Computer Vision

Problem set 3

Eunho Yang (CS ID : eunho)

II. Programming problem

I implemented the main components listed in the problem document with functions (more than one functions for one component)

- Getting correspondence

'getCorres' function to get correspondence points with 'ginput'. This function returns two matrices contain clicked points for each image. Each row in the matrix represents input point in (row, column) coordinates.

- Computing the homography parameters

'calculateH' function calculates matrix H with inputs of two images and their correspondence points. As learned in the class, we can make the matrix equation which contains

[x y 1 0 0 0 (-xx') (-yx')] * [a b c d e f g h]^T = x'

[0 0 0 x y 1 (-y'x) (-yy')] * [a b c d e f g h]^T = y' for every correspondence points (x,y), (x',y')

And using '₩' operator, we can solve the regression problem and have matrix H

'verifyH' function has 5 input arguments: *im1, im2, H, im1_co, im2_co*. im1 and im2 are images to be merged. H is H matrix and, im1_co and im2_co are correspondent points for each image. If H equals 0, it just shows the clicked points on the images. If not, it computes the correspondent points of image2 by applying homography to im1_co.

- Warping between image planes

'warp' function warps image im1 into im2 plane with homography matrix H. First, four edge points of im1, apply homography and find the output image size. And then apply inverse warping from every output image pixel to im1. If the result of inverse warping is NaN, it contains -1.

'warp' uses 'applyH' function. Its input arguments are matrix H and point matrix to apply homography.

- Create the output mosaic

'mosaic' function executes all the above functions and merge two input images. Image1 is warped into the image2 plane. Image1 overwrite the pixel values of image2 if pixel of warped im1 does not contain -1.

1. Apply your system to the provided pair of images, and display the output mosaic.

- warp 'uttower1.jpg' into the plane of 'uttower2.jpg'

>> im1 = imread('uttower1.jpg');

>> im2 = imread('uttower2.jpg');

>> [out o1 o2] = mosaic(im1,im2);

'mosaic' function firstly shows the clicked points for each image. Color for each point represents the relative order (it is randomly generated color. We can check the order and the positions)

Figure <Clicked points>

And then, it shows the results of the verification. Points on the second image are the results of applying homography to points of image1.

Figure <Verification of H>

'mosaic' function returns mosaic image, warped image1 and image2.

>> imshow(uint8(o1))

>> imshow(uint8(o2))

>> imshow(out)

- warp 'uttower2.jpg' into the plane of 'uttower1.jpg'

>> [out o1 o2] = mosaic(im2,im1);

Clicked points

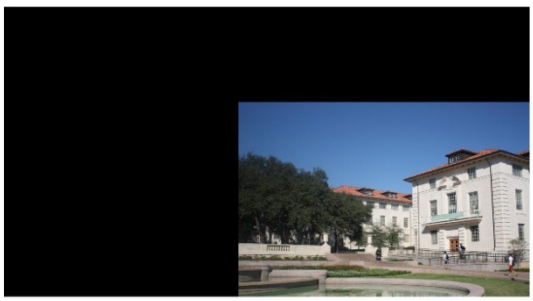
Figure <Clicked points>

Verify H

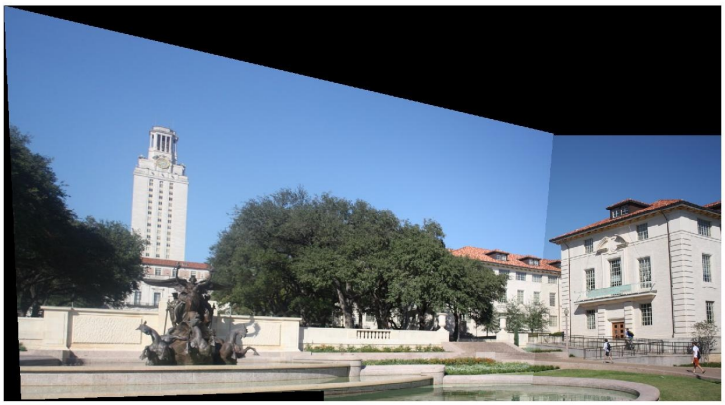
Figure <Verification of H>

>> imshow(uint8(o1))

>> imshow(uint8(o2))

