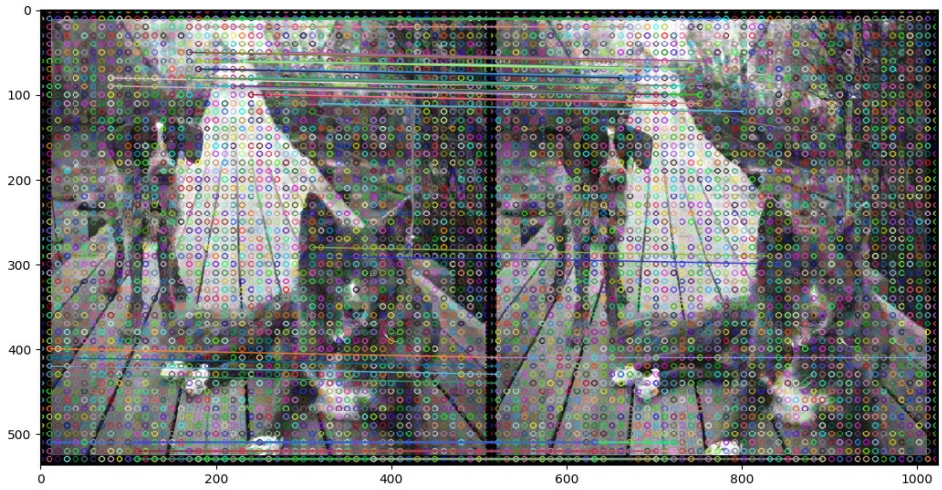
- Same like find out the correlation for all the windows. Take the minimum correlation value corresponding window in right image. This window is the correspondence window of left image window.

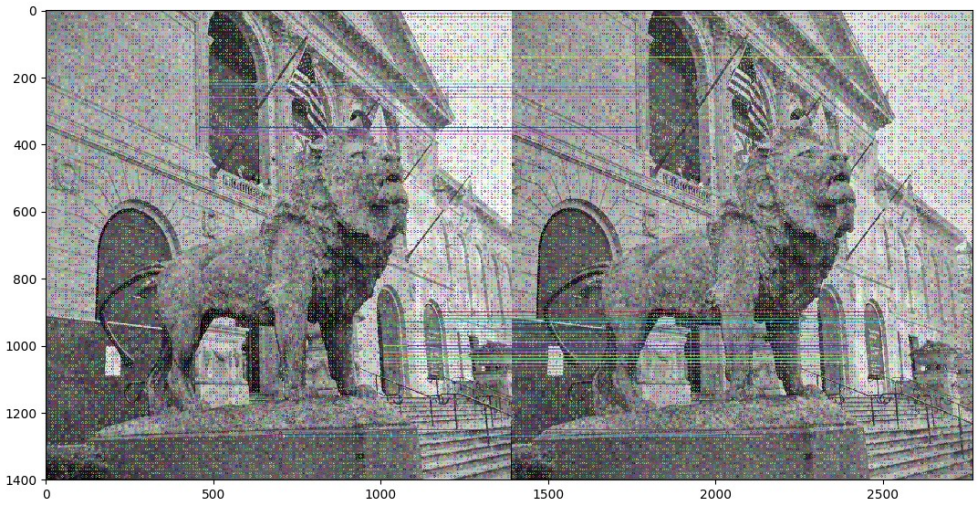
Code:

