Part 2 : Image Mosaics

In this part we do it the same way that we did in part by first getting the point correspondences between two images using SIFT Descriptor as shown (Figure 4). After that we got the homography between them using Singular Value Decomposition to get H. After that we implemented warping, where we warp one image into the plane of the second image and display the combined views, then we warp the points from the source image into the reference frame of the destination to avoid holes in the output as shown (Figure 5), then applying linear interpolation to get the proper coordinates for each point. Then we Create the output mosaic as shown (Figure 6), Homography output is shown as (Figure 6.A.2), (Figure 6.B.2), (Figure 6.C.2).

Used code is attached after figures

Figure 4 : Getting getting the point correspondences using SIFT Descriptor

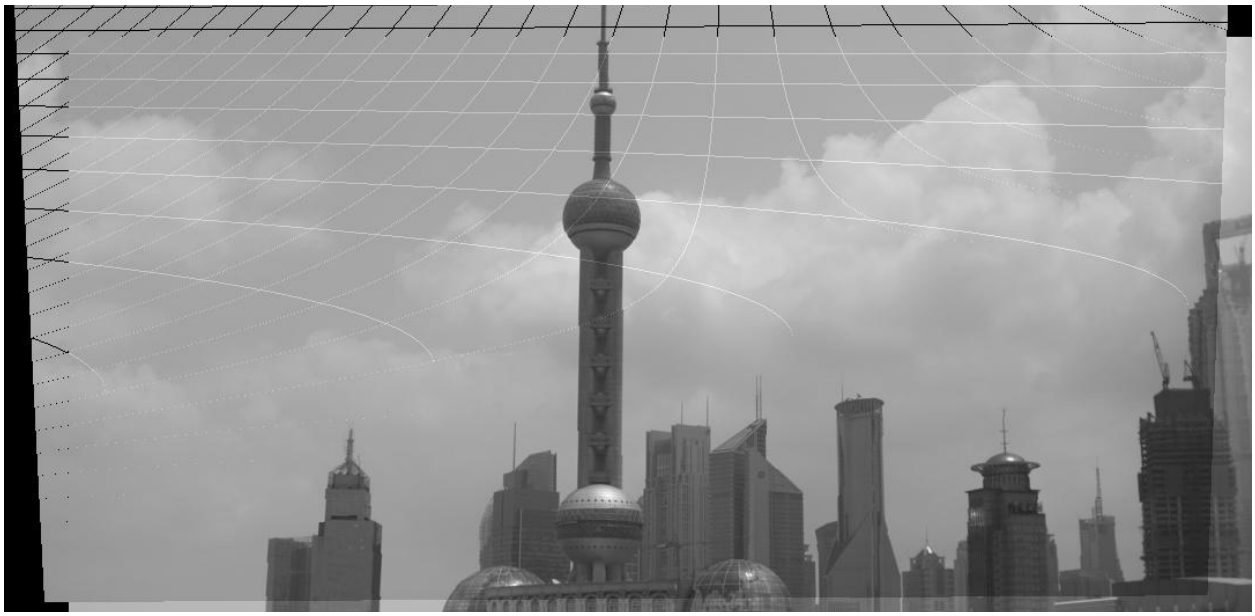
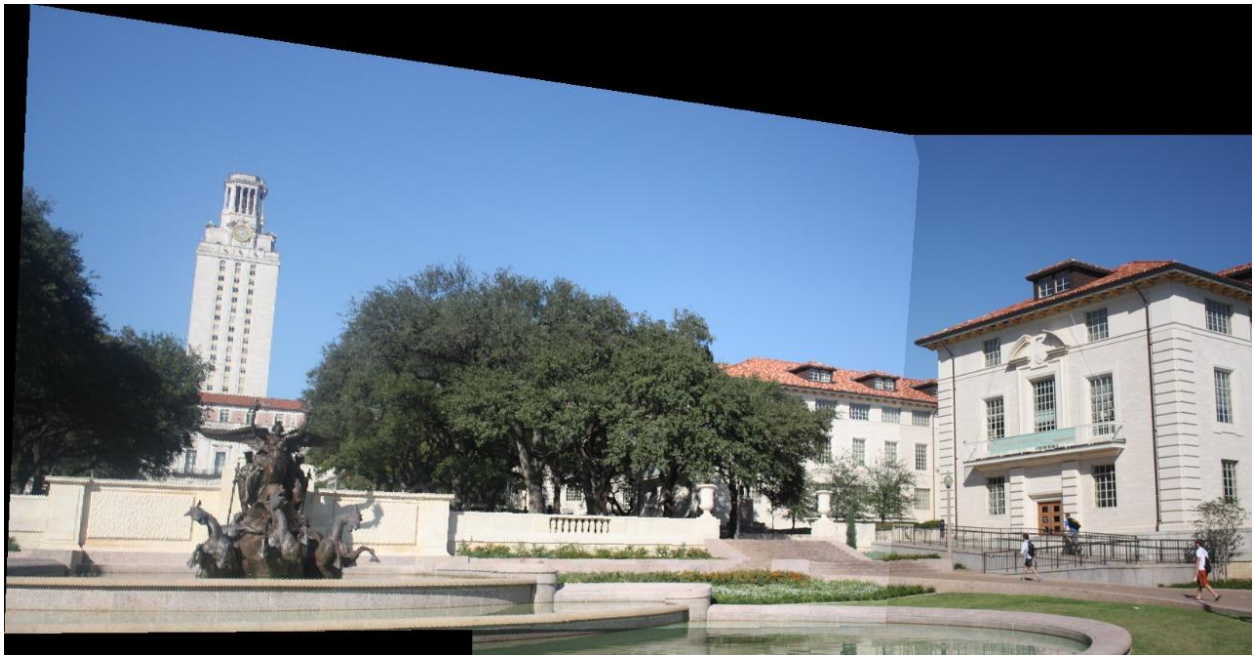


Figure 5 : After Warping, before Linear Interpolation

Figure 6: Image Mosaics



(A.1) : Output Mosaic for test case I

```
Homography from built-in function
[[ 1.29844370e+00 -7.28812388e-02 -5.74509670e+02]
 [ 1.69257597e-01  1.22127305e+00 -1.68398835e+02]
 [ 2.78210586e-04  2.39738843e-05  1.00000000e+00]]
--------------------------------
Homography from our implementation
[[ 1.30676735e+00 -7.02404829e-02 -5.80077080e+02]
 [ 1.70745568e-01  1.23160231e+00 -1.71514730e+02]
```

(A.2) : Homography output for test case I

(B.1) : Output Mosaic for test case II

```
Homography from built-in function
[[ 1.05324230e+00  4.95365671e-02 -5.64334136e+01]
 [-4.37414265e-05  1.07013985e+00 -2.14465062e+02]
 [ 1.26099053e-05  8.65935468e-05  1.00000000e+00]]
--------------------------------
Homography from our implementation
[[ 1.05445818e+00  5.00782833e-02 -5.69577030e+01]
 [ 2.13228006e-04  1.07128365e+00 -2.14981750e+02]
 [ 1.31921639e-05  8.74637393e-05  1.00000000e+00]]
```

(B.2) : Homography output for test case II

(C.1) : Output Mosaic for test case III - (Bouns)

```
Homography from built-in function
[[ 1.05324230e+00  4.95365671e-02 -5.64334136e+01]
 [-4.37414265e-05  1.07013985e+00 -2.14465062e+02]
 [ 1.26099053e-05  8.65935468e-05  1.00000000e+00]]
--------------------------------
Homography from our implementation
[[ 1.05445818e+00  5.00782833e-02 -5.69577030e+01]
 [ 2.13228006e-04  1.07128365e+00 -2.14981750e+02]
 [ 1.31921639e-05  8.74637393e-05  1.00000000e+00]]
```

(C.2) : Homography output for test case III - (Bouns)

Code used:

```
import numpy as np
import cv2
from PIL import Image
import matplotlib.pyplot as plt
from scipy import linalg
from scipy import ndimage, misc
from google.colab.patches import cv2_imshow
```

```
#Interpolataion implementaion
```

```
def interpolataion(original_image,rotated_index,scale,H_inv):

    x=np.floor(rotated_index[1]).astype(int)
    x=x*scale
    y=np.floor(rotated_index[0]).astype(int)
    y=y*scale
    original_index=np.matmul(H_inv,(y,x,scale))
    top_left=original_image[int(original_index[1])%len(original_image),int(original_index[0])%len(original_image[0])]



    x=np.ceil(rotated_index[1]).astype(int)
    x=x*scale
    y=np.floor(rotated_index[0]).astype(int)
    y=y*scale
    original_index=np.matmul(H_inv,(y,x,scale))
    top_right=original_image[int(original_index[1])%len(original_image),int(original_index[0])%len(original_image[0])]



    x=np.floor(rotated_index[1]).astype(int)
    x=x*scale
    y=np.ceil(rotated_index[0]).astype(int)
    y=y*scale
    original_index=np.matmul(H_inv,(y,x,scale))
    down_right=original_image[int(original_index[1])%len(original_image),int(original_index[0])%len(original_image[0])]
```

```
    x=np.ceil(rotated_index[1]).astype(int)
    x=x*scale
    y=np.ceil(rotated_index[0]).astype(int)
    y=y*scale
    original_index=np.matmul(H_inv,(y,x,scale))
    down_left=original_image[int(original_index[1])%len(original_image),int(original_index[0])%len(original_image[0])]


    return top_left,top_right,down_right,down_left
```

#Stitching Implemtaion

```
def stitching(first_image,second_image):

    img_reference=cv2.cvtColor(first_image,cv2.COLOR_BGR2GRAY) #Image that will be rotated
    frame=cv2.cvtColor(second_image,cv2.COLOR_BGR2GRAY) #Fixed Image

    sift = cv2.SIFT_create()
    kp_reference, des_reference = sift.detectAndCompute(img_reference,None)
    kp_frame, des_frame = sift.detectAndCompute(frame,None)
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des_reference,des_frame, k=2)

    correspondence_reference = []
    correspondence_frame = []
    for m,n in matches:
        if m.distance < 0.5*n.distance:
            correspondence_reference.append(kp_reference[m.queryIdx].pt)
            correspondence_frame.append(kp_frame[m.trainIdx].pt)

    A = np.empty((0,9))
    for i in range(len(correspondence_reference)):
        u,v=correspondence_frame[i]
        x,y=correspondence_reference[i]

        x=np.array([
                    [-x , -y , -1 , 0 ,0 ,0 ,u*x, u*y , u],
                    [0,0,0,-x,-y,-1,v*x,v*y,v]
                    ])
```

```
        A = np.vstack( (A,x) )

    u, s, v = linalg.svd(A)
    DOF=np.reshape(v[np.argmin(s)],(3,3))

    img_reference=first_image
    frame=second_image
    final_frame=np.zeros(  (second_image.shape[0]+first_image.shape[0]*2,second_image.shape[1]+first_image.shape[1]*2,3)  ).astype(np.uint8)

    for i in range (img_reference.shape[0]):
        for j in range (img_reference.shape[1]):
                new_cordinates=np.matmul(DOF,(j,i, 1))
                scale=new_cordinates[2] #getting scale to be used for interpolation if needed
                new_cordinates=new_cordinates/new_cordinates[2]
                if (new_cordinates[0].is_integer()==0 or new_cordinates[1].is_integer()==0 ):   #if new pixel cordinates is not a whole number
                    top_left,top_right,down_right,down_left=interpolataion(img_reference,new_cordinates,scale,np.linalg.inv(DOF)) #getting pixels around

                    final_frame[i+int(img_reference.shape[0]),j+int(img_reference.shape[1]),:]=frame[i,j,:]
                    final_frame[np.floor(new_cordinates[1]).astype(int)+int(img_reference.shape[0]),np.floor(new_cordinates[0]).astype(int)+int(img_reference.shape[1]),:]=top_left
                    final_frame[np.ceil(new_cordinates[1]).astype(int)+int(img_reference.shape[0]),np.floor(new_cordinates[0]).astype(int)+int(img_reference.shape[1]),:]=top_right
                    final_frame[np.ceil(new_cordinates[1]).astype(int)+int(img_reference.shape[0]),np.ceil(new_cordinates[0]).astype(int)+int(img_reference.shape[1]),:]=down_right
                    final_frame[np.floor(new_cordinates[1]).astype(int)+int(img_reference.shape[0]),np.ceil(new_cordinates[0]).astype(int)+int(img_reference.shape[1]),:]=down_left

                else:
                    final_frame[i+int(img_reference.shape[0]),j+int(img_reference.shape[1]),:]=frame[i,j,:]
                    final_frame[new_cordinates[1]+int(img_reference.shape[0]),new_cordinates[0]+int(img_reference.shape[1]),:]=img_reference[i,j,:]

    xs,ys,zs = np.where(final_frame!=0)
    final_frame = final_frame[min(xs):max(xs)+1,min(ys):max(ys)+1,min(zs):max(zs)+1]
    H,status=cv2.findHomography(np.array(correspondence_reference), np.array(correspondence_frame),cv2.RANSAC, 5.0)
    return final_frame,DOF/DOF[2][2],H
```