ECE 661 - Homework-2

Vishveswaran Jothi
vjothi@purdue.edu

08-30-2016

1 Theory and Mathematical Calculation required to solve this problem

Theory behind the solution for this problem is explained in the below section. First we need to find the homography between the world plane and the image plane. Once the homography is found using the sample points in both images. The world plane is mapped to image plane to obtain the corresponding pixel values of the image and to insert it into the world plane. The Method to obtain homography and pixel value is described in the following sections.

1.1 Homography Estimation

The equation that transforms a point X in the 2D image plane to a point X' in another 2D plane is given by X'=HX.
The H matrix is given by:

$$H = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

In our calculation we set "i" (i.e) $H_{3,3}$ to one.
Now, Consider a point X = $[x,\ \ y,\ \ z]^T$ to be transformed to X' = $[x',\ \ y',\ \ z']^T$
x'= ax+by+zc
y'= dx+ey+zf
z'= gx+hy+z
so $x_{new} = \frac{x'}{z'}; y_{new} = \frac{y'}{z'}$
Therefore, $x_{new} = \frac{ax+by+zc}{gx+hy+z}$
$y_{new} = \frac{dx+ey+fz}{gx+hy+z'}$
$x_{new} = ax + by + zc - gxx_{new} - hyx_{new}$
$y_{new} = dx + ey + fz - gxy_{new} - hyy_{new}$
Now we have 8 unknowns and two equations. So we need atleast 4 points as minimum to find the unknowns. Now we write the four equations in the matrix form as

$$Ah = b,$$

where A is given by:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_{1new} & -y_1x_{1new} \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_{1new} & -y_1y_{1new} \\ .... & & & & & & & \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_{4new} & -y_4x_{4new} \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y_{4new} & -y_4y_{4new} \end{bmatrix}$$

h is given by:

$$\begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix}$$

and b =

$$\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{bmatrix}$$

h=(A^-1).b

If we use more than four points to compute homography matrix, we can use either Pseudo Inverse or Singular Value Decomposition on A matrix to find the parameters of the homography matrix (h). Thus Homography is found between two planes.

1.2 Selection of pixel values

We can select the pixel values in three methods.
1. Round-down/ Round-up
2. Weighted Average from the nearby pixels.
3. Bi-linear Transformation

In this exercise, two of three are computed (i.e) Round-down and Weighted Average transformation.

1.2.1 Round-down/ Round-up

In this method we round down the pixel value to the nearest pixel. Say if the x and y coordinates are [102.345,120.167], then the resultant coordinate is given by [102,120]. This method reduces the computation time but the accuracy of the pixel value in the resultant image has to be compromised. The output using this method is presented in Figure 5 to Figure 7. Also in this problem we are not concerned with the minute detailing of the picture unlike edge detection / corner detection. This method is a perfect fit for this problem. The total run time is 1 minute and 53.053 seconds.

1.2.2 Weighted Average

This method is to accurately obtain the pixel value even if the resultant pixel coordinate falls between two pixels. This method is also implemented and the output can be seen in Figure 8 to Figure 10. In this method, when the pixel falls between two pixel coordinates, the surrounding four pixel coordinates are considered and the pixel value of each of the surrounding pixel coordinate is added with a weighting ratio of the distance from which the resultant pixels falls from its lower and upper rounded pixel value. The time taken to run the script is 3minute 18.437seconds. The difference in the resultant image between the Weighted average method and Round-down method is shown in Figure 11.
Consider four points ABCD and the transformed pixel falls in between them. Then the distance between point P and A be DA, similarly for p to B its DB and for C its DC and for D its DD. The equation of weighted average can be given by:

$$mapped_point_value = \frac{\frac{A}{DA^2} + \frac{B}{DB^2} + \frac{C}{DC^2} + \frac{D}{DD^2}}{\frac{1}{DA^2} + \frac{1}{DB^2} + \frac{1}{DC^2} + \frac{1}{DD^2}}$$

