Report

March 4, 2019

1 Assignment 3

1.0.1 Roll no: 20161170 Name: Shubh Maheshwari

1.1 Stereo Calibration

```
In [1]: # Imports
        import cv2
        import numpy as np
        import matplotlib.pyplot as  plt

        %matplotlib inline

In [2]: # Helper functions
        def display_images(img_list, shape,fig_size=(8,8),is_gray=None):
            """
                Display multple images using matplotlib
                @param img_list:=> mxn matrix of images to be displayed
                @param shape:=> mxn shape
                @param is_gray:=> mxn matrix, is the i,j th the image grayscaled

                return None
            """

            if is_gray is None:
                is_gray = np.zeros(shape)

            m,n = shape
            fig = plt.figure(figsize=fig_size)

            for i in range(m):
                for j in range(n):
                    ax = fig.add_subplot(m,n,i*n + j+1)
                    if is_gray[i,j] == 1:
                        ax.imshow(img_list[i][j],cmap='gray')
                    else:
                        img_list[i][j] = cv2.resize(img_list[i][j],(200,200))
                        ax.imshow(img_list[i][j])
                    ax.axis('off')
            plt.show()
            return
```

2 Question 1

Using Dense Sift key point descriptors do matching on a pair of images

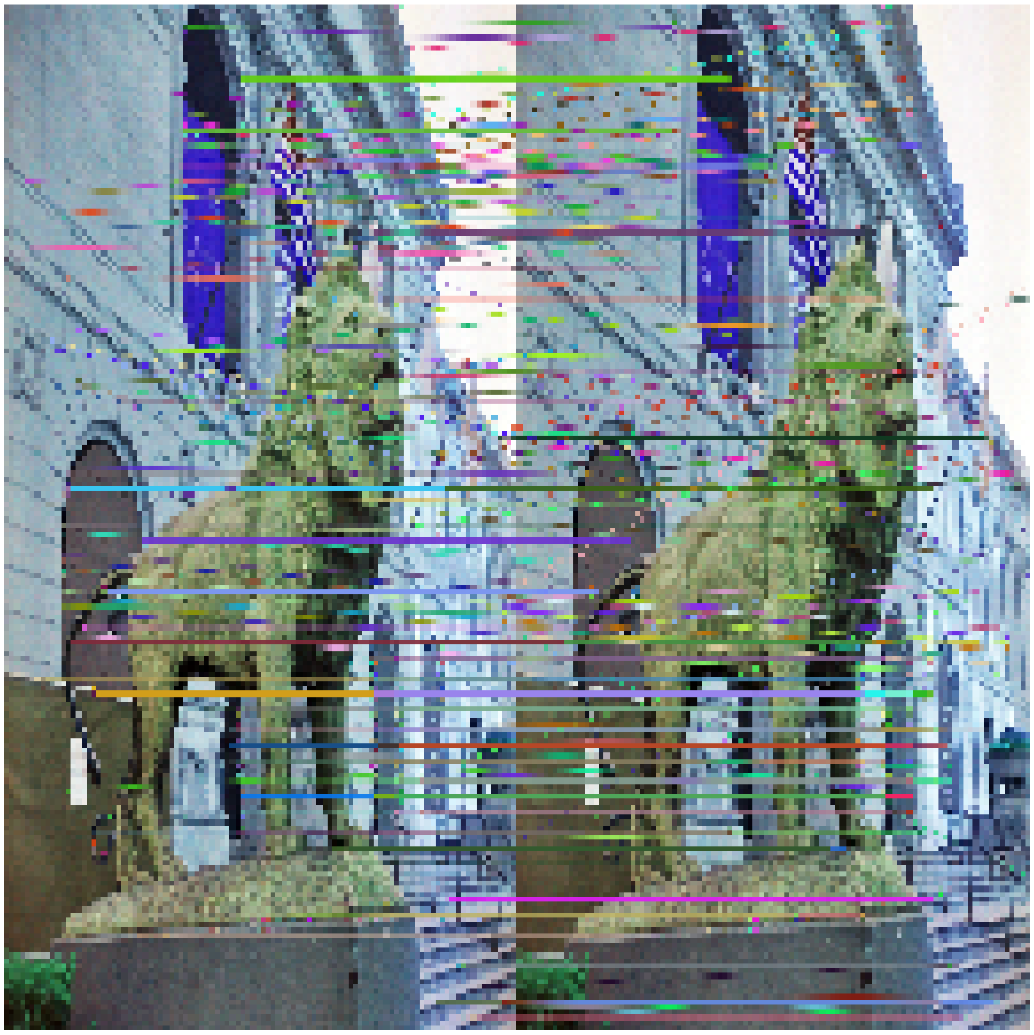
```
In [3]: def dense_sift_matching(img1,img2,min_match_cnt=500):
        #     Display the images
            display_images([[img1,img2]],shape=(1,2))

        #     Initiliaze CV2 SIFT
            sift = cv2.xfeatures2d.SIFT_create()

        # Define our keypoints

            h,w,c = img1.shape
            kp=[]
            for i in range(1,h,10):
                for j in range(1,w,10):
                    kp.append(cv2.KeyPoint(i, j, 3))
            print("Checking Keypoinys",len(kp))


            #     Taking gray scale images for SIFT
            gray_im1 = cv2.cvtColor(img1,cv2.COLOR_RGB2GRAY)
            kp1,des1 = sift.compute(gray_im1,kp)

            gray_im2 = cv2.cvtColor(img2,cv2.COLOR_RGB2GRAY)
            kp2,des2 = sift.compute(gray_im2,kp)

            # BFMatcher with default params
            bf = cv2.BFMatcher(cv2.NORM_L1, crossCheck=True)
            matches = bf.match(des1,des2)
            matches = sorted(matches, key = lambda x:x.distance)[0:min_match_cnt]
            print("Found Matches:",len(matches))


            draw_params = dict(matchesMask=None,
                               singlePointColor=None,
                               flags=2)

            dis_im = cv2.drawMatches(img1,kp1,img1,kp1,matches,None,**draw_params)
            display_images([[dis_im]],shape=(1,1),fig_size=(8,16))
            return kp1,des1,kp2,des2,matches,dis_im
```

