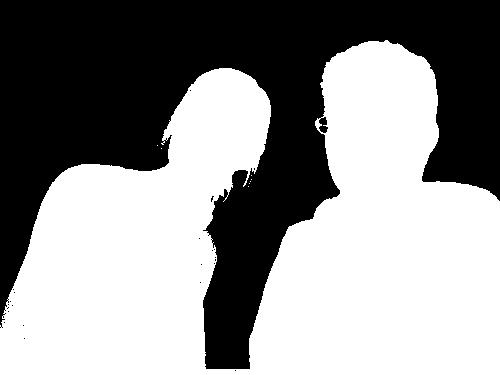
CS364/AM792 : Computer Vision

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 opened a color image, coverted it to grayscale and applied a median averaging (smoothing) filter to it. I created a new blank image, visited each pixel and set the values to *B(r , c) = A(r , c) + k(A(r , c) − Ā(r , c))* where B is the blank image, A is the original greyscaled image and *Ā is the smoothed greyscale image and k=1.*

*Afterwards, I scaled the pixel values linearly to fit into the range (0,255). All operations was done on floating point numbers and thereafter converted back to integers.*

*B A Ā*

**

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

3. For nearest neighbour, I opened the image and retrieved its dimensions. I initialized a new, white blank image with the dimensions of the original image multiplied by the scale factor. I visited each pixel in the new blank image and set the value according to the formula:

B(x\_new , y\_new ) = A(x\_r , y\_r ) where x\_r = round(x\_old) and y\_r = round(y\_old)  
with one exception: where the rounding generates an invalid position outside the image borders, the rounding function will subtract 1 from the final result.

For billinear, I opened the image and retrieved its dimensions. I initialized a new, white blank image with the dimensions of the original image multiplied by the scale factor. I visited each pixel in the new blank image and checked whether the position could be exactly mapped back to a valid pixel position in the original image. If so, I copied the pixel value. If not, I calculated tuples t1, t2, t3 and t4 which corresponds to A(x\_f , y\_f ), A(x\_f, y\_c ), A(x\_c, y\_f), A(x\_c, y\_c) and the euclidean distance between them. I then set the pixel value for all color channels according to

B(x\_new , y\_new ) = (y\_c − y \_old )[(x\_c − x\_old )A(x\_f , y\_f ) + (x\_old − x\_f )A(x\_c , y\_f )]+ (y\_old − y\_f ) [(x\_c − x\_old )A(x\_f , y\_c ) + (x\_old − x\_f )A(x\_c , y\_c )

**scale factor = 0.5**

Original Nearest neighbour Billinear



**scale factor = 2**

Original Nearest neighbour Billinear

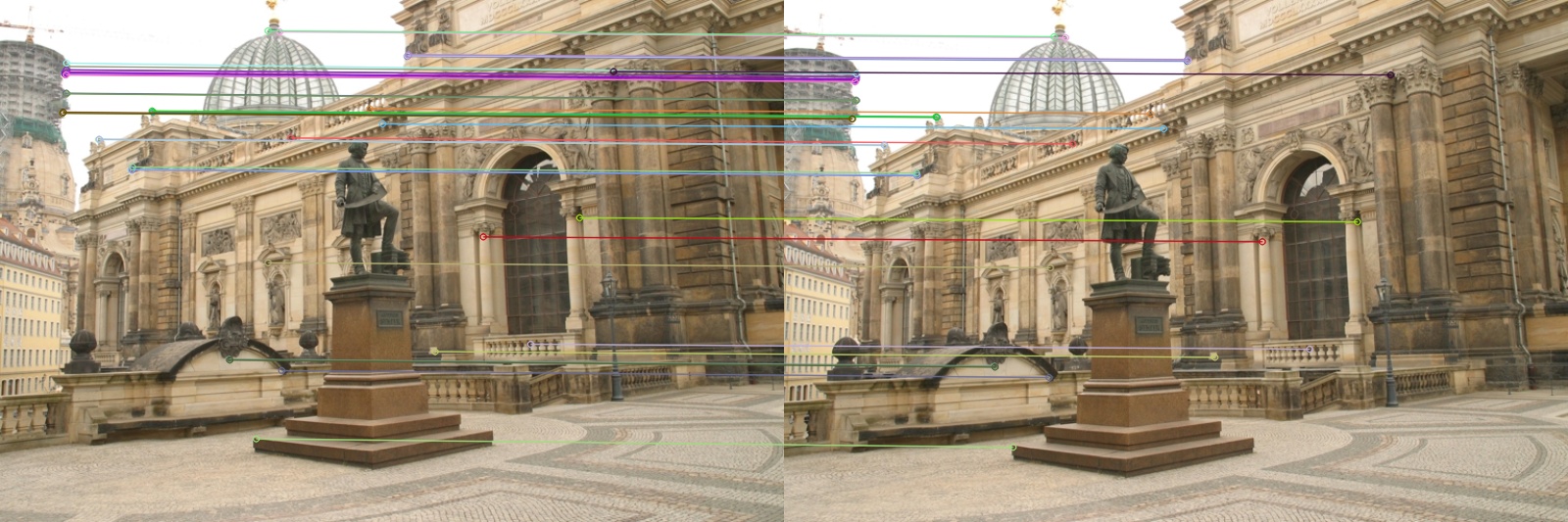


Nearest neighbour Billinear



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

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)