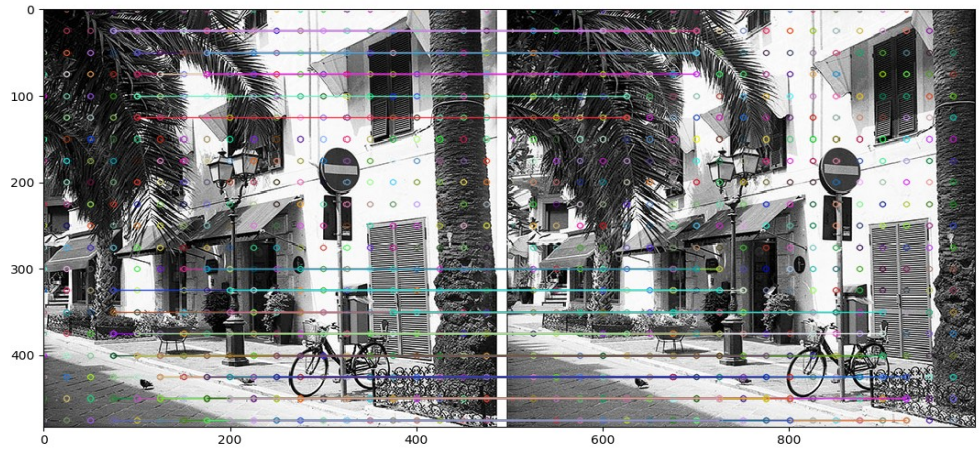
```
import cv2
import math
import numpy as np
from skimage.util import view_as_windows
from matplotlib import pyplot as plt

#Reading the image and converting into gray
u = cv2.imread('StereoImages/Stereo_Pair1.jpg',0)
w, h = int(u.shape[0]), int(u.shape[1]/2)

#Dividing the stereo image into two parts: left_img and right_img
left = u[:w][:,:h]
right = u[:w][:,h:]

temp1=0, temp2=0, count=0

img1 = []
img2 = []

#window size (5,5)
w, h = int(left.shape[0] / 5), int(left.shape[1] / 5)
```

```
#Extracting the patches from right image
for i in range(h):
    for j in range(w):
        count = count+1

        #Single patch in left_img
        sample = left[temp1:temp1+5][:,temp2:temp2+5]
        temp2 = temp2+5
        if(sample.shape == (5,5)):
            img1.append(sample)
            min_d = 9999999

            #Finding the correlation between two windows
            for pat in patches:
                t1 = np.sum(np.dot(sample,pat))
                t2 = math.sqrt(np.sum(sample**2))
                t3 = math.sqrt(np.sum(pat**2))
                corr = t1 / (t2+t3)              #Correlation
                if(corr < min_d):
                    min_d = corr
                    corr_p = pat
        img2.append(corr_p)
        img2.append(corr_p)   #appending matching window into a list

    temp2 = 0
    temp1 = temp1 + 5

print("Intensity window-based matching patches", img1, img2)
```

Output Images:







Question 3: Comparison between Dense SIFT and Intensity window based approach

- Dense SIFT will capture a lot of redundant info in an image but this redundant data can be removed in a learning phase. In intensity window-based method also lot of irrelevancy info is there.
- In Dense SIFT to do the feature matching in two image windows, we can use any feature matching function like Brute-Force Matching with ORB Descriptors, FLANN based Matcher...etc. But in intensity window-based method we should calculate the correlation. Based on correlation measure will match the features in two images.
- Calculating correlation measure taking more time but the FlannBasedMatcher, BFMatcher taking less time.

- Comparatively Dense SIFT giving good and accurate results than the intensity window-based matching.

Question – 4: Rectify the pairs of images

Description:

- To simplify the problem of finding matching points between images will use the rectify the pairs of images (Image rectification). It's a transformation process used to project images onto a common image plane.
- Given a pair of stereo images, the intrinsic parameters of each camera, and the extrinsic parameters of the system, R, and T, compute the image transformation that makes epipolar lines collinear and parallel to horizontal axis.

Algorithm:

- Read the stereo pair image using **imread()**. And divide the images into parts. One image is left and another right image.
- Convert these two parts RGB to Gray().
- Next step is to finding the keypoints and descriptors. To find out those parameters using SIFT. So that creating **SIFT** object, **xfeatures2d.SIFT_create().**
- Finding the Keypoints and Descriptors for both left and right images by using **sift.detectAndCompute()** function.
- Now, need to find out the matching features in both images. For that using FLANN based matcher. Actually Brute Force return only a list of single objects, we cannot iterate them, so using FLANN. For this considering the (parameters i.e., tree = 50, K-NN = 2, ratio = 0.8)
- As per Lowe's paper applying ratio test, finding the best matching points in both images.
- After finding the best matching points, applying these points to a function **cv2.findFundamentalMat().** It returns the fundamental matrix.
- By using this fundamental matrix, we need to find out the rectification matrix for both left and right images. For this, first reshape the matching points, give this points to a function **cv2.stereoRectifyUncalibrated().** It returns rectification matrix for both images.
- Next step, wrap the input image with rectification matrix. It gives the output is a rectified image.

Code:

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

```
img = cv2.imread('Stereo_Pair3.jpg',0)                    #Reding Image


#Dividing the image
w, h = int(img.shape[0]), int(img.shape[1]/2)

img1 = img[:w][:,:h]
img2 = img[:w][:,h:]

sift = cv2.xfeatures2d.SIFT_create()

#Find the keypoints and descriptors using SIFT
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)

#Finding the matches

FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50)
flann = cv2.FlannBasedMatcher(index_params,search_params)
matches = flann.knnMatch(des1,des2,k=2)
pts1, pts2 = [ ], [ ]

#Based on Lowe's paper, applying ratio test
for i,(m,n) in enumerate(matches):
    if m.distance < 0.8*n.distance:
        pts2.append(kp2[m.trainIdx].pt)
        pts1.append(kp1[m.queryIdx].pt)
pts1 = np.array(pts1)
pts2 = np.array(pts2)

#Computing Fundamental Matrix
F,mask= cv2.findFundamentalMat(pts1,pts2,cv2.FM_LMEDS)

pts1 = pts1[:,:][mask.ravel()==1]
pts2 = pts2[:,:][mask.ravel()==1]
pts1 = np.int32(pts1)
pts2 = np.int32(pts2)

#Reshaping the patches
pat1_new = pts1.reshape((pts1.shape[0] * 2, 1))
pat2_new = pts2.reshape((pts2.shape[0] * 2, 1))
```

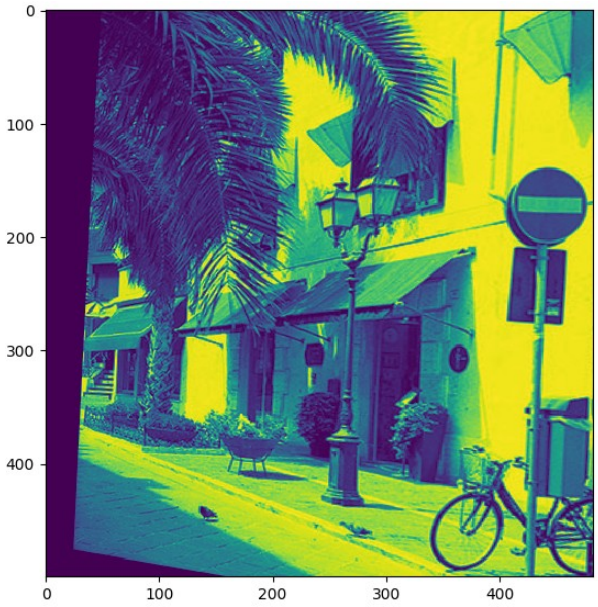
#Applying stereo rectifier to compute rectification matrix
retBool ,rectmat1, rectmat2 =
cv2.stereoRectifyUncalibrated(pat1_new,pat2_new,F,(100, 600))

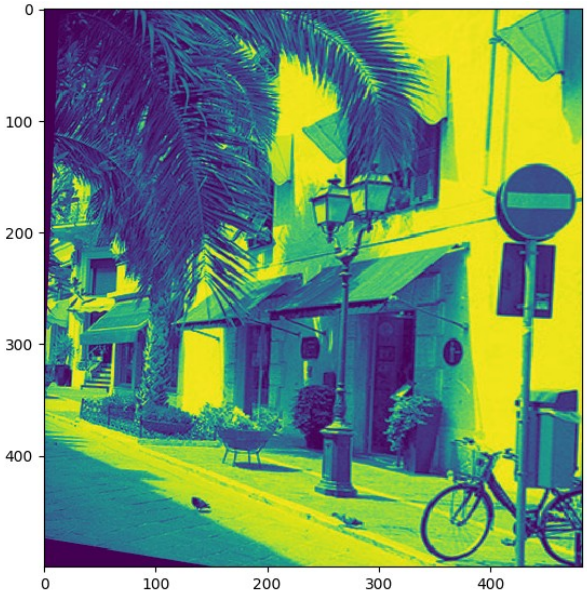
#Applying wrap perspective
per_img1 = cv2.warpPerspective(dst1,rectmat1,dst1.shape)
per_img2 = cv2.warpPerspective(dst2,rectmat2,dst2.shape)