3 Desne SIFT Matching

```
In [4]: # Case 1
        img = cv2.imread('./Stereo Images/Stereo_Pair1.jpg')
        h,w,c = img.shape
        img1 = img[:,0:w//2,:]
        img2 = img[:,w//2:w,:]
        sift_param = dense_sift_matching(img1,img2)
        cv2.imwrite('./results/Sift_Pair1.jpg',sift_param[-1])


        # Case 2
        img = cv2.imread('./Stereo Images/Stereo_Pair2.jpg')
        h,w,c = img.shape
        img1 = img[:,0:w//2,:]
        img2 = img[:,w//2:w,:]
        sift_param = dense_sift_matching(img1,img2)
        cv2.imwrite('./results/Sift_Pair2.jpg',sift_param[-1])


        # Case 3
        img = cv2.imread('./Stereo Images/Stereo_Pair3.jpg')
        h,w,c = img.shape
        img1 = img[:,0:w//2,:]
        img2 = img[:,w//2:w,:]
        sift_param = dense_sift_matching(img1,img2)
        cv2.imwrite('./results/Sift_Pair3.jpg',sift_param[-1])
```

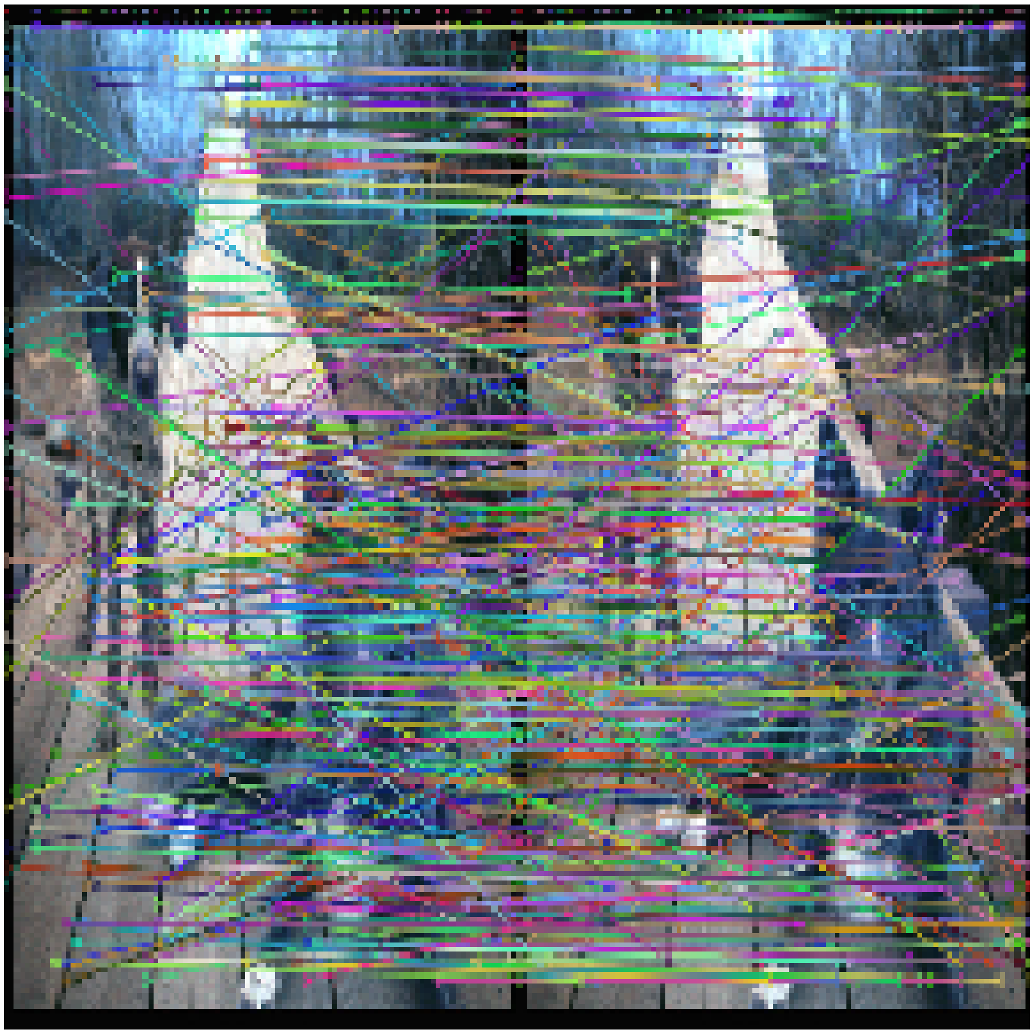
```
Checking Keypoinys 19460
```

```
Found Matches: 500
```



```
Checking Keypoinys 2450
Found Matches: 500
```

```
Checking Keypoinys 2808
Found Matches: 500
```



```
Out[4]: True
```

