Assignment 3
Computer Vision

Image Mosaics

Nayera Abd El Samie ***ID:4032***
Esraa Wael El-Hawash ***ID:3959***
Guehad Mohamed Ahmed ***ID:3861***

Image Mosaics

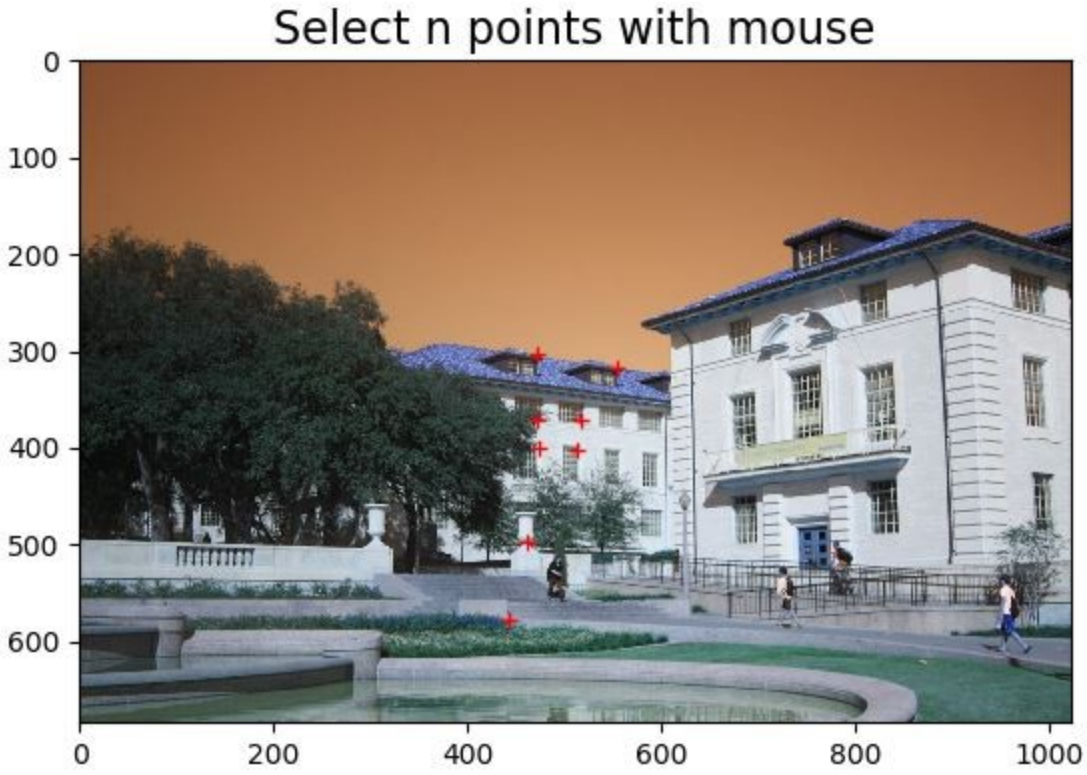
In this exercise, we were asked to implement an image stitcher that uses image warping and homographies to automatically create an image mosaic. We will focus on the case where we have two input images that should form the mosaic, where we warp one image into the plane of the second image and display the combined views. This problem will give some practice manipulating homogeneous coordinates, computing homography matrices, and performing image warps.
For simplicity, we will specify corresponding pairs of points manually using mouse clicks.

1) Getting correspondences:

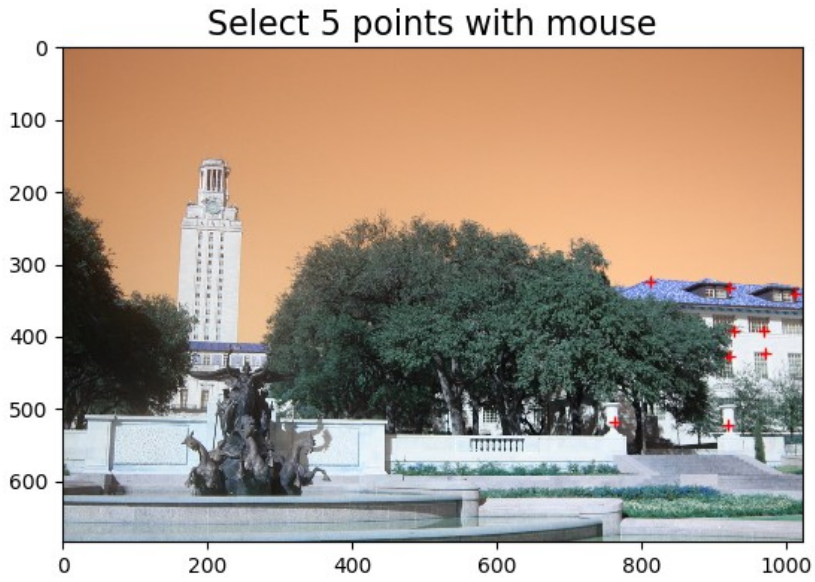
we started by reading the 2 views in 2 images using the cv2 imread function.
Next we plot the first image using plt.imshow() and then we prompt the user to click on the mouse to identify the points of interest using the ginput() function which takes as parameters the number of clicks which we left to the choice of the user.

This function returns the array of points (x & y coordinates) clicked which would equal the number of clicks.
The same is done for the next image and by now we have two arrays of clicked points in the 2 views we wish to work with.

Points selected in 1st View:



Points selected in 2nd View:



2) Computing the homography parameters:

The Goal is to write a function that takes a set of corresponding image points and computes the associated 3x3 homography matrix H. This matrix transforms any point p in one view to its corresponding homogeneous coordinates in the second view.

We needed the arrays of pairs of points that we received from the ginput function to be re-arranged in this manner:

$$p_i = \begin{bmatrix} -x_i & -y_i & -1 & 0 & 0 & 0 & x_i x_i' & y_i x_i' & x_i' \\ 0 & 0 & 0 & -x_i & -y_i & -1 & x_i y_i' & y_i y_i' & y_i' \end{bmatrix}$$

And for each point (I) however many there are this was the general matrix form we needed to reach to get H:

$$PH = \begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1 x_1' & y_1 x_1' & x_1' \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1 y_1' & y_1 y_1' & y_1' \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_2 x_2' & y_2 x_2' & x_2' \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & x_2 y_2' & y_2 y_2' & y_2' \\ -x_3 & -y_3 & -1 & 0 & 0 & 0 & x_3 x_3' & y_3 x_3' & x_3' \\ 0 & 0 & 0 & -x_3 & -y_3 & -1 & x_3 y_3' & y_3 y_3' & y_3' \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x_4 x_4' & y_4 x_4' & x_4' \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & x_4 y_4' & y_4 y_4' & y_4' \end{bmatrix} \begin{bmatrix} h1 \\ h2 \\ h3 \\ h4 \\ h5 \\ h6 \\ h7 \\ h8 \\ h9 \end{bmatrix}$$

So that’s exactly what we did we did a function that takes the pairs of points and separates them into x and y and that puts them in the form [xi,yi,1,0,0,0,-xi…] and so on and another function that takes the output of the 2 rows formed and concatenates them to however many pairs of points chosen and stacks them to the large matrix (nx9) we see above.

The 1d column array is the h values we’ll get from solving our equations which we will later reshape to the 3x3 form.

Homogeneous:

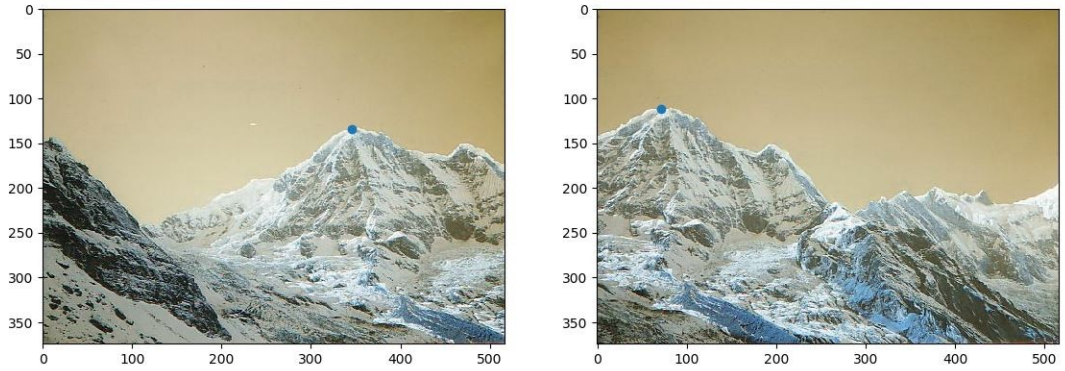
Our first approach to find H was through ‘PH=0’ which we solved using the svd function (Singular Value Decomposition).
We reshape the output to 3x3 and we’re done.

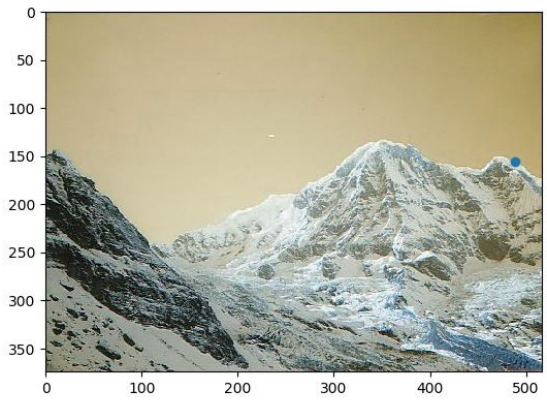
Non-Homogeneous:

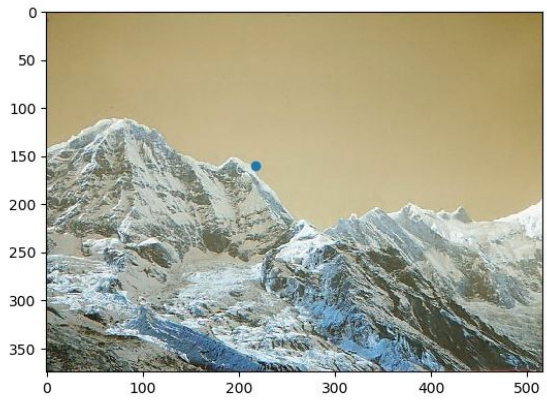
Next approach was to use PH=p’ , p’ being the points in the second view
we did from perspective of solving an Ax=b equation.
In the same function that concatenates the large matrix P we create an array an append the second view points in it and that function returns both the ‘P’ and p’.
we then use np.linalg.lstsq to solve this equation and we’re left with H which we reshape to 3x3 and then we’re done.

Verification:

To verify the H matrix we got from our implementation we did the following:
we start by plotting both views in the same figure next to each other.
Then in a while loop that loops for n points
we use the ginput function again to choose which point we want to map in the other view.
We then initialize a new array of these two chosen coordinates, Next we multiply the chosen point array with the H matrix we found.
The output point is transformed to coordinates and scattered (plotted) on the figure of the second view pointing out the prime point.







3) Warping between image planes:

After computing homography matrix, we use it to transform one of the views to stitch it to the other and that is done by the following steps:

First, we set the maximum dimensions of the resulted image which will be double the size of one image (from column wise) as if we have unrolled the 2 images side by side, while the number of rows will be same.

Since the images are RGB we will split them and loop on each of them to extract each pixel intensity. Since we want to project one plane to another we used homogeneous dimensions so that the transformed pixel is the multiplication of the homography matrix and the pixel homogeneous dimensions.

After that we will convert the homogeneous dimensions to Cartesian coordinates and take each transformed pixel intensity from the image and put it in the resulted array. However, the problem with the forward warping is the holes between pixels.

The solution is using inverse warping to fill these holes using bilinear interpolation so, knowing the intensity of neighbors of the holes we could fill them.

Then we add the intensities of the other view, doing the same operation on each channel and lastly merge the 3 channels and get the output.

4) Ransac:

we also tried using ransac to automatically detect features and key points and get the matches between them without choosing the points from both view manually which led to much better results and more accurate outputs.

Simply what we did was call the SIFT_create function from cv2 library which extracts features from images.

Then we get key point and descriptors from the ‘detectAndCompute() function for each image.
We then use the BF matcher and knnMatch for descriptors and we turn the matches to an np array.

Next we make sure that more than 4 matches were found before continuing.
For each match found in the source image we extract the index of key point there and same goes for the destination image.

Now we have the actual points as coordinates and we put them as parameters in the built-in cv2.findHomography() function alongside the ‘RANSAC’ parameter.

5) Results: