Name: Ali Hussein Ali

Number: 39

Problem Set 4
Image Mosaics

In this exercise, you will 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.

Figure 1: Image Mosaics

(a) First View

(b) Second View

(c) Final Output

Here are the main components you will need to implement:

1 Getting correspondences

Use the provided code to get manually identified corresponding points from two views. "ginput" function provides an easy way to collect mouse click positions. The results will be sensitive to the accuracy of the corresponding points, when providing clicks, choose distinctive points in the image that appear in both views.

2 Computing the homography parameters

Write a function that takes a set of corresponding image points and computes the associated 3×3 homography matrix H. This matrix transforms any point p in one view to its corresponding homogeneous coordinates in the second view, p', such that $p' = Hp$. Note that p and p' are both $3D$ points in homogeneous coordinates. The function should take a list of $n \geq 4$ pairs of corresponding points from the two views, where each point is specified with its $2D$ image coordinates.

We can set up a solution using a system of linear equations $Ax = b$, where the 8 unknowns of H are stacked into an 8-vector x , the $2n$-vector b contains image points from one view, and the $2n \times 8$ matrix A is filled appropriately so that the full system gives us $\lambda p' = Hp$. There are only 8 unknowns in H because we set $H_{3,3} = 1$. Solve for the unknown homography matrix parameters. [Useful numpy functions: 'lstsq', reshape]

Verify that the homography matrix your function computes is correct by mapping the clicked image points from one view to the other, and displaying them on top of each respective image (matplotlib: imshow, followed by hold and plot). Be sure to handle homogenous and non-homogenous coordinates correctly.

3 Warping between image planes

Write a function that can take the recovered homography matrix and an image, and return a new image that is the warp of the input image using H . Since the transformed coordinates will typically be sub-pixel values, you will need to sample the pixel values from nearby pixels. For color images, warp each RGB channel separately and then stack together to form the output.

To avoid holes in the output, use an inverse warp. Warp the points from the source image into the reference frame of the destination, and compute the bounding box in that new reference frame. Then sample all points in that destination bounding box from the proper coordinates in the source image (linear interpolation). Note that transforming all the points will generate an image of a different shape / dimensions than the original input.

[Useful functions: numpy.meshgrid, scipy.interpolate.RectBivariateSpline]

4 Create the output mosaic

Once we have the source image warped into the destination images frame of reference, we can create a merged image showing the mosaic. Create a new image large enough to hold both (registered) views; overlay one view onto the other, simply leaving it black wherever no data is available. Do not worry about artifacts that result at the boundaries.

5 Requirements

After writing and debugging your system:

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

2. Show one additional example of a mosaic you create using images that you have taken. You might make a mosaic from two or more images of a broad scene that requires a wide angle view to see well. Or, make a mosaic using two images from the same room where the same person appears in both.

3. [Optional] Warp one image into a frame region in the second image. To do this, let the points from the one view be the corners of the image you want to insert in the frame, and let the corresponding points in the second view be the clicked points of the frame (quadrilateral) into which the first image should be warped. Use this idea to replace one surface in an image with an image of something else. For example – overwrite a billboard with a picture of your choice, or project a drawing from one image onto the street in another image, or replace a portrait on the wall with someone elses face, or paste a Powerpoint slide onto a movie screen,..

- **Getting corresponding Manually**

pts1,pts2 = getCorrespondence(imageA, imageB,4)

pass the two images as aparamters and the number of points

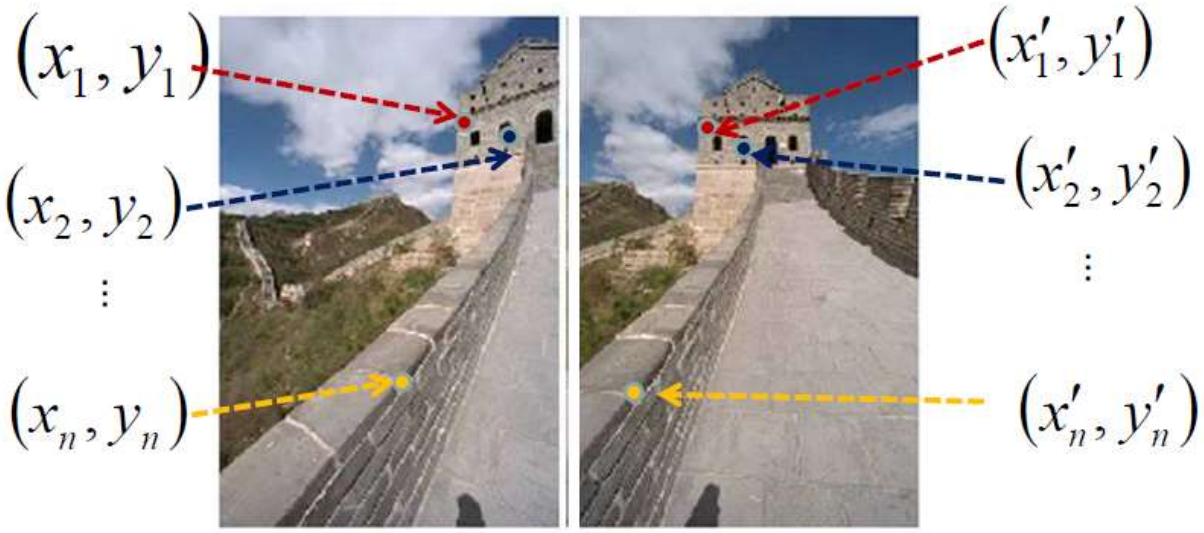
number of points =4

as we need at least 8 equations to get The homography matrix

```
def getCorrespondence(imageA, imageB , num):
    fig = plt.figure()
    figA = fig.add_subplot(1,2,1)
    figB = fig.add_subplot(1,2,2)
    figA.imshow(imageA,origin='upper')
    figB.imshow(imageB,origin='upper')
    plt.axis('image')
    pts = plt.ginput(n=num*2, timeout=0)
    pts = np.reshape(pts, (2, pts, 2))
    return pts[0],pts[1];
```

- **Computing the homography parameters**

Homography

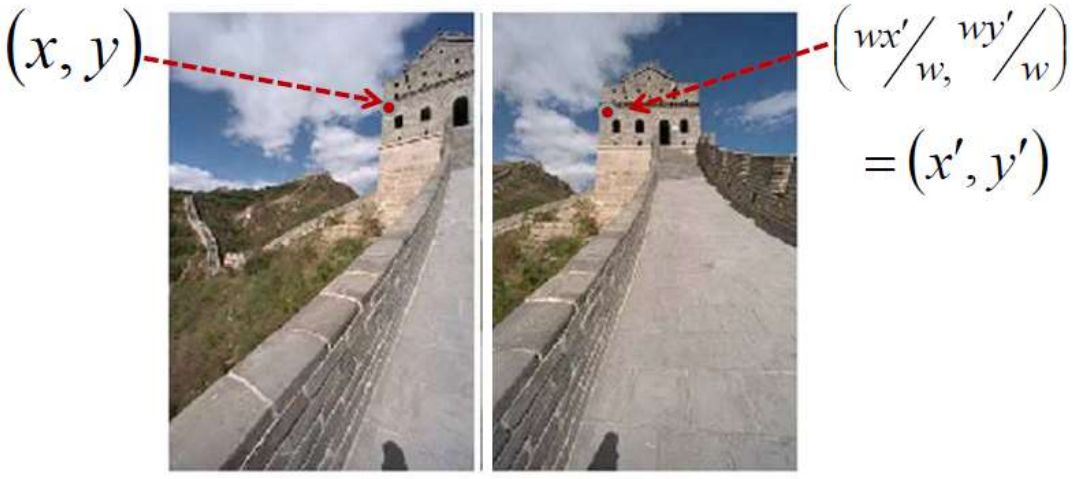


To compute the homography given pairs of corresponding points in the images, we need to set up an equation where the parameters of H are the unknowns…

p’ = Hp

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Can set scale factor *i*=1. So, there are 8 unknowns. Set up a system of linear equations: **Ah = b** where vector of unknowns h = [a,b,c,d,e,f,g,h]^T Need at least 8 eqs, but the more the better… Solve for h. If overconstrained, solve using least-squares: $\min \|Ah - b\|^2$



To **apply** a given homography **H**

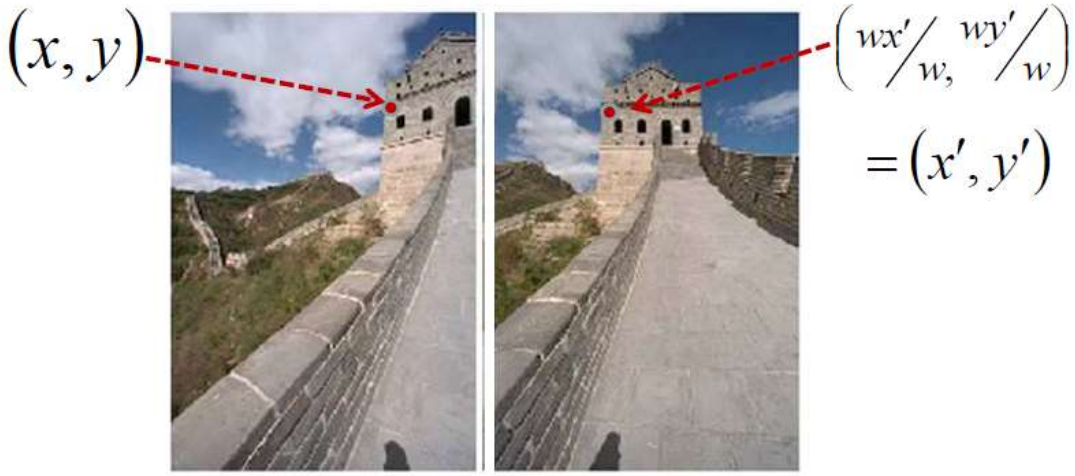
- Compute **p'** = **Hp** (regular matrix multiply)
- Convert **p'** from homogeneous to image coordinates

```
# Computing the homography parameters
def computeH(pts1,pts2):
    l1, l2 = len(pts1), len(pts2)
    assert l1 == l2

    A=[]
    B=[]
    for j in range(l1*2):
        i=int(j/2)
        if(j%2==0):
            A.append([pts1[i][0],pts1[i][1],1,0,0,0,-pts2[i][0]*pts1[i][0],-pts2[i][0]*pts1[i][1]])
        else:
            A.append([0,0,0,pts1[i][0],pts1[i][1],1,-pts2[i][1]*pts1[i][0],-pts2[i][1]*pts1[i][1]])

    for i in range(l2):
        B.append(pts2[i][0])
        B.append(pts2[i][1])

    # H matrix 3*3
    a,b,c,d,e,f,g,h = lstsq(A, B)[0]

    return [[a,b,c],[d,e,f],[g,h,1]]
```

p' = Hp

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Verify that the homography matrix your function computes is correct




```
def Verify(H,pts):
    len= np.shape(pts)[0]
    for i in range(0,len):
        p=np.zeros((3,1), dtype=np.double);
        p[0]=pts[i][0]
        p[1]=pts[i][1]
        p[2]=1

        newp= np.dot(H, p)
        newp=newp/newp[2]

        print 'point('+str(pts[0])+','+str(pts[1])+') ---map to--->
('+str(newp[0])+','+str(newp[1])+')'
```

Warping between image planes

Function new image that is the warp of the input image using H

```
def warping2Images(H , image , image2):
    H_img = image.shape[0];
    W_img = image.shape[1];
```

Map The boundries of image To image2

```
# get the boundaries point
    Point1=np.dot(H,np.array([0,0,1]));
    Point1= Point1/ Point1[2];
    Point2=np.dot(H,np.array([W_img-1,0,1]));
    Point2= Point2/ Point2[2];
    Point3=np.dot(H,np.array([0,H_img-1,1]));
    Point3=Point3/Point3[2];
    Point4=np.dot(H,np.array([W_img-1,H_img-1,1]));
    Point4=Point4/Point4[2];

    dim1=np.shape(image);
    height2=dim1[0];
    width2=dim1[1];


    # get the exact location for the point (x,y)
    minX = min( Point1[0], Point2[0],Point3[0],Point4[0]);
    minX=min(minX,0);

    maxX = max( Point1[0], Point2[0],Point3[0],Point4[0]);
    maxX=max(maxX,width2);

    minY = min( Point1[1], Point2[1],Point3[1],Point4[1]);
    minY=min(minY,0);

    maxY = max( Point1[1], Point2[1],Point3[1],Point4[1]);
    maxY=max(maxY,height2);

    shift_V = 0;
    shift_H = 0;
    if(minY < 0):
        shift_V = -minY;
    if(minX < 0):
        shift_H = -minX;
    print shift_V;
    print shift_H;
```

The returned Image size

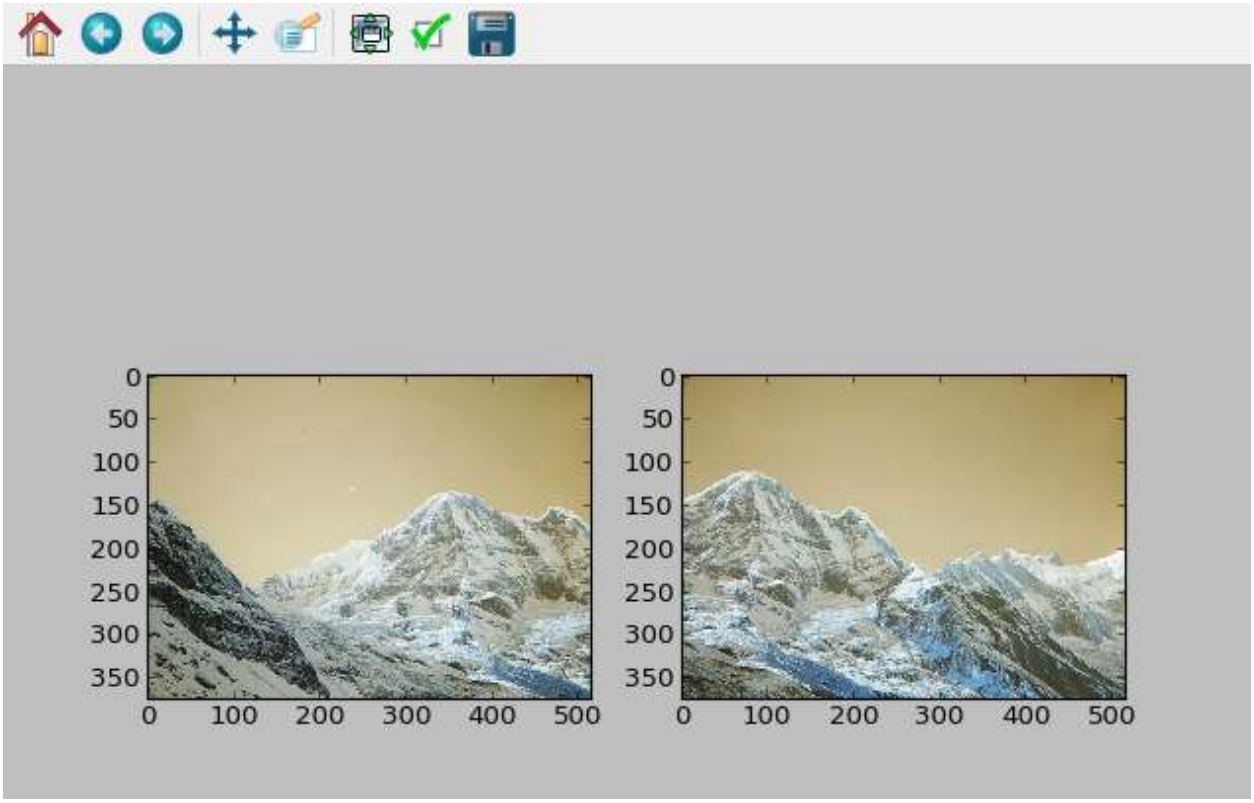
```
    warped_image = np.zeros((maxY+shift_V , maxX+shift_H,3),
dtype=np.double);
    HInverse = np.linalg.inv(H);
    # Interpolation
    hh = np.arange(H_img)
    ww= np.arange(W_img)

    red = image[:, :, 0]
    green = image[:, :, 1]
    blue = image[:, :, 2]
```

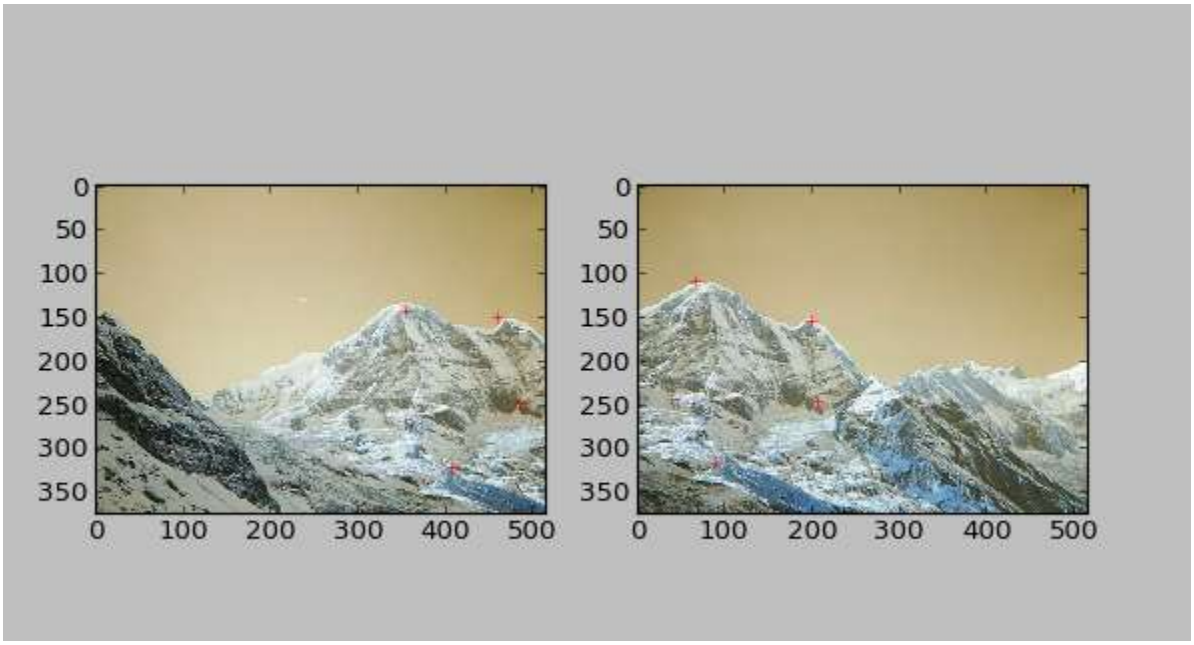
Create RGB Channels

```
    red_container = scipy.interpolate.RectBivariateSpline(hh, ww, red, kx=2,
ky=2);
```

```
    green_container = scipy.interpolate.RectBivariateSpline(hh, ww, green,
kx=2, ky=2);
    blue_container = scipy.interpolate.RectBivariateSpline(hh, ww, blue,
kx=2, ky=2);

transfer The Pixels to The Warpped Image

    for i in range(int(minY), int(maxY)):
        for j in range( int(minX), int(maxX)):
            temp_point = np.array([j,i,1]);
            point = np.dot(HInverse, temp_point);
            point = point/point[2];
            x=int(point[1]);
            y=int(point[0]);

            warped_image[i+ shift_V][j+shift_H][0]=int(red_container.ev(x ,
y));
            warped_image[i+ shift_V][j+shift_H][1]=int(green_container.ev(x ,
y));
            warped_image[i+ shift_V][j+shift_H][2]=int(blue_container.ev(x ,
y));



    h2,w2,d2 = image2.shape;

    for i in range(h2):
        for j in range(w2):
            warped_image[int(i+shift_V)][int(j+shift_H)]=(image2[i][j]);
    return warped_image;
```

Sample Run


0
50
100
150
200
250
300
350
0
100
200
300
400
500


0
50
100
150
200
250
300
350
0
100
200
300
400
500

The Corrosponding Points

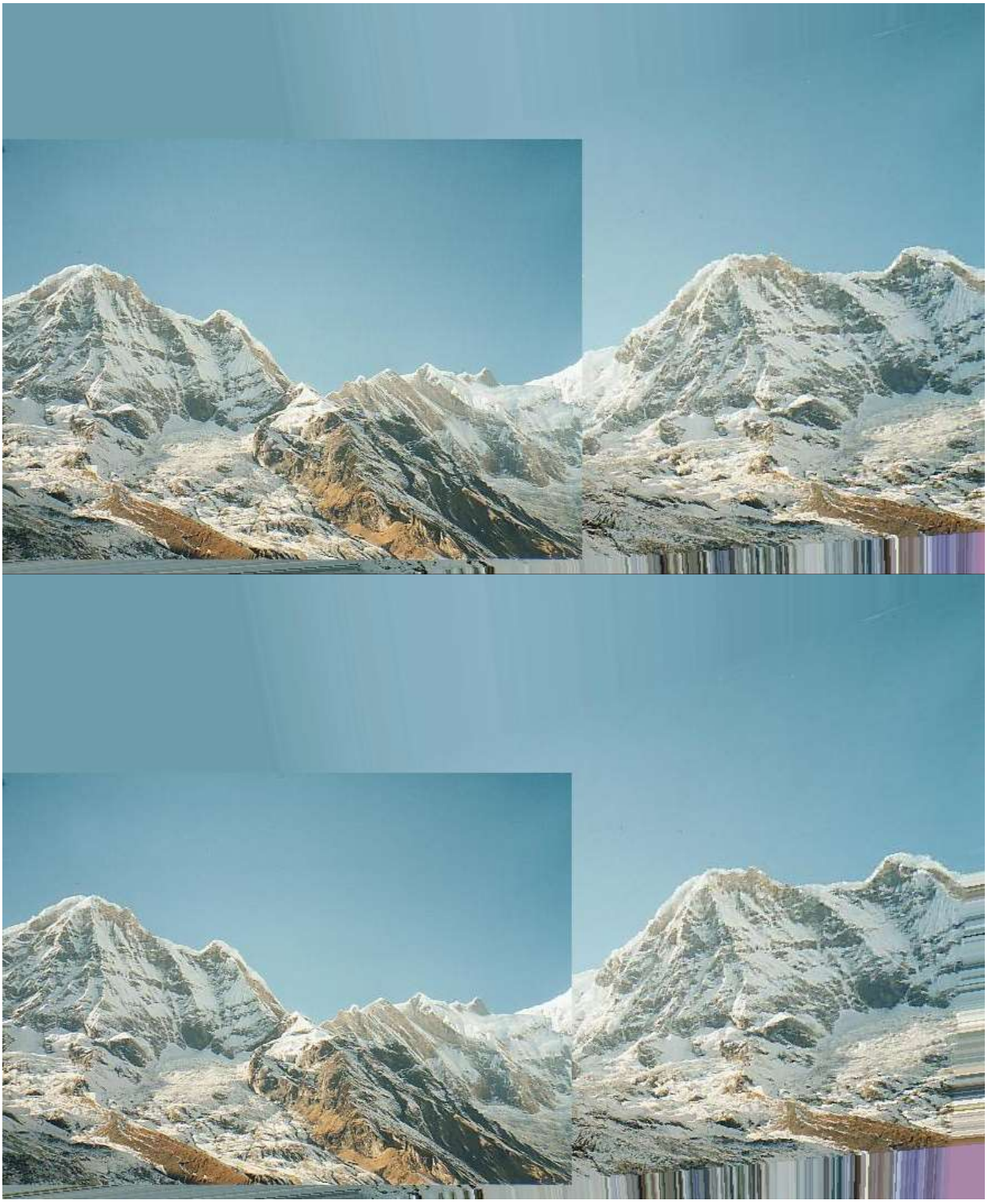
```
[[  67.04354839  108.53306452]
 [ 200.04596774  154.39596774]
 [ 206.92540323  246.12177419]
 [  89.975       317.20927419]]
[[ 460.42217742  149.80967742]
 [ 485.64677419  248.41491935]
 [ 407.67983871  321.79556452]
 [ 354.9375      142.93024194]]
```

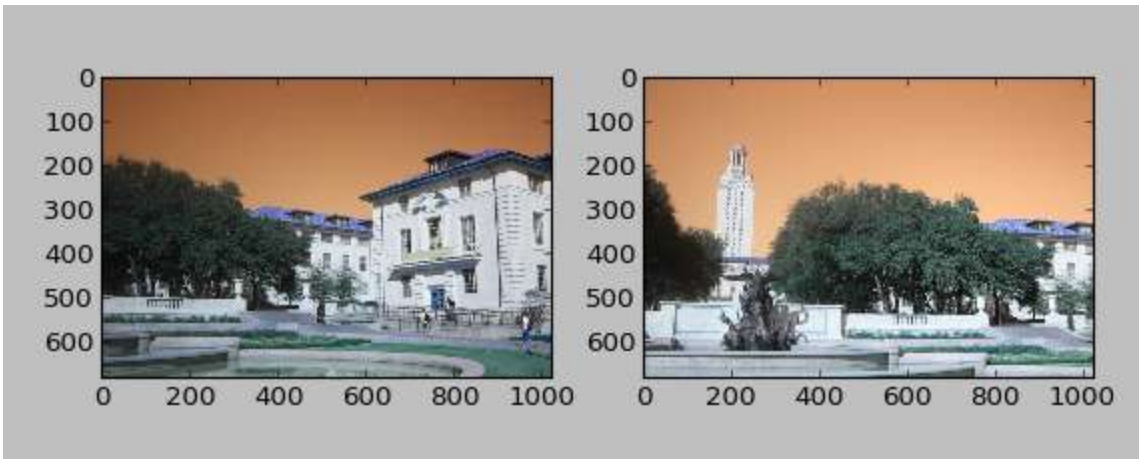
homography matrix

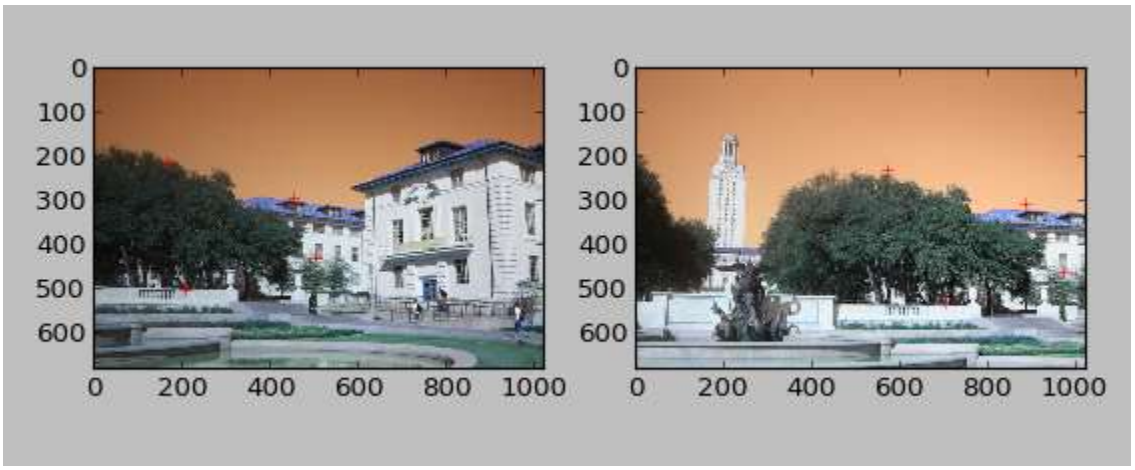
```
[[-0.97835244594319559, -0.82237250911657056, 472.72234950995659], [-
0.11652736392892482, -0.24387470831224078, 137.70949499979099], [-
0.0023944552542942699, -0.0013734719017839403, 1]]
```

Verfication of H

```
point([  67.04354839  108.53306452],[ 200.04596774  154.39596774]) ---map to-
--> ([ 460.42217742],[ 149.80967742])

point([  67.04354839  108.53306452],[ 200.04596774  154.39596774]) ---map to-
--> ([ 485.64677419],[ 248.41491935])

point([  67.04354839  108.53306452],[ 200.04596774  154.39596774]) ---map to-
--> ([ 407.67983872],[ 321.79556451])

point([  67.04354839  108.53306452],[ 200.04596774  154.39596774]) ---map to-
--> ([ 354.93749999],[ 142.93024193])
```








```
[[ 453.69354839  295.58064516]
 [ 503.65483871  431.83870968]
 [ 208.42903226  504.50967742]
 [ 167.5516129   213.82580645]]
[[ 887.24193548  309.20645161]
 [ 973.53870968  463.63225806]
 [ 705.56451613  531.76129032]
 [ 573.8483871   231.99354839]]
```

homography matrix

```
[[0.73519116730886103, 0.63916650779152107, 332.83537581405403], [-
0.16576142962402063, 1.2953372740777369, -9.5929220666042063], [-
0.00036640352468719772, 0.00044064856523773344, 1]]
```

Verification Of H

```
point([ 453.69354839  295.58064516],[ 503.65483871  431.83870968]) ---map to-
--> ([ 887.24193548],[ 309.20645161])
point([ 453.69354839  295.58064516],[ 503.65483871  431.83870968]) ---map to-
--> ([ 973.53870968],[ 463.63225806])
point([ 453.69354839  295.58064516],[ 503.65483871  431.83870968]) ---map to-
--> ([ 705.56451613],[ 531.76129032])
```

point([453.69354839 295.58064516],[503.65483871 431.83870968]) ---map to--> ([573.8483871],[231.99354839])