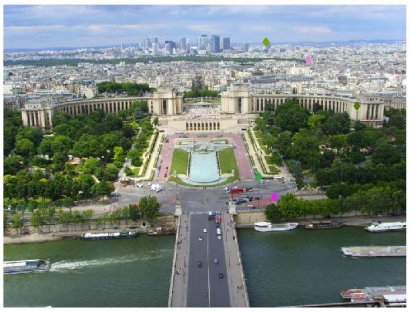


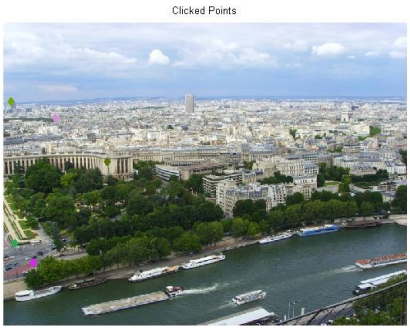
>> imshow(out)



2. Show two additional examples of mosaics you create using images that you have taken.

a. Input images (paris1.jpg and paris2.jpg) and clicked points





>> imshow(out)

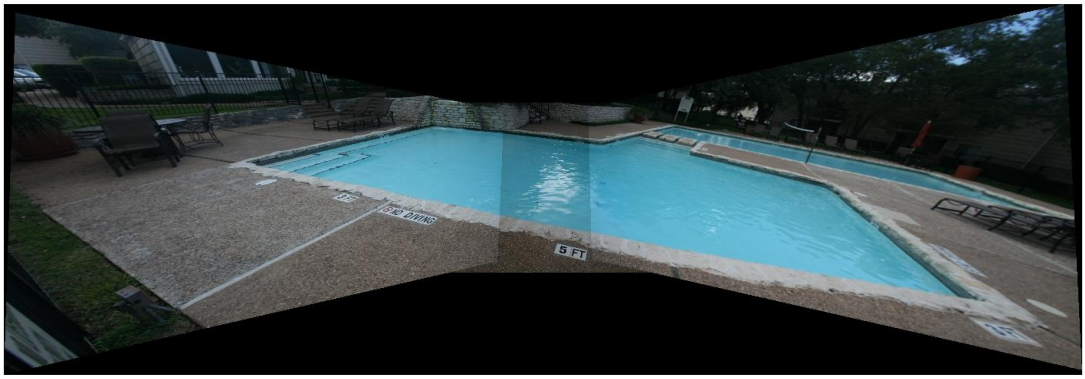
There are discontinuities on ships because they are moving.

b. Input images (pool1.jpg, pool2.jpg and pool3.jpg) and clicked points - Test on three pictures

>> imshow(out_temp) **à** intermediate output from two images

Run mosaic function again with above output and 3rd image
Clicked points

>> imshow(out_temp) **à** Final output



3. Warp one image into a 'frame' region in the second image

>> im_bg = imread('lcd1.jpg');

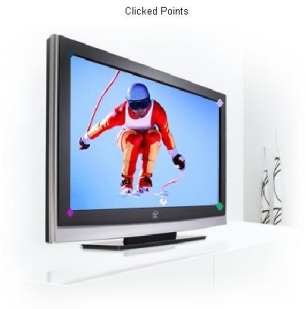
>> im_face = imread('face.jpg');

>> imshow(im_face)

>> imshow(im_bg)

>>[out o1 o2] = mosaic(im_face, im_bg);

clicked points



>> imshow(uint8(o1))

>> imshow(out)

III. Extra credit

1. Implement RANSAC for robustly estimating the homography matrix from noisy correspondences.

'ransac' function takes two correspondence points matrices as input arguments and returns the maximum set of correspondences which are consistent. To activate 'RANSAC' functionality, please uncomment the line 20 of 'mosaic.m' file

- without RANSAC

I clicked 8 correspondence pairs. 7 pairs are correct and one (gray in the below image) is noisy input.

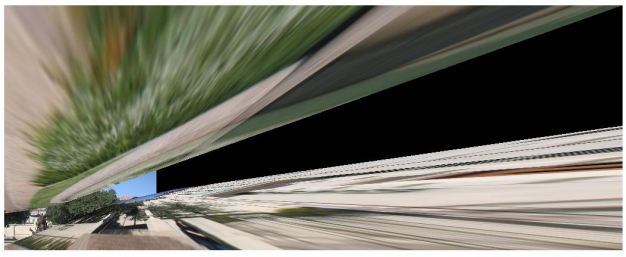
>> [out o1 o2] = mosaic(im1,im2);

Figure <Clicked points>

Figure <Verification of H>

As show in the above figure, the results from verification indicate wrong points in the image2.

>> imshow(out)



We can confirm that final output is severely distorted.

- with RANSAC

I tested with same points after calling 'ransac' function.

>> [out o1 o2] = mosaic(im1,im2);

Figure <Verification of H>

Above figure shows the results of verification process. We can find that noisy input is removed. As result, we can get a good mosaic image

>> imshow(out)

- Another RANSAC example

Total 9 correspondence pairs with 2 noisy pairs.

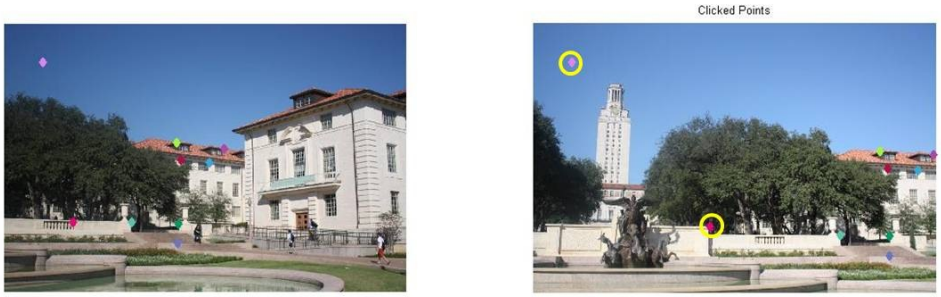


Figure <Clicked points>

After running RANSAC,

Figure <Verification of H>

Finial mosiac image is omitted because it is quite similar with the previous result.

2. Refine the initial correspondences automatically by searching small patches near the clicked points for a good alignment.

For all pairs (all the nodes in the patch of image1 * all the nodes in the patch of image2), 'refineCorres' function tries to find the pairs with minimum SSD. Sometimes it will be helpful to make a good result, but in many cases, it does not perform well. It highly depends on the points that I clicked and the size of local window. I think another complicated mechanism (ex. using SSD of the pixels in the local window) is needed to improve the performance. Note that following result is one of the good cases.

- without refinement

>> [out o1 o2] = mosaic(im1,im2);

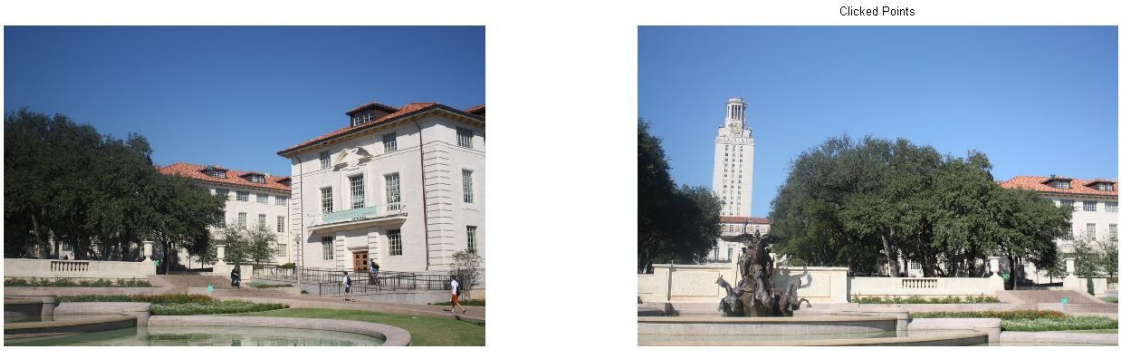


Figure <Clicked points>

I clicked the correspond pairs with slight errors.

>> imshow(out)

Output is not so bad. However, we can find small discontinuities in the boundaries between two images.

- with refinement,

After saving above clicked points of previous result, I applied the same correspondences into 'refineCorres' function. The result of 'clicked points' is the same as the above case. The following figures show the verification of H after refinements.

Figure <Verification of H>

We can find out that selected points in image2 (right one) move slightly toward the correct target points. (Images are too small to check selected points exactly. I attached the original output files (clicked_points.fig and after_refine.fig).

>> imshow(out)

3. Rectify an image with some known planar surface.

I implemented 'changeView' function to do this task. This function is similar with 'mosaic' function, but it takes only one input image and the points to be mapped as the arguments.

```
>> im1 = imread('pdp.jpg');
>> output = changeView(im1, [1 1; 300 1; 300 400; 1 400]);
```

```
>> imshow(uint8(output))
```

4. Make a short video in the style of the HP commercials.

I implement 'makeFrames' function to do this task. This function makes several output images where input image is warped into selected frames. Direction and plane for output images should be predetermined and embedded in the function. I tried to make just one set of output images with very simple moving as followings