4 Question 2

Correlation Matching using patches

```
In [5]: # Define Correlation as dot product(normalized)
        def corr(v1,v2):
            return v1.T.dot(v2)/(np.sqrt(v1.T.dot(v1))*np.sqrt(v2.T.dot(v2)))
```

```
In [6]: def correlation_matching(img1,img2,window_size=128,stride=128,thresh = 0.01):
            h1,w1,c = img1.shape
            h2,w2,c = img2.shape
```

```
#     Pass through all the patches in img1 and find patch in img2 with least
    best_matches = []
    for y1 in range(0,h1-window_size,stride):
        for x1 in range(0,w1-window_size,stride):
            least_dis = 1.0
            for y2 in range(0,h2-window_size,stride):
                for x2 in range(0,w2-window_size,stride):
                    v1 = img1[y1:y1+window_size, x1:x1+window_size,:].flatten()
                    v2 = img2[y2:y2+window_size, x2:x2+window_size,:].flatten()
                    dis = corr(v1,v2)
                    if least_dis > dis:
                        least_dis = dis
                        least_coord = [x1,y1,x2,y2,dis]
            best_matches.append(least_coord)
    return best_matches
```

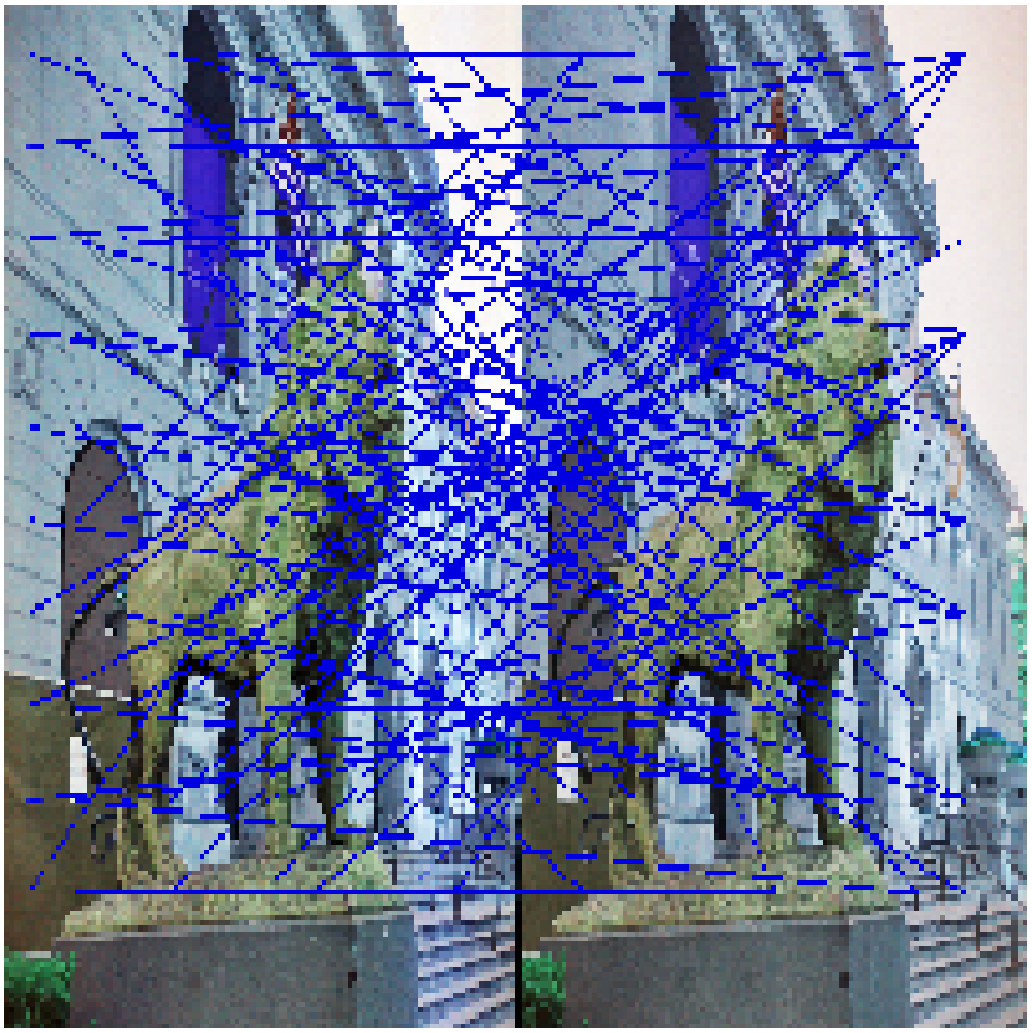
```
In [7]: def draw_matches(img,matches,window_size=128):
            h,w,c = img.shape
            for match in matches:
                pt1 = (match[1]+window_size//2,match[0]+window_size//2)
                pt2 = (match[3]+window_size//2+w//2,match[2]+ window_size//2)
                line_img = cv2.line(img,pt1,pt2,(0,0,225),3)

            display_images([[line_img]],shape=(1,1))
            return line_img
```

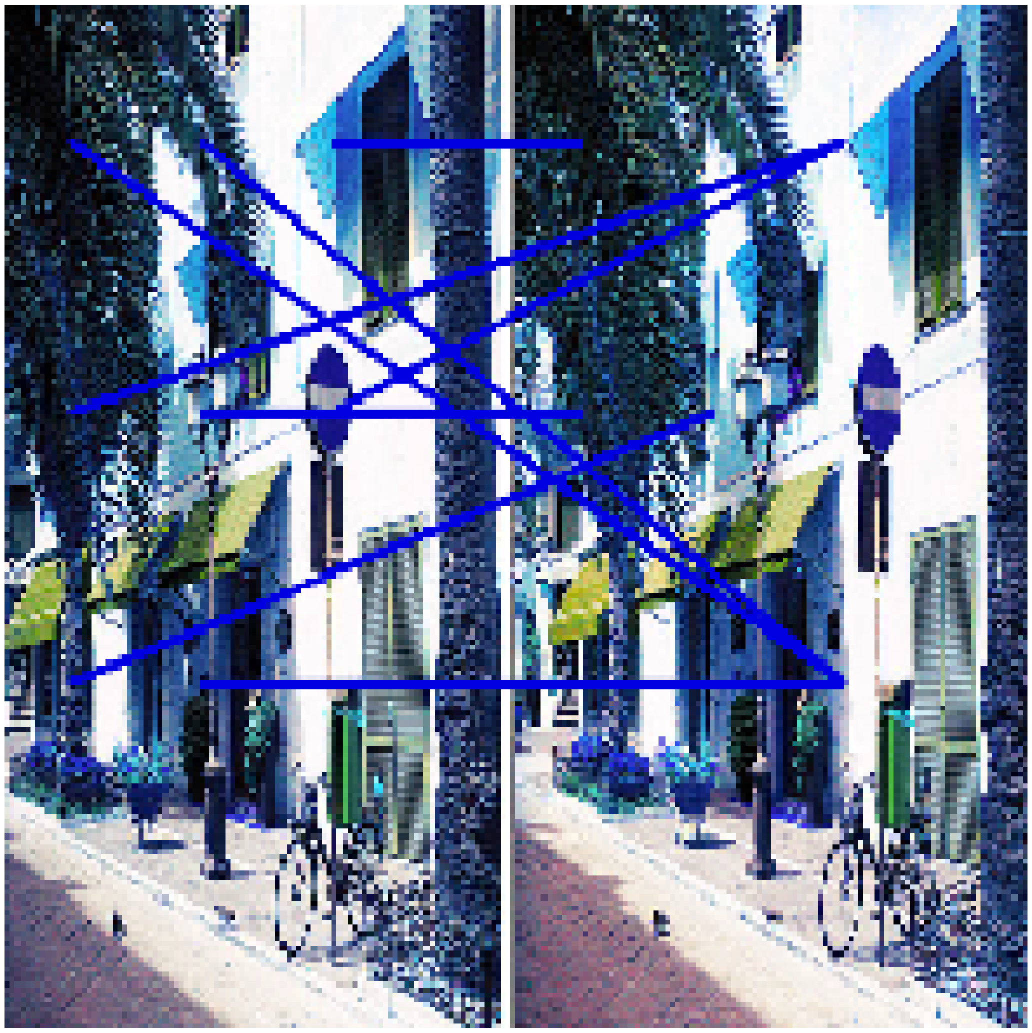
5 Intensity Based Matching

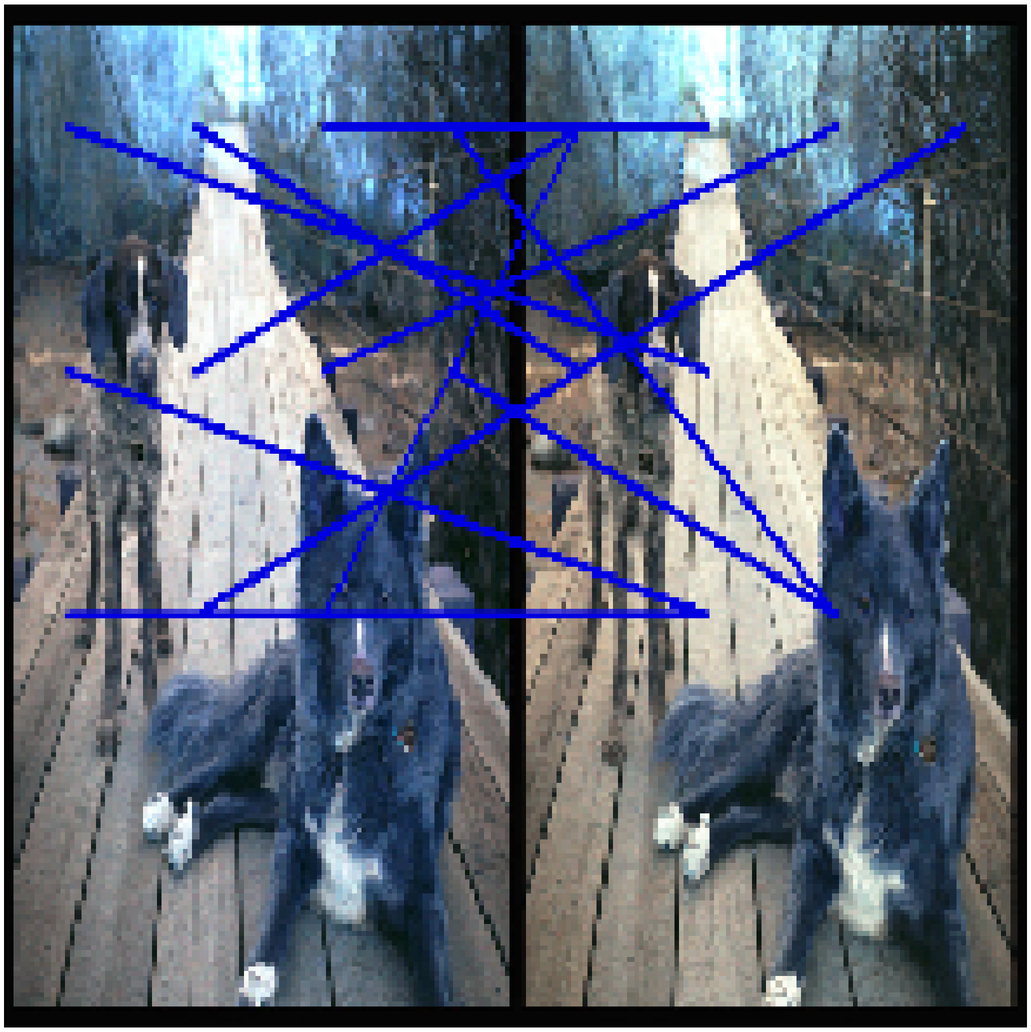
```
In [8]: # Case 1
        img = cv2.imread('./Stereo Images/Stereo_Pair1.jpg')
        h,w,c = img.shape
        img1 = img[:,0:w//2,:]
        img2 = img[:,w//2:w,:]
        matches = correlation_matching(img1,img2)
        dis_im = draw_matches(img,matches)
        cv2.imwrite('./results/Dense_Pair1.jpg',dis_im)

        # Case 2
        img = cv2.imread('./Stereo Images/Stereo_Pair2.jpg')
        h,w,c = img.shape
        img1 = img[:,0:w//2,:]
        img2 = img[:,w//2:w,:]
        matches = correlation_matching(img1,img2)
        dis_im = draw_matches(img,matches)
        cv2.imwrite('./results/Dense_Pair2.jpg',dis_im)
```

```
# Case 3
img = cv2.imread('./Stereo Images/Stereo_Pair3.jpg')
h,w,c = img.shape
img1 = img[:,0:w//2,:]
img2 = img[:,w//2:w,:]
matches = correlation_matching(img1,img2)
dis_im = draw_matches(img,matches)
cv2.imwrite('./results/Dense_Pair3.jpg',dis_im)
```

```
/usr/local/lib/python3.5/dist-packages/ipykernel/__main__.py:3: RuntimeWarning: divide by zero e
  app.launch_new_instance()
```







```
Out[8]: True
```

6 Question 3

We can clearly notice matching using SIFT discriptors gives much better reslts than naive correl
matching. High error in correlation based method is because a scene can have many similiar patch
In case of stereo cameraa, their is a small offset in the camera location. This leads to the sam
having different pixel values for a patch trying to represent the same scene.

7 Question 4

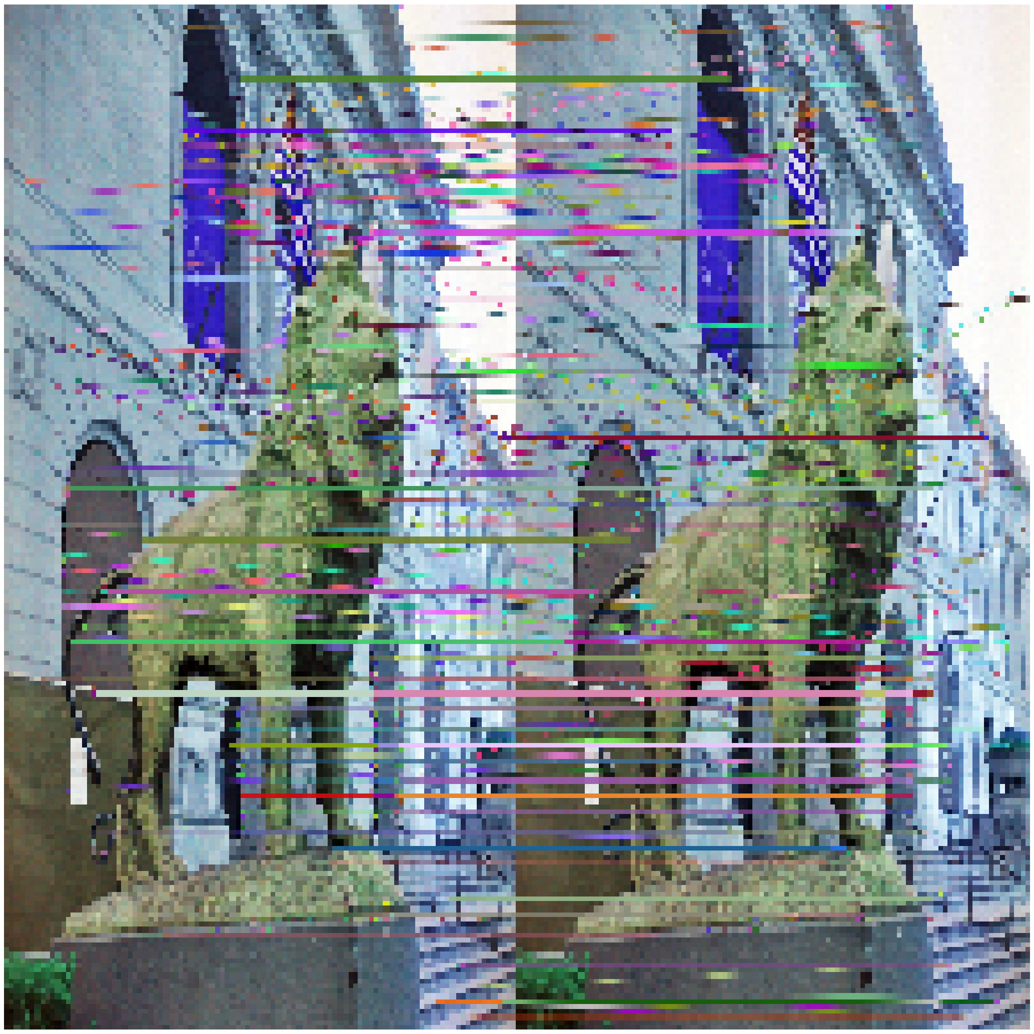
Stereo Rectification

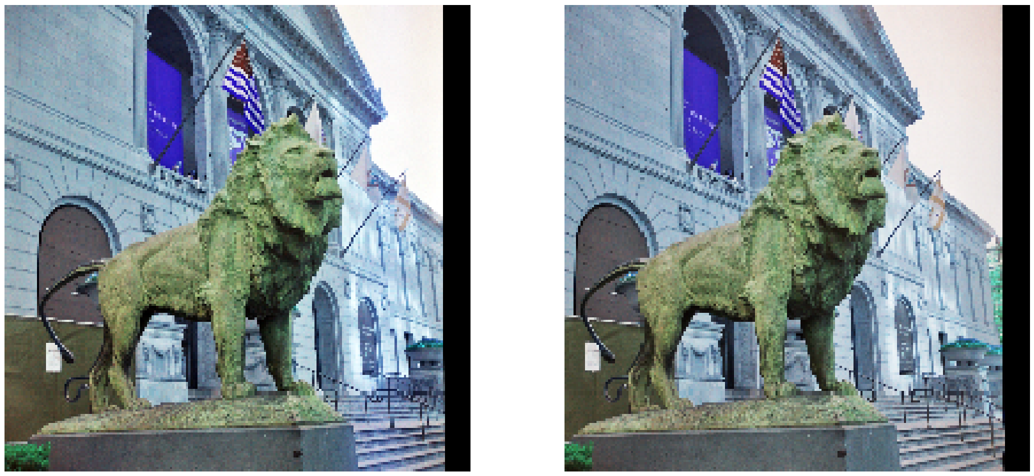
```
In [9]: def stereo_rectification(img1,img2,sift_params,match_th =0.8):
            kp1,des1,kp2,des2,_,_ = sift_params

            index_params = dict(algorithm = 1, trees = 5)
            search_params = dict(checks=50)
            flann = cv2.FlannBasedMatcher(index_params,search_params)
            matches = flann.knnMatch(des1,des2,k=2)

        #     Store best matches
            pts1 = []
            pts2 = []
            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.float32(pts1)
            pts2 = np.float32(pts2)

            F, mask = cv2.findFundamentalMat(pts1,pts2,cv2.RANSAC)

            # We select only inlier points
            pts1 = pts1[mask.ravel()==1]
            pts2 = pts2[mask.ravel()==1]
            img_size = img1.shape[0:2]
            p,H1,H2=cv2.stereoRectifyUncalibrated(pts1, pts2, F, img_size)

            H3= H1.dot(H2)
            img1_corrected = cv2.warpPerspective(img1, H1, img_size)
            img2_corrected = cv2.warpPerspective(img2, H3, img_size)

            return img1_corrected, img2_corrected
```

8 Stereo Rectifiacton

```
In [11]: # Case 1
         img = cv2.imread('./Stereo Images/Stereo_Pair1.jpg')
         h,w,c = img.shape
         img1 = img[:,0:w//2,:]
         img2 = img[:,w//2:w,:]
         sift_param = dense_sift_matching(img1,img2)
         rect_img1, rect_img2 = stereo_rectification(img1,img2,sift_param,match_th =0.8)
         display_images([[rect_img1,rect_img2]],shape=(1,2))
         cv2.imwrite('./results/Stereo_Rect1_1.jpg',rect_img1)
         cv2.imwrite('./results/Stereo_Rect1_2.jpg',rect_img2)
```

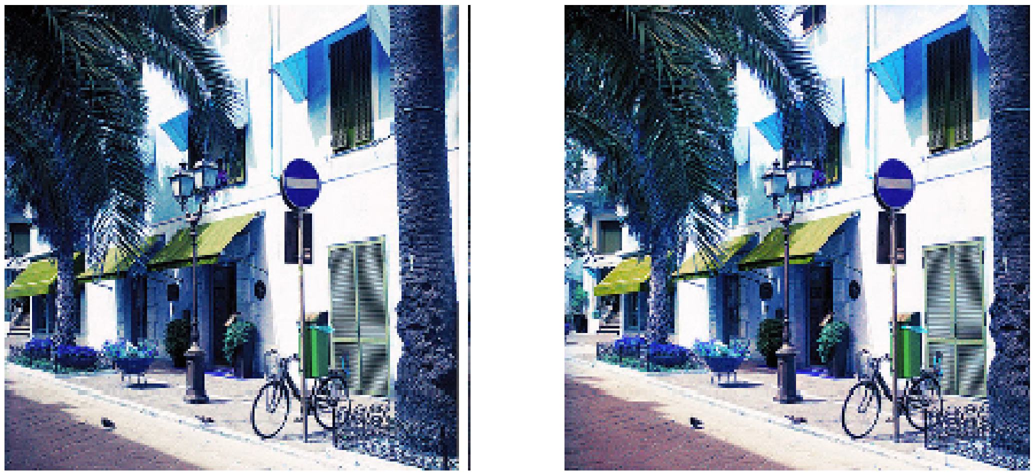
```
Checking Keypoinys 19460
Found Matches: 500
```



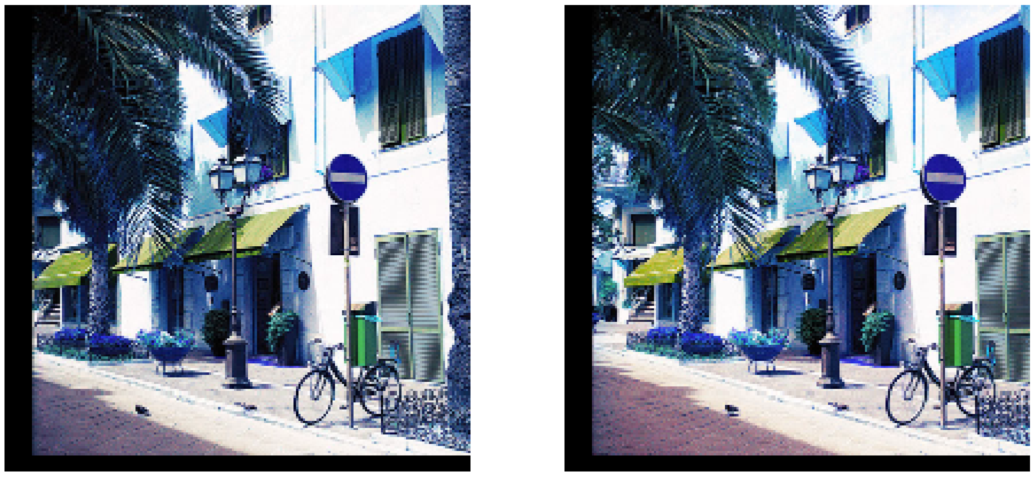


```
Out[11]: True
```

```
In [12]: # Case 2
         img = cv2.imread('./Stereo Images/Stereo_Pair2.jpg')
         h,w,c = img.shape
         img1 = img[:,0:w//2,:]
         img2 = img[:,w//2:w,:]
         sift_param = dense_sift_matching(img1,img2)
         rect_img1, rect_img2 = stereo_rectification(img1,img2,sift_param,match_th =0.8)
         display_images([[rect_img1,rect_img2]],shape=(1,2))
         cv2.imwrite('./results/Stereo_Rect2_1.jpg',rect_img1)
         cv2.imwrite('./results/Stereo_Rect2_2.jpg',rect_img2)
```

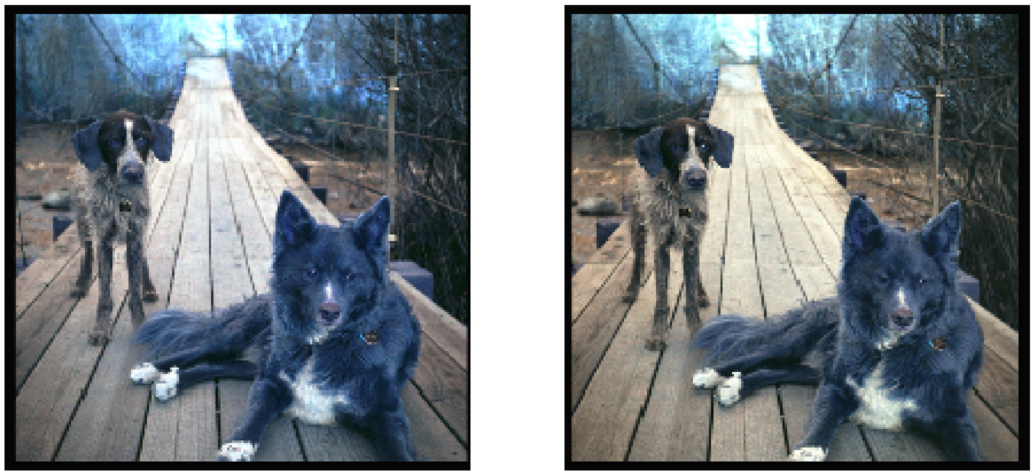


```
Checking Keypoinys 2450
Found Matches: 500
```



```
Out[12]: True
```

```
In [13]: # Case 3
         img = cv2.imread('./Stereo Images/Stereo_Pair3.jpg')
         h,w,c = img.shape
         img1 = img[:,0:w//2,:]
         img2 = img[:,w//2:w,:]
         sift_param = dense_sift_matching(img1,img2)
         rect_img1, rect_img2 = stereo_rectification(img1,img2,sift_param,match_th =0.8)
         display_images([[rect_img1,rect_img2]],shape=(1,2))
         cv2.imwrite('./results/Stereo_Rect3_1.jpg',rect_img1)
         cv2.imwrite('./results/Stereo_Rect3_2.jpg',rect_img2)
```



```
Checking Keypoinys 2808
Found Matches: 500
```