#Displaying rectification images
plt.imshow(per_img1), plt.show()
plt.imshow(per_img2), plt.show()

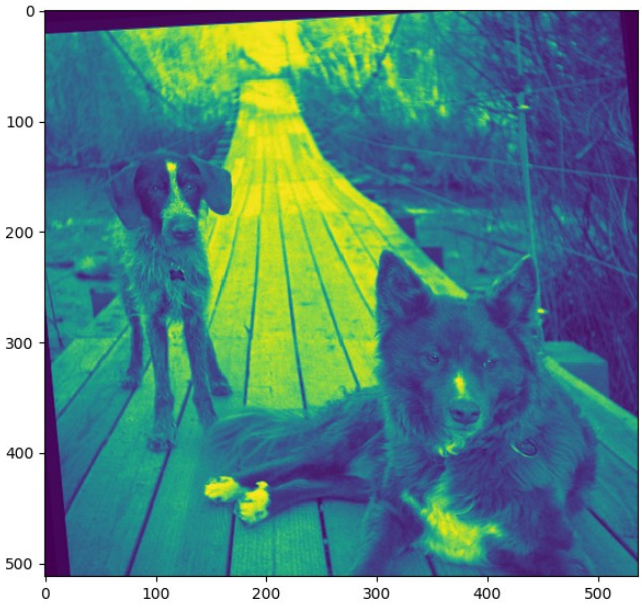
Output Images:

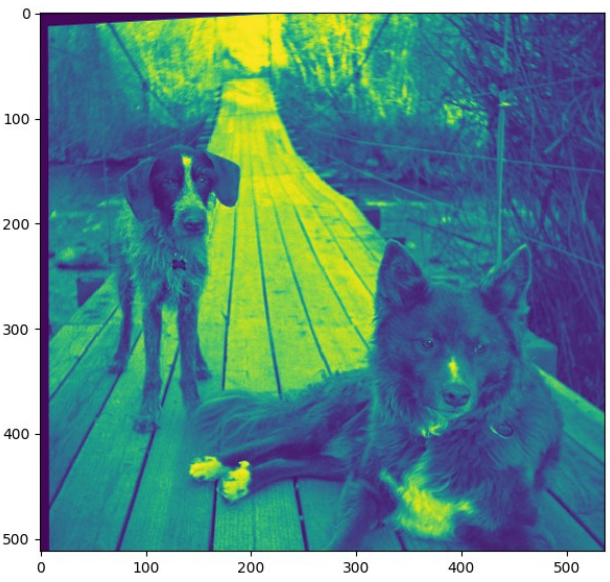
- **Input Image1:**



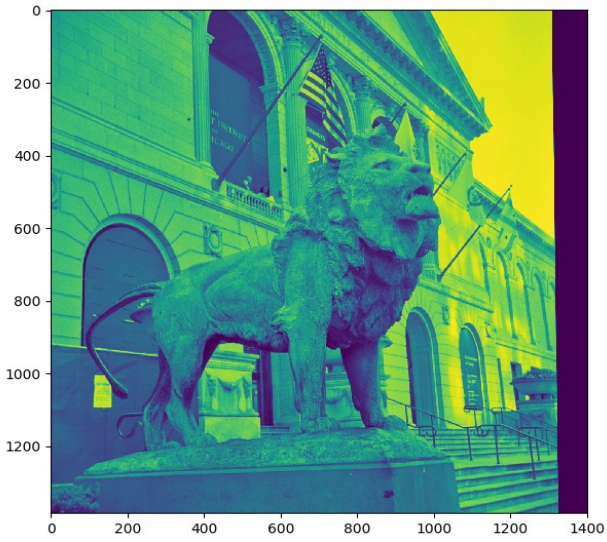


- **Input Image2:**





- **Input Image3**



Applying these images on first two questions, the output: