Computer Vision Assignment 2

Camera Calibration

By- Apoorva Srivastava (2019702014)

Image Mosaicing:

Image Mosaicing is the process of stitching n images together to create a panaromic view.The images are required to be taken by only providing Rotation to the Camera and no translation.Images must also have some region overlapping with the next image and should be provided in left to right rder for the following algorithm.

Algorithm:

1. Compute the sift-key points and descriptors for left and right images.
2. Compute distances between every descriptor in one image and every descriptor in the other image.
3. Select the top best matches for each descriptor of an image.
4. Run RANSAC to estimate homography.
5. Warp to align for stitching.
6. Finally stitch them together after performing required trimming.
7. Repeat this for N images so that N images can be stitched together.

Implementation:

1. The first 3 steps of the algorithm are performed using in-built function for SIFT and obtained keypoints are compared on the basis of descriptors and the best matches are chosen.

2. Homography is calculated by applying RANSAC and DLT on 2D-2D correspondences obtained in Step1.

Matrix Form:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \sim \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Equations:

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

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

$$\begin{array}{lcc} & \textbf{2N x 8} & \textbf{8 x 1} \quad \textbf{2N x 1} \\ \textbf{Point 1} & \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \end{bmatrix} & \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4' \\ y_4' \end{bmatrix} \\ \textbf{Point 2} & \begin{bmatrix} x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_2' \end{bmatrix} & \\ \textbf{Point 3} & \begin{bmatrix} x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_3' & -y_3y_3' \end{bmatrix} & \\ \textbf{Point 4} & \begin{bmatrix} x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y_4' & -y_4y_4' \end{bmatrix} & \\ \textbf{additional points} & \vdots & \vdots \end{array}$$

h_{33}=1 due to homogenous coordinates. So there are 8 unknowns and by applying SVD on the obtained Matrix we get the unknowns. Then using RANSAC we choose the best fit values for the homography matrix.

3. Warping of the image is done using the inbuilt function cv2.WarpPerspective() by providing the right image and the Homography estimated in the step2.

4. To bring back the Stitched image back to the rectangular shape a function named trim() has been designed to trim the undesired black region.

