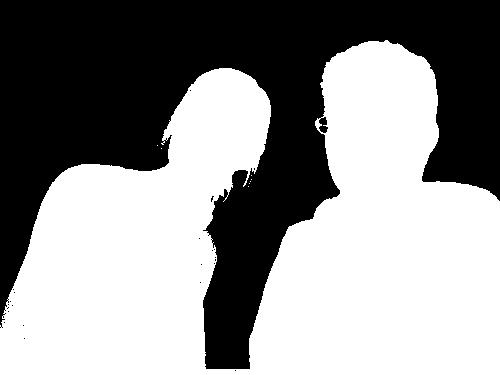
CS364/AM792 : Computer Vision

Assignment 1: Image Processing and Feature Matching

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. The first task was done by using Python’s Pillow library with which I opened both the greenscreen.jpg and my custom bg.jpg file. I resized the custom background to be a bit larger than the greenscreen picture. I then created matrices of pixel values from both the greenscreen and background images. I visited each pixel from the greenscreen image and changed the pixels to black when the pixel had a green channel value of higher than 127, a red channel value of lower than 120 and a blue channel value of lower than 150. Then, I created a new blank image, visited each pixel in the edited greenscreen picture, and when copied the black pixels to the new image. When the pixels was not black, I copied the pixels but changed them to white.Next, I visited each pixel in the background image’s matrix and when it reached the right bottom corner’s pixels, I replaced the RGB values of the background image with the RGB values of the greenscreen image, provided that the greenscreen’s pixel was not a black pixel. Thus, I changed a range of green pixels on the greenscreen image to a uniform black color and when pasted on top of the background, the black pixels were ignored.



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. I added a black frame to the picture, applied the averaging filter and removed the black frame again.  
 3 x 3 average mask 7 x 7 average mask 13 x 13 average mask

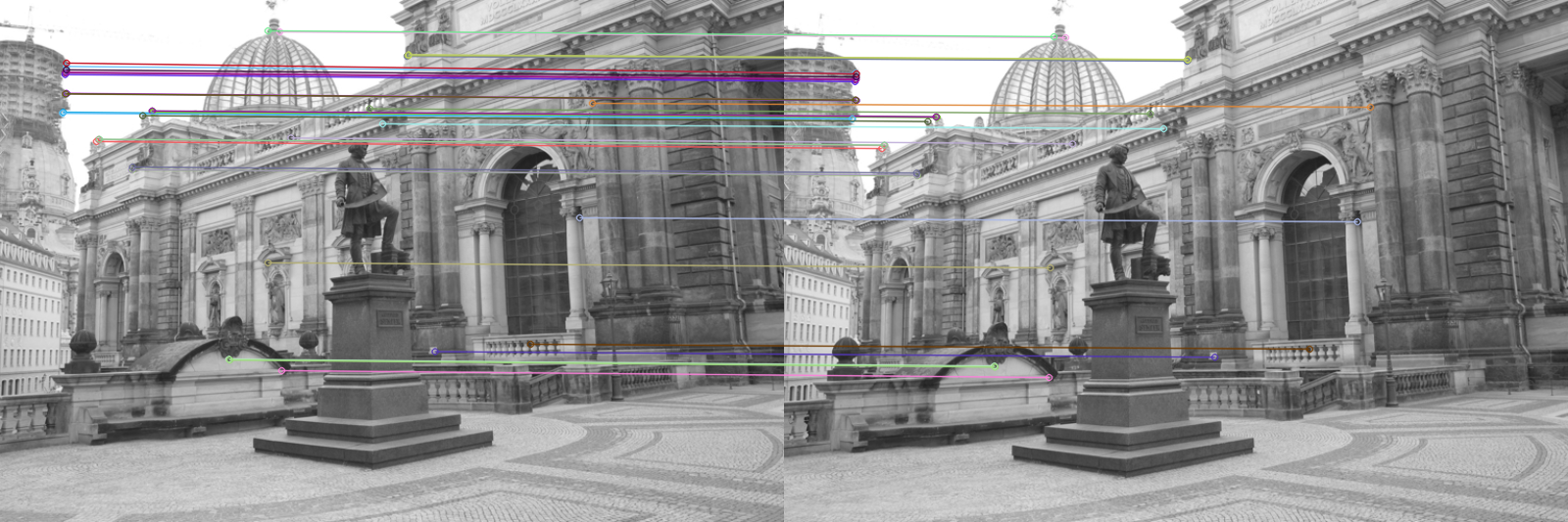
 3 x 3 median mask 7 x 7 median mask 13 x 13 median mask



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. (a) I used OpenCV’s ORB feature matcher. The process is simple: you import two images,  
 resize them to fit on one screen, call OpenCV’s ORB feature matching’s detectAndCompute function to return feature coordinates and descriptor vectors on both  
 images and then use OpenCV’s ORB feature matching’s match function to match the  
 corresponding feature points. Then you generate the dots and lines between the pictures  
 by calling the drawMatches function and display the result.

I wrapped all the above mentioned functions from OpenCV in more user-friendly  
 functions and used it to execute two examples.

4.(b)