5. For generalizing the above method for N images, the above steps are repeated in loop and

Code:

```
#Stitch is a function that stitches 2 images together,where 1st input argument should be left image
#and the second argument should be right image
def Stitch(imgL,imgR,iteration):
        p1=[]
        p2=[]
        img1=cv2.cvtColor(imgL,cv2.COLOR_BGR2GRAY)
        img2=cv2.cvtColor(imgR,cv2.COLOR_BGR2GRAY)
        sift=cv2.xfeatures2d.SIFT_create()
        kp1,des1 =sift.detectAndCompute(img1,None)
        kp2,des2 =sift.detectAndCompute(img2,None)
        cv2.imwrite('image_mosaicing/images6/left image_keypoints'+str(iteration)+'.jpg',cv2.drawKeypoints(img1,kp1,None))
        cv2.imwrite('image_mosaicing/images6/right image_keypoints'+str(iteration)+'.jpg',cv2.drawKeypoints(img2,kp2,None))
        match = cv2.BFMatcher()
        matches = match.knnMatch(des1,des2,k=2)
        good = []
        for m,n in matches:
            if m.distance < 0.5*n.distance:
                good.append(m)

        if len(good)>10:
            draw_params = dict(matchColor = (0,0,255), # draw matches in red color
                               singlePointColor = None,
                               flags = 2)

            img3 = cv2.drawMatches(img1,kp1,img2,kp2,good,None,**draw_params)
            cv2.imwrite('image_mosaicing/images6/drawMatches' +str(iteration) +'.jpg', img3)

            for i in range(len(good)):
                idx1=good[i].queryIdx
                idx2=good[i].trainIdx
                p1.append(kp1[idx1].pt)
                p2.append(kp2[idx2].pt)
            p1=np.asarray(p1)
            p2=np.asarray(p2)
            H=RANSAC(p1,p2)
            dst = cv2.warpPerspective(imgR,H,(imgR.shape[1]+imgL.shape[1], imgL.shape[0]+imgR.shape[0]))
            cv2.imwrite('image_mosaicing/images6/warped' +str(iteration) +'.jpg', dst)
            dst[0:imgL.shape[0], 0:imgL.shape[1]] = imgL
            crop_im=trim(dst)
            return(crop_im)
        else:
            print('Not enough matches found')
            return(None)
```

```
#This function crops the intermdiate results
def trim(img):
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    _,thresh = cv2.threshold(gray,2,255,cv2.THRESH_BINARY)
    contours = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contours = contours[0] if len(contours) == 2 else contours[1]
    for i in contours:
        x,y,w,h = cv2.boundingRect(i)
        crop = img[y:y+h,x:x+w]
    return(crop)
```

```
#RANSAC function calculates the Homography between the 2 images given the correspondence
def RANSAC(p1,p2):
    from random import sample
    RANSAC_iter=100
    num_points =[i for i in range(p1.shape[0])]
    diff_norm=np.empty((RANSAC_iter,1))
    diff=np.empty((p1.shape[0],1))
    M = np.empty((8,9))
    H12=np.empty((RANSAC_iter,3,3))
    for k in range(RANSAC_iter):
        four_p = sample(num_points, 4)
        j=0
        for i in four_p:
            # u2,v2,1,0,0,0,-u1u2,-u1v2,-u1 :u2=p2x,v2=p2y,u1=p1x,v1=p1y
            # 0,0,0,u2,v2,1,-v1u2,-v1v2,-v1
            M[j,:]=[p2[i,0],p2[i,1],1,0,0,0,-p1[i,0]*p2[i,0],-p1[i,0]*p2[i,1],-p1[i,0]]
            M[j+1,:]=[0,0,0,p2[i,0],p2[i,1],1,-p1[i,1]*p2[i,0],-p1[i,1]*p2[i,1],-p1[i,1]]
            j+=2
        (U, S, V) = np.linalg.svd(M, full_matrices=True)
        f = V[-1,:]
        H12[k] =np.reshape(f,(3,3))
        for i in range(p1.shape[0]):
                p2_=np.array([p2[i,0],p2[i,1],1])
                p1_=np.matmul(H12[k],p2_)
                p1_=p1_/(p1_[2]+0.000000001)
                diff[i]=np.linalg.norm(p1[i]-p1_[0:2])
        diff_norm[k]=np.sum(diff)/p1.shape[0]
    idx_min= np.argmin(diff_norm)
    H=H12[idx_min]
    H=H/H[2,2]
    print(H)
    print('MinError in H',diff_norm[idx_min])
    return(H)
```

```
#The Main Code
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os
from os import listdir
from os.path import isfile, join
import glob
import sys
#all images that are to be stitched are kept in separate folder together and output of each
#stitching iteration are stored back in the same folder
#The images must be in the order of (TOP to BOTTOM) from (LEFT MOST to RIGHT MOST)
mypath='C:\\Users\\Asus\\Computer Vision\\Assignment2\\image_mosaicing\\images6'
onlyfiles = [ f for f in listdir(mypath) if isfile(join(mypath,f)) ]
images = np.empty(len(onlyfiles), dtype=object)
out = np.empty(len(onlyfiles)-1, dtype=object)
for n in range(len(onlyfiles)):
    images[n] = cv2.imread( join(mypath,onlyfiles[n]) )
j=0
if len(images)%2==0:       #if there are even number of images
    for i in range(0,len(images),2):
        out[j]=Stitch(images[i],images[i+1],i)
        if out[j].any()==None:
            print('Noisy image present')
            sys.exit()
        cv2.imwrite('image_mosaicing/images6/output_im'+str(j)+'.jpg',out[j])
        j+=1
    if(out.shape[0]>1):
        out[j]=Stitch(out[0],out[1],j)
        cv2.imwrite('image_mosaicing/images6/output_im'+str(j)+'.jpg',out[j])
        if(j+1<out.shape[0]):
            out[j+1]=Stitch(out[j],out[j-2],j+1)
            cv2.imwrite('image_mosaicing/images6/output_im'+str(j+1)+'.jpg',out[j+1])

else:                              #if there are odd number of images
    for i in range(0,len(images)-1,2):
        out[j]=Stitch(images[i],images[i+1],i)
        if out[j].any()==None:
            print('Noisy image present')
            sys.exit()
        cv2.imwrite('image_mosaicing/images6/output_im'+str(j)+'.jpg',out[j])
        j+=1

    if(out.shape[0]>1):
        out[j]=Stitch(out[0],out[1],j)
        cv2.imwrite('image_mosaicing/images6/output_im'+str(j)+'.jpg',out[j])
        if(j+1<out.shape[0]):
            out[j+1]=Stitch(out[j],out[j-2],j+1)
            cv2.imwrite('image_mosaicing/images6/output_im'+str(j+1)+'.jpg',out[j+1])

    out[-1]=Stitch(out[-2],images[-1],out.shape[0])
    cv2.imwrite('image_mosaicing/images6/output_im'+str(j)+'.jpg',out[j])
```

Results:

1) **Input**

Output

All intermediate & final results are shared in the link below:

https://iiitaphydmy.sharepoint.com/:f:/g/personal/apoorva_srivastava_research_iiit_ac_in/EhMDQwvobMdPuJNK4sV8UbwBcfWKsGYLVknX1uvvvMgTEw?e=2PUDXF

2) **Input:**

Output:

The image was obtained after reordering them in the order of of (TOP to BOTTOM) from (LEFT MOST to RIGHT MOST).

All intermediate & final results are shared in the link below:

https://iiitaphyd-my.sharepoint.com/:f:/g/personal/apoorva_srivastava_research_iiit_ac_in/EtMMjMzw5gVDp45OpFjUY8ABbsPIztrGPMoUrz4BV44-Kg?e=S9t0Xi

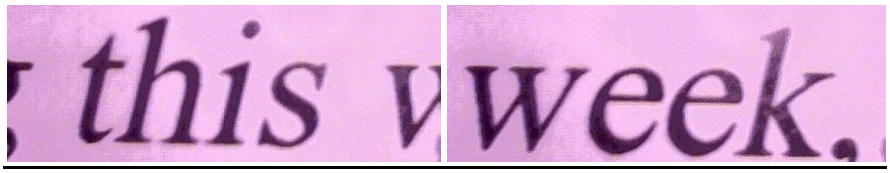
3.Input

Output:

All intermediate & final results are shared in the link below:

https://iiitaphyd-my.sharepoint.com/:f:/g/personal/apoorva_srivastava_research_iiit_ac_in/EjYbxf1S57dGpMr0WkaxD0UBIF7cD8xcBc6QyCBMl2wn2Q?e=TcPAfO

4.Input



Output: As the input images don't have sufficient correspondences found after applying SIFT hence they could not be stitched together.

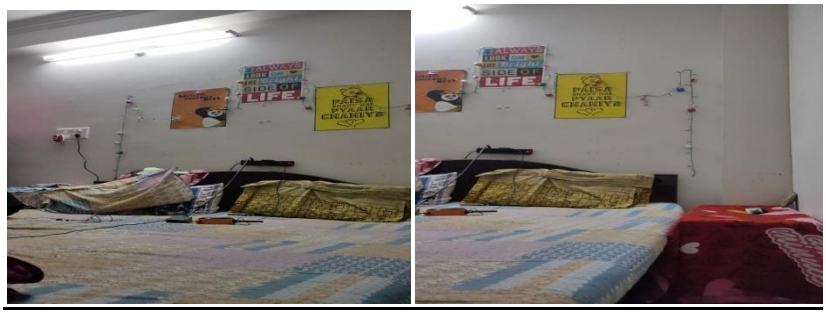
5.Input

Output:

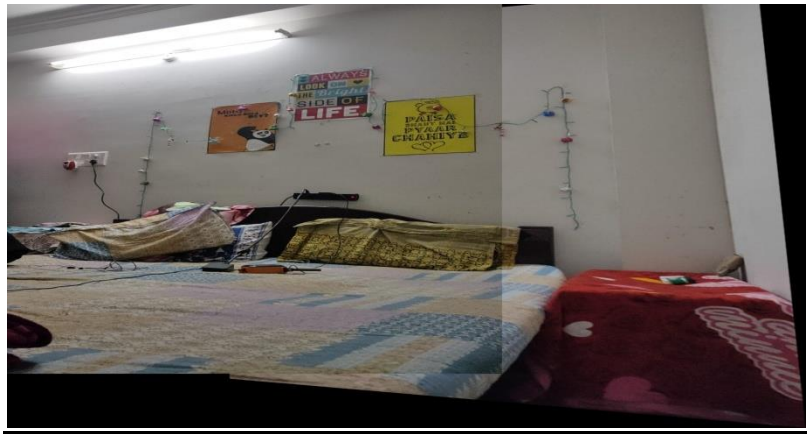
All intermediate & final results are shared in the link below:

https://iiitaphyd-my.sharepoint.com/:f:/g/personal/apoorva_srivastava_research_iiit_ac_in/EsiRrhqEyQ9Is_TcomS9FvABFXJbBQSn51kL4oiAyuO2OA?e=iLYpb1

6.Input: Trying the code on self clicked images:



Output:



All intermediate & final results are shared in the link below:

https://iiitaphyd-my.sharepoint.com/:f:/g/personal/apoorva_srivastava_research_iiit_ac_in/EjL0Sbf4-11DmdKi11ix8yoBV9uvWJbKHy9fsx8dXzzsDg?e=maxGhb

Stereo Matching:

Computer stereo vision is the extraction of 3D information from digital images. By comparing information about a scene from two vantage points, 3D information can be extracted by examining the relative positions of objects in the two panels.

In traditional stereo vision, two cameras, displaced horizontally from one another are used to obtain two differing views on a scene, in a manner similar to human binocular vision. By comparing these two images, the relative depth information can be obtained in the form of a disparity map, which encodes the difference in horizontal coordinates of corresponding image points. The values in this disparity map are inversely proportional to the scene depth at the corresponding pixel location.

In a computer vision system, several pre-processing steps are required:

1. The image must first be undistorted, such that barrel distortion and tangential distortion are removed. This ensures that the observed image matches the projection of an ideal pinhole camera.
2. The image must be projected back to a common plane to allow comparison of the image pairs, known as image rectification.
3. An information measure which compares the two images is minimized. This gives the best estimate of the position of features in the two images, and creates a disparity map.

Stereo Setup:

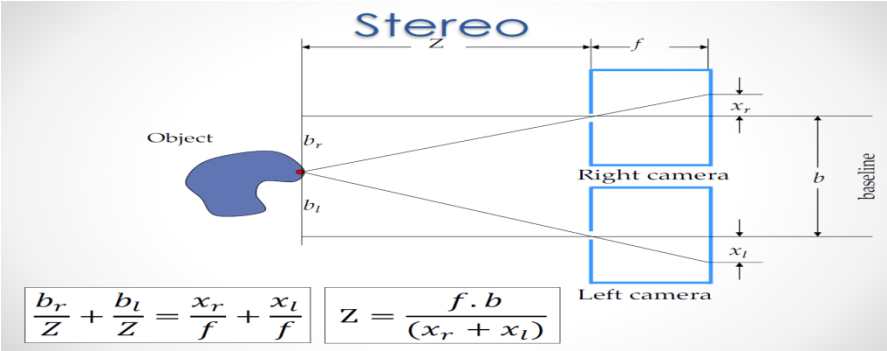
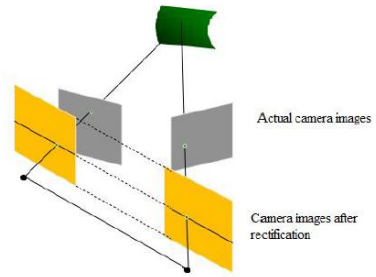
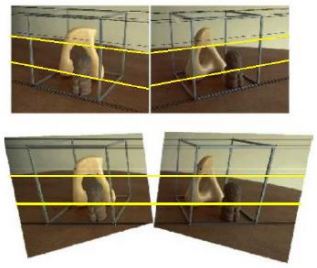


Image Rectification:





Implementation:

There were multiple methods employed to obtain the correspondences between the Stereo pairs given:

All stereo input images and intermediate results are at following link: https://iiitaphyd-my.sharepoint.com/:f:/g/personal/apoorva_srivastava_research_iiit_ac_in/EsY475a75OtMsR48osLQRXUB5cYWPn6Un1K4B8kTJCrGgQ?e=szjJUh

1. **Intensity Window-based correlation on un-rectified images:**

   Algorithm:

   1. From the left image some Good Feature points were extracted using cv2.GoodFeaturestotrack().
   2. Around each such point a 3x3 window was considered which was individually matched with every 3x3 window of the second image for every pixel using the normalized correlation-

      $$\text{Normalized correlation} = \frac{v_1^T v_2}{\sqrt{v_1^T v_1}\sqrt{v_2^T v_2}}$$

      where v1 and v2 are intensity vectors.
   3. The maximum correlation value pixel was chosen as the correspondences.

Code:

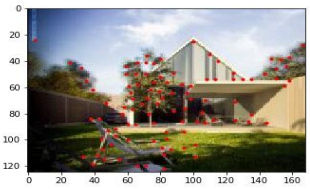
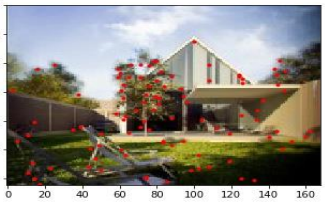
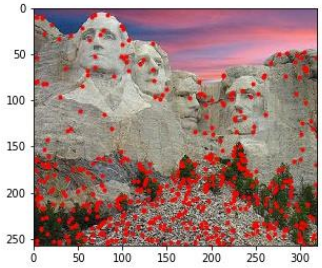
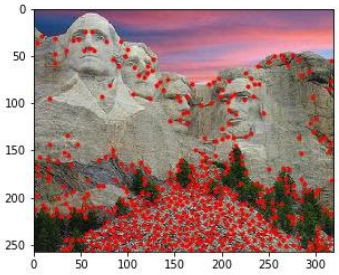
```python
def zero_pad(arr,kernel_half):
    x,y=np.shape(arr)
    pad_h=np.zeros((arr.shape[0],kernel_half))
    arr=np.hstack((pad_h,arr))
    arr=np.hstack((arr,pad_h))
    pad_v=np.zeros((kernel_half,arr.shape[1]))
    arr=np.vstack((pad_v,arr))
    arr=np.vstack((arr,pad_v))
    return arr
```

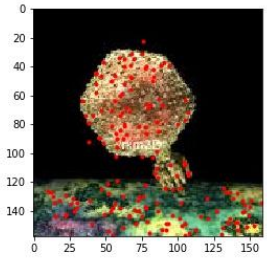
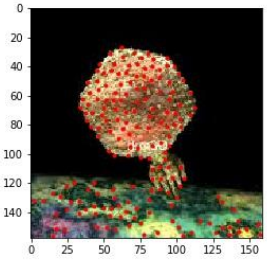
```python
import numpy as np
import cv2

imgL=cv2.imread('stereo_images/1L.jpg')
imgR=cv2.imread('stereo_images/1R.jpg')
imgL_gray=cv2.cvtColor(imgL,cv2.COLOR_BGR2GRAY)
imgR_gray=cv2.cvtColor(imgR,cv2.COLOR_BGR2GRAY)
#print(imgL_gray.shape,imgR_gray.shape)
points = cv2.goodFeaturesToTrack(imgL_gray,500,.1,5)
num_points=len(points)
points=np.reshape(points,(num_points,2))
#print(points)
points_x=points[:,1]
points_y=points[:,0]
arr  =imgL_gray
arr2 =imgR_gray
#print(arr.shape,arr2.shape)
x,y=np.shape(arr2)
matches=np.empty((num_points,4))
kernel=3
#diff=np.empty((3,3))
count=0
kernel_half=int(kernel/2)
arr=zero_pad(arr,kernel_half)
arr2=zero_pad(arr2,kernel_half)
print(arr.shape,arr2.shape)
for (u,v) in zip(points_x,points_y):
```

```
for (u,v) in zip(points_x,points_y):
    temp=arr[int(u)-kernel_half:int(u)+kernel_half+1,int(v)-kernel_half:int(v)+kernel_half+1]
    temp=temp.flatten()
    #temp=np.reshape(temp,(len(temp)))
    #print(temp.shape)
    ssd=[]
    idx_x=[]
    idx_y=[]
    #print('next point')
    for i in range(kernel_half,x+kernel_half,1):
        for j in range(kernel_half,y+kernel_half,1):
            win=arr2[i-kernel_half:i+kernel_half+1,j-kernel_half:j+kernel_half+1]
            #print(win.shape)
            win=win.flatten()
            #win=np.reshape(win,(len(win)))
            #print(win.shape)
            diff=np.dot(temp,win)
            if np.linalg.norm(temp)!=0 and np.linalg.norm(win)!=0:
                diff=diff/np.linalg.norm(temp)
                diff=diff/np.linalg.norm(win)
            elif np.linalg.norm(temp)==0 and np.linalg.norm(win)!=0:
                diff=diff/0.00000001
                diff=diff/np.linalg.norm(win)
            elif np.linalg.norm(temp)!=0 and np.linalg.norm(win)==0:
                diff=diff/0.00000001
                diff=diff/np.linalg.norm(temp)
            elif np.linalg.norm(temp)==0 and np.linalg.norm(win)==0:
                diff=diff/0.00000001
                diff=diff/0.00000001
            ssd.append(diff)
            idx_x.append(i)
            idx_y.append(j)
    index=np.argwhere(ssd==max(ssd))
    matches[count,0]=u
    matches[count,1]=v
    matches[count,2]=idx_x[index[0][0]]
    matches[count,3]=idx_y[index[0][0]]
    count+=1
```

```
import matplotlib.pyplot as plt
# num_points=150
# points = cv2.goodFeaturesToTrack(imgL_gray,num_points,.1,15)
# points=np.reshape(points,(num_points,2))
#print(points)
img1 = plt.imread('stereo_images/1L.jpg')
plt.plot(matches[:,1],matches[:,0],'r.')

plt.imshow(img1)
plt.savefig('stereo_1L.jpg')
plt.show()
img2 = plt.imread('stereo_images/1R.jpg')
plt.plot(matches[:,3],matches[:,2],'r.')

plt.imshow(img2)
plt.savefig('stereo_1R.jpg')
plt.show()
```

Results and Analysis:

All results are shared in the link below: https://iiitaphyd-my.sharepoint.com/:f:/g/personal/apoorva_srivastava_research_iiit_ac_in/EjYQbE5J9ZlIjdmt00rgiaoBAN_DonUy70hHABCl0A2rvg?e=kQSG2o

The Correspondences were not very accurate as intensity is not a very reliable criteria and much depends on the choice of the window also.

2. Intensity Window-based correlation on rectified images:

Algorithm:

1. The images given were rectified using MATLAB Code.
2. From the left Rectified image some Good Feature points were extracted using cv2.GoodFeaturestotrack().
3. Around each such point a 3x3 window was considered which was individually matched with every 3x3 window of the second image for every pixel of the corresponding row of lft ima as theey were rectified using the normalized correlation-

$$\text{Normalized correlation} = \frac{v_1^T v_2}{\sqrt{v_1^T v_1}\sqrt{v_2^T v_2}}$$ where v1 and v2 are intensity vectors.

4. The maximum correlation value pixel was chosen as the correspondences.

Code: Rectification Code:

```
%This program rectifies a pair of stereo images
num = 25;
image_left = imread('stereo_images/3L.jpg');
image_right =imread('stereo_images/3R.jpg');


I1 = rgb2gray(image_left);
I2 = rgb2gray(image_right);
points1 = detectHarrisFeatures(I1);
points2 = detectHarrisFeatures(I2);
[features1,valid_points1] = extractFeatures(I1,points1);
[features2,valid_points2] = extractFeatures(I2,points2);
indexPairs = matchFeatures(features1,features2);
pts1 = valid_points1(indexPairs(:,1),:);
pts2 = valid_points2(indexPairs(:,2),:);

f = estimateFundamentalMatrix(pts1,pts2,...
    'Method','Norm8Point');
[t1, t2] = estimateUncalibratedRectification(f,pts1,...
    pts2,size(image_right));
[I1Rect,I2Rect] = rectifyStereoImages(image_left,image_right,t1,t2);
imwrite(I1Rect,'3L_rect.jpg');
imwrite(I2Rect,'3R_rect.jpg');
```

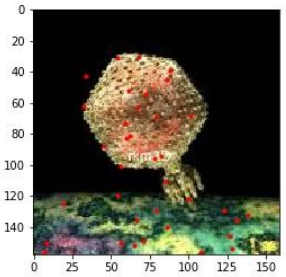
Correspondence Matching Code:

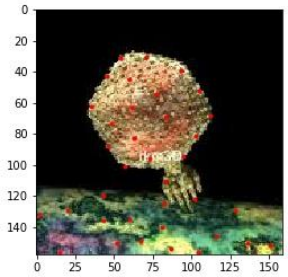
```
def zero_pad(arr,kernel_half):
    x,y=np.shape(arr)
    pad_h=np.zeros((arr.shape[0],kernel_half))
    arr=np.hstack((pad_h,arr))
    arr=np.hstack((arr,pad_h))
    pad_v=np.zeros((kernel_half,arr.shape[1]))
    arr=np.vstack((pad_v,arr))
    arr=np.vstack((arr,pad_v))
    return arr
```

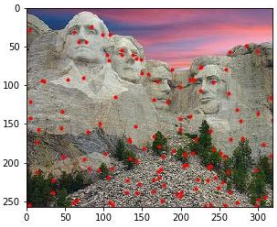
```
import numpy as np
import cv2

imgL=cv2.imread('stereo_images/3L.jpg')
imgR=cv2.imread('stereo_images/3R.jpg')
imgL_gray=cv2.cvtColor(imgL,cv2.COLOR_BGR2GRAY)
imgR_gray=cv2.cvtColor(imgR,cv2.COLOR_BGR2GRAY)
#print(imgL_gray.shape,imgR_gray.shape)
points = cv2.goodFeaturesToTrack(imgL_gray,500,.1,15)
num_points=len(points)
points=np.reshape(points,(num_points,2))
#print(points)
points_x=points[:,1]
points_y=points[:,0]
arr  =imgL_gray
arr2 =imgR_gray
#print(arr.shape,arr2.shape)
x,y=np.shape(arr2)
matches=np.empty((num_points,4))
kernel=5
#diff=np.empty((3,3))
count=0
kernel_half=int(kernel/2)
arr=zero_pad(arr,kernel_half)
arr2=zero_pad(arr2,kernel_half)
print(arr.shape,arr2.shape)
for (u,v) in zip(points_x,points_y):
    temp=arr[int(u)-kernel_half:int(u)+kernel_half+1,int(v)-kernel_half:int(v)+kernel_half+1]
    temp=temp.flatten()
    ssd=[]
    idx_y=[]
    #print('next point')
    i=int(u)
    for j in range(kernel_half,y+kernel_half,1):
            win=arr2[i-kernel_half:i+kernel_half+1,j-kernel_half:j+kernel_half+1]
            win=win.flatten()
            #print(win.shape)
            diff=np.dot(temp,win)
            if np.linalg.norm(temp)!=0 and np.linalg.norm(win)!=0:
                diff=diff/np.linalg.norm(temp)
                diff=diff/np.linalg.norm(win)
            elif np.linalg.norm(temp)==0 and np.linalg.norm(win)!=0:
                diff=diff/0.00000001
                diff=diff/np.linalg.norm(win)
            elif np.linalg.norm(temp)!=0 and np.linalg.norm(win)==0:
                diff=diff/0.00000001
                diff=diff/np.linalg.norm(temp)
            elif np.linalg.norm(temp)==0 and np.linalg.norm(win)==0:
                diff=diff/0.00000001
                diff=diff/0.00000001
            ssd.append(diff)
            idx_y.append(j)
    index=np.argwhere(ssd==max(ssd))
    matches[count,0]=u
    matches[count,1]=v
    matches[count,2]=u
    matches[count,3]=idx_y[index[0][0]]
    count+=1
print(matches)
```

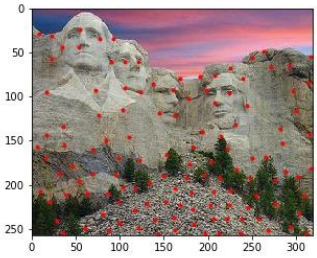
```
import matplotlib.pyplot as plt
img1 = plt.imread('stereo_images/3L.jpg')
plt.plot(matches[:,1],matches[:,0],'r.')
plt.imshow(img1)
plt.savefig('stereo_rect3L.jpg')
plt.show()

img2 = plt.imread('stereo_images/3R.jpg')
plt.plot(matches[:,3],matches[:,2],'r.')

plt.imshow(img2)
plt.savefig('stereo_rect3R.jpg')
plt.show()
```

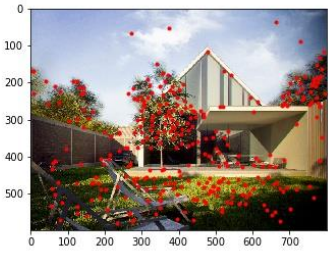
Results and Analysis:

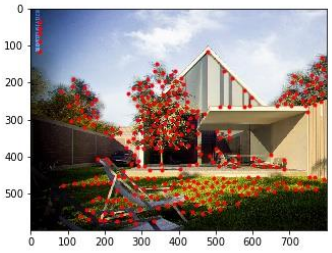












All results are shared in the link below: https://iiitaphyd-my.sharepoint.com/:f:/g/personal/apoorva_srivastava_research_iiit_ac_in/Equcs1y88ABNvIsZAQLwq7MBYHSxrdHKeKON3ZRMf4sZgQ?e=OAO2L7

The Correspondences were better than non- rectified case and rectification helped them find fast but still they are not very accurate as intensity is not a very reliable criteria and much depends on the choice of the window also.

3. **Dynamic programming solution for Stereo Correspondence on the rectified images:**

Dynamic Programming Solution

- Cost of matching: $C(i-1,j-1)+c(i,j)$ if pixels match, $C(i-1,j)+C_o$ if left occlusion, and $C(i,j-1)+C_o$ if right occlusion, where C_o is a high occlusion cost
- Select the minimum from those three and declare match or occlusion accordingly
- Can be setup nicely as a dynamic programming solution working in the i,j space, starting with leftmost pixel match
- Cost of matching: $O(N^2)$ where N is the number of pixels in each scanline.

Several matching scenarios:

- If left pixel $(i-1)$ matches with right pixel $(j-1)$, next pixel i can match pixel j, if the match is good
- Otherwise, it may continue the match with $(j-1)$ with an occlusion cost (due to left occlusion)
- Or, $(i-1)$ can match with j with another occlusion cost (due to right occlusion)

Algorithm:

```
for i = 0 to N do P(i,0)=2; V(i,0)=i*C_O; Initialize 1st column of vertices
for j = 0 to N do P(0,j)=3; V(0,j)=j*C_O; Initialize 1st row of vertices
for i = 1 to N do
    for j = 1 to N do
        v1=V(i-1,j-1) + c(i,j)          Matching edge cost
        v2=V(i-1,j  ) + C_O             Vertical edge/Occlusion
        v3=V(i  ,j-1) + C_O             Horizontal edge/Occlusion
        V(i,j) = min(v1,v2,v3)          Choose shortest path
        P(i,j) = argmin(v1,v2,v3)       Assign predecessor (1, 2, or 3)
i=N; j=N;
while(j>0)                              Backtrack from vertex T
    if P(i,j)==1 then RightMate[j]=i; i=i-1; j=j-1;
    else if P(i,j)==2 do i=i-1;
    else RightMate[j]=UNMATCHED; j=j-1;
```

Code:

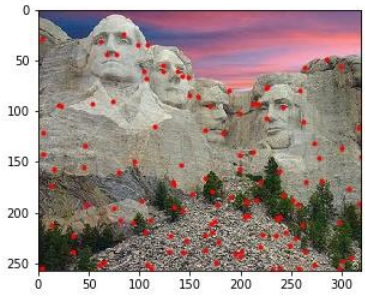
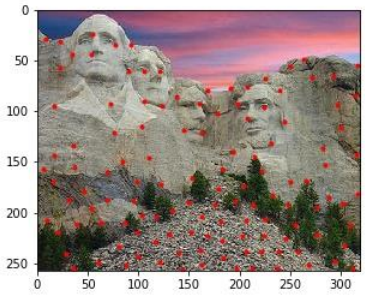
```
import numpy as np
import cv2
from random import sample

imgL=cv2.imread('stereo_images/1L_rect.jpg')
imgR=cv2.imread('stereo_images/1R_rect.jpg')
imL=imgL[:,:,0]
imR=imgR[:,:,0]
#k=25
N=imgL.shape[1]
M=imgL.shape[0]
print(imgL[0,0])
print(M,N)
V=np.empty((N,N))
P=np.empty((N,N))
C_occl=0.22
label=np.empty(3)
matches=[]
for k in range(M):
    for i in range(N):
        V[i,0]=i*C_occl
    for j in range(N):
        V[0,j]=j*C_occl
    for i in range(1,N,1):
        for j in range(1,N,1):
            label[0]=(imL[k,i-1]/255-imR[k,j-1]/255)**2 + (imL[k,i]/255-imR[k,j]/255)**2
            #print(label[0])
            label[1]=(imL[k,i-1]/255-imR[k,j]/255)**2   + C_occl
            label[2]=(imL[k,i]/255-imR[k,j-1]/255)**2   + C_occl
            #print(label)
            V[i,j]=np.min(label)
            P[i,j]=np.argmin(label)
    i=0
    j=0
    while(j<N and i<N):
        if P[i,j]==0:
            matches.append((k,i,k,j))
            i+=1
            j+=1
        elif P[i,j]==1:
            i+=1
        else:
            j+=1
```

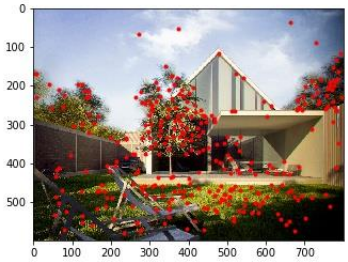
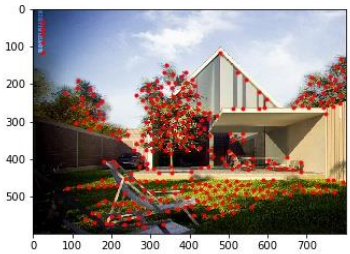
```
p=len(matches)
print(p)
matches=np.asarray(matches)
matches=np.reshape(matches,(p,4))
print(matches)
```

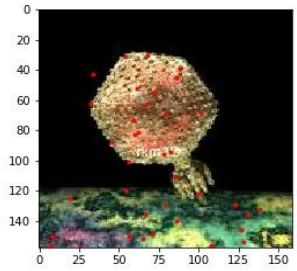
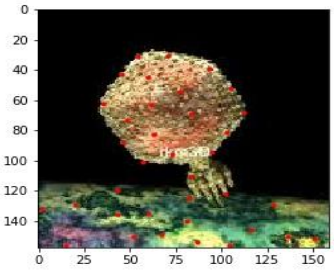
```
[60 60 60]
165 235
33253
[[  0   0   0   2]
 [  0   2   0 185]
 [  0   3   0 186]
 ...
 [164 230 164 232]
 [164 231 164 233]
 [164 232 164 234]]
```

```
import matplotlib.pyplot as plt
matches2=sample(list(matches[:]),150)
matches2=np.asarray(matches2)
img1 = plt.imread('stereo_images/1L_rect.jpg')
plt.plot(matches2[:,0],matches2[:,1],'r.')

plt.imshow(img1)
plt.savefig('stereo_dyn1L.jpg')
plt.show()
img2 = plt.imread('stereo_images/1R_rect.jpg')
plt.plot(matches2[:,2],matches2[:,3],'r.')

plt.imshow(img2)
plt.savefig('stereo_dyn1R.jpg')
plt.show()
```

Results and Analysis:

All results are shared in the link below: https://iiitaphyd-my.sharepoint.com/:f:/g/personal/apoorva_srivastava_research_iiit_ac_in/ElqIu8RmV5FIuVkaecyApH4BmGINSG6rEL_VMtTiOxwgig?e=PnJz3g

The Correspondences were better than rectified case and method was highly fast and accurate with still some outliers. The method is robust against Occlusion and fine tuning the Occlusion cost matters a lot.