2 Implementation of the Solution

2.1 Task 1 Projecting Seinfeld image into 1a, 1b, 1c

Step1: Obtain the Boundary points of PQRS from each image using gimp editor.
Step2: Load the images 1a,1b,1c,1d into the python script.
Step3: Find the homography between the images 1d and 1a, 1d and 1b, 1d and 1c using the four boundary points of 1a,1b,1c and image boundary of the image 1d.
Step4: Apply the homography on the world plane (reference plane) to get the corresponding point on the image plane that needs to be transformed. This method is used rather than applying the image plane to transform into the world plane or reference plane because the using the latter with cause black spots or noise into the resultant image since all points in the image plane cannot be perfectly matching with the reference plane especially near the boundary regions between image and reference image.
Step5: For mapping the reference plane to the image plane, a dummy image similar to the base image is used with two pixel values white and black (255, 0) as the mapping region is not rectangle, the normal method of using for loop and repeating the process from the starting point(top left) of the boundary region to the end (bottom right) of the bounding region yield poor results.
Step6: The pixel value is obtained by using approximation methods like "Round down" Figure 5 and Weighted average Figure 8. Thus obtained pixel value is mapped into the image in the reference plane.

The difference in the approximation and weighted average is significant while time taken for the weighted average and approximation is less. Hence weighted average is implemented in the Task -2 and Task-3

2.2 Task 2 Projecting 1a to 1c plane

Step1: Load the images and the reference points obtained for Task 1 from gimp.
Step2: Find Homography matrix from 1c to 1b, then between 1b to 1a. To find the resultant matrix multiply both homographies in the order $H_{ba}XH_{cb}$. The reason is as same given for the task 1 (i.e)

Figure 1: The Reference image 1a.jpg

to avoid noises and pixel values of zeros inside the image boundary.

Step3: Using the 1c as reference create a base image with same boundary as image 1c and fill it with zeros so it looks black.

Step4: Using the homography matrix map the 1c image plane to 1a image plane to obtain the pixel values. Here only "Weighted Average" method is used to reduce the distortion even though the computation is slightly high. The corresponding pixel values in 1a are copied to 1c image plane. The resultant image looks similar to image 1c. The run time was 6 minutes and 34.72 seconds.

Figure 2: The Reference image 1b.jpg

Figure 3: The Reference image 1c.jpg

Figure 4: The Image to be projected into all the three reference images. The coordinates considered are the bounds of the complete image.

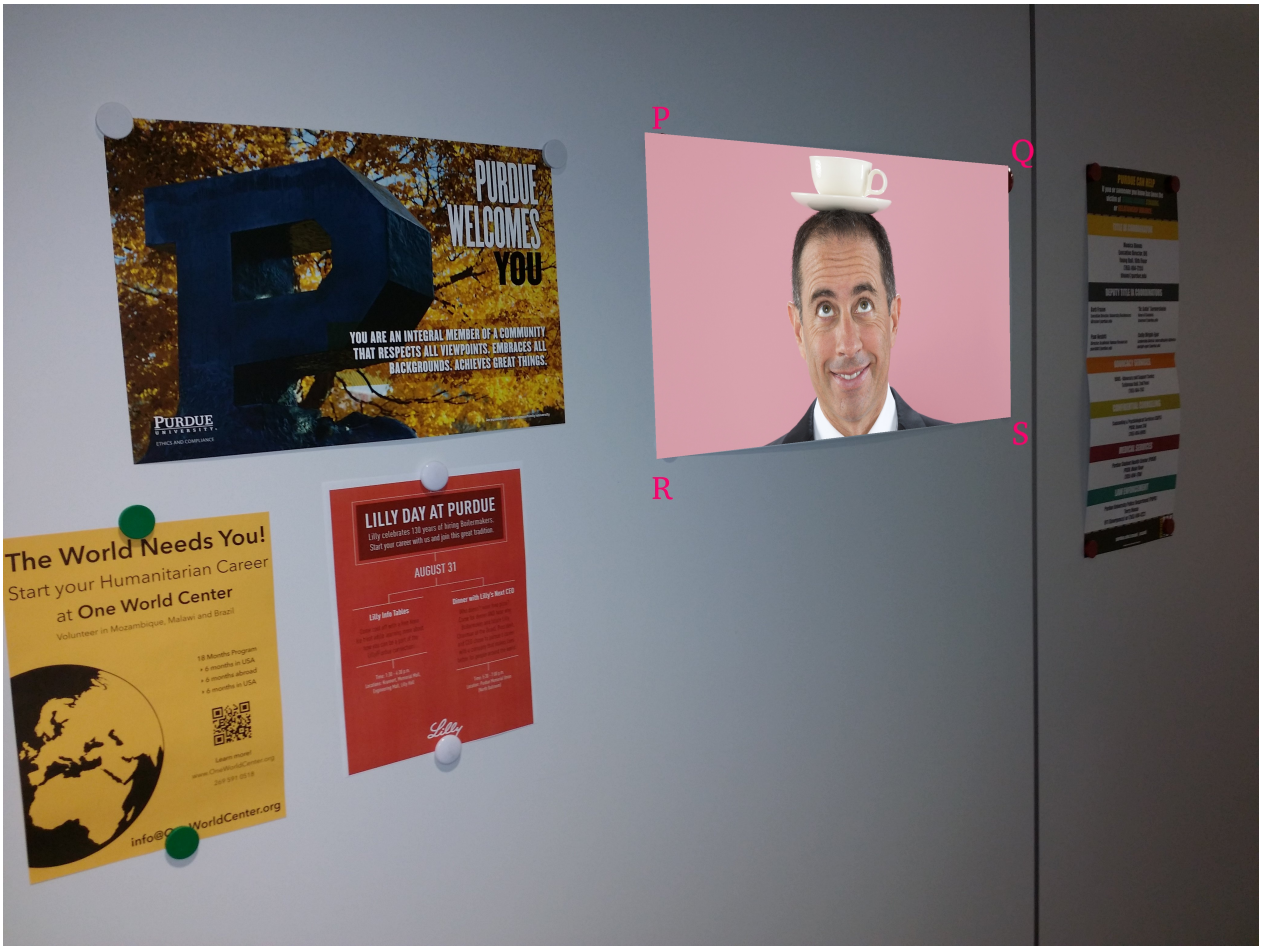

Figure 5: The image of Seinfeld into 1a.jpg

Figure 6: The image of Seinfeld into 1b.jpg

Figure 7: The image of Seinfeld into 1c.jpg

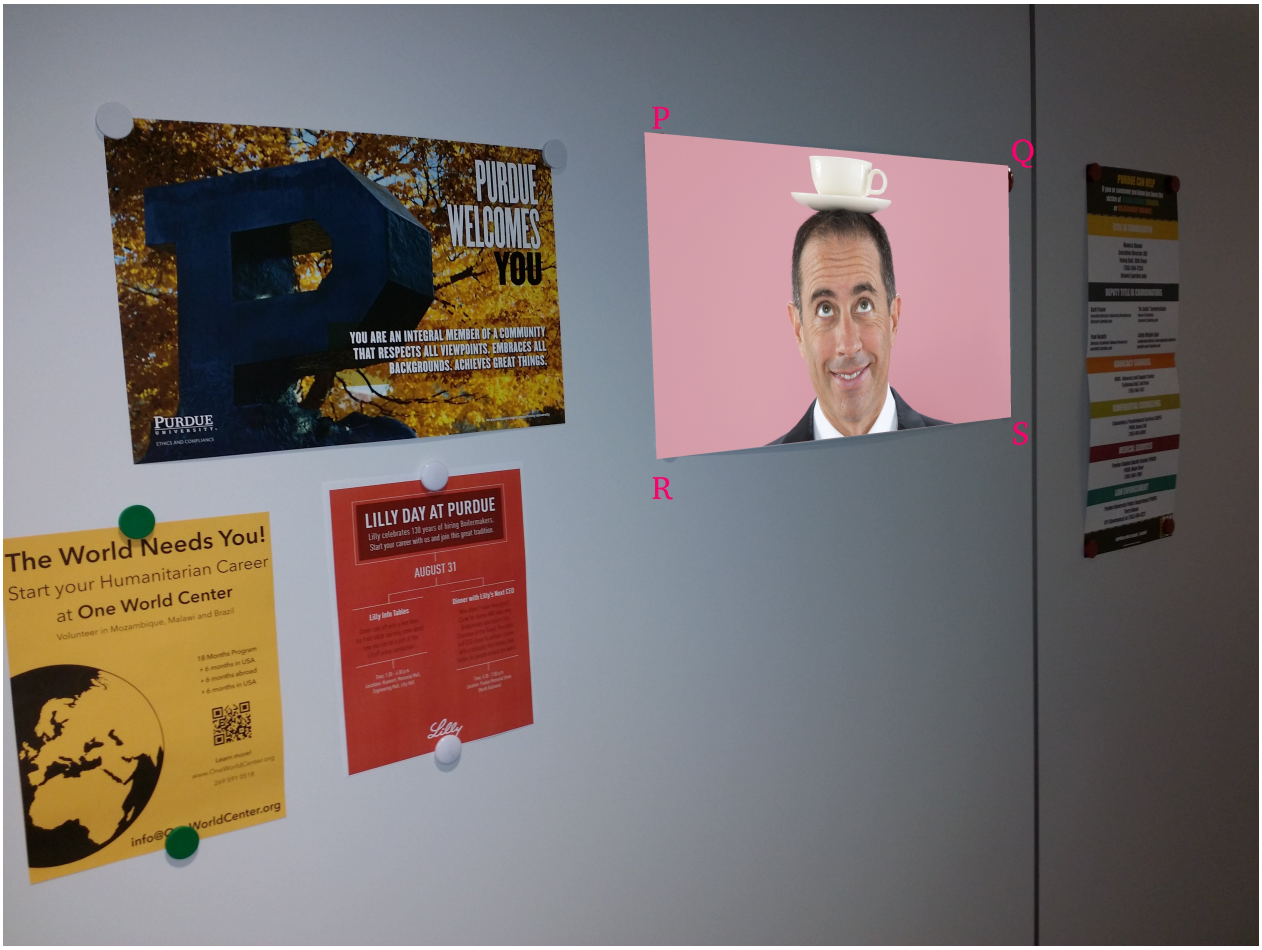

Figure 8: The image of Seinfeld into 1a.jpg with Weighted average for pixel acquisition

Figure 9: The image of Seinfeld into 1b.jpg with Weighted average for pixel acquisition

Figure 10: The image of Seinfeld into 1c.jpg with Weighted average for pixel acquisition

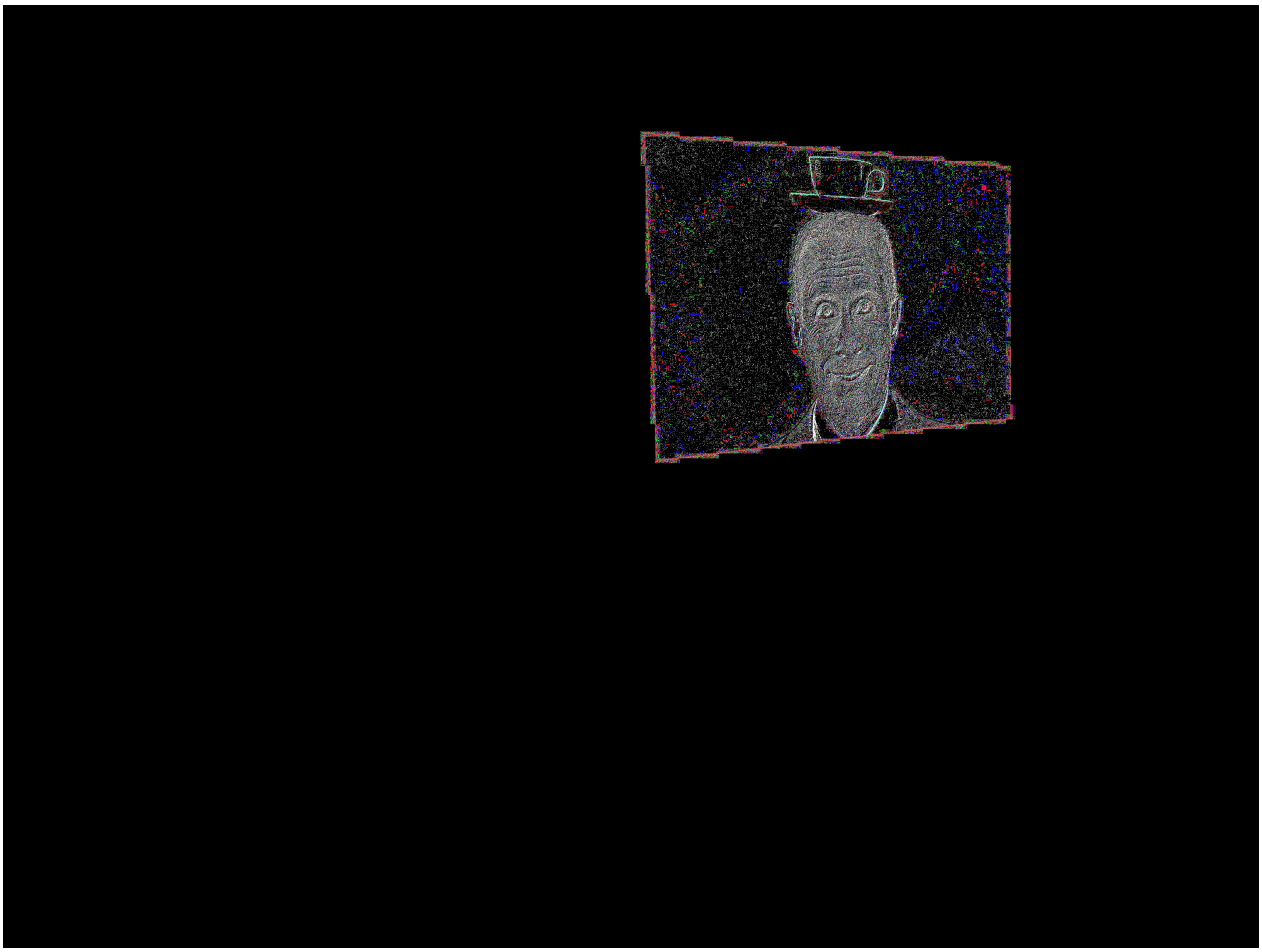

Figure 11: The Difference image between Figure 5 and Figure 8 to emphasis the significance of weighted average.

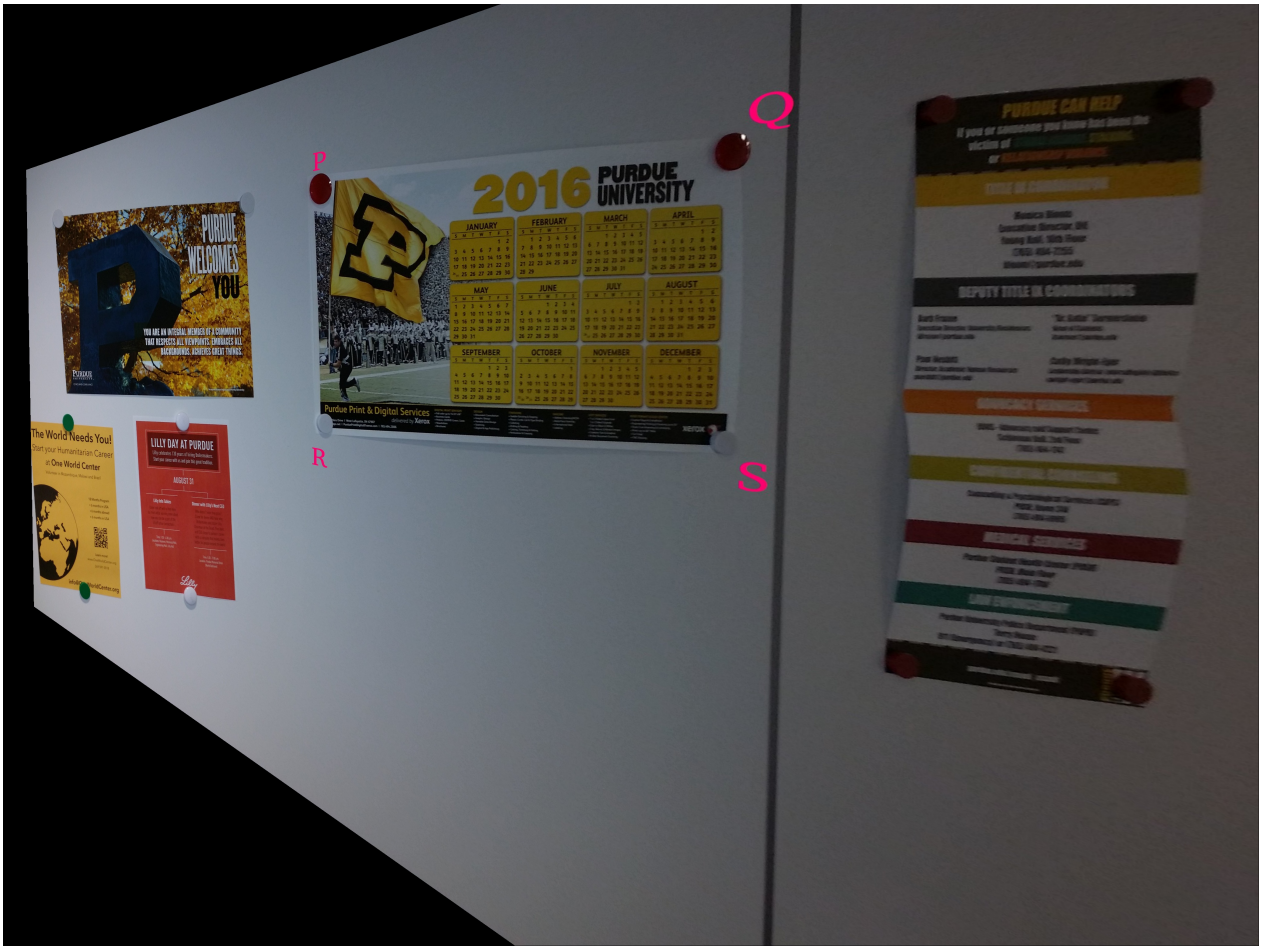

Figure 12: The image that represents 1a to 1c with Weighted average for pixel acquisition

Figure 13: The reference image to project my face taken from left view

Figure 14: The reference image to project my face taken from straight view

Figure 15: The reference image to project my face taken from Right view

Figure 16: Face being projected into the Left view of the Notice Board

Figure 17: Face being projected into the Straight view of the Notice Board

Figure 18: Face being projected into the Right view of the Notice Board

Figure 19: Left view of the notice board is transformed into the Right view of the Notice Board with 12 pts for Calculating Homography

2.3 Task 3 Repeating Task 1 and 2 on my images

The task 1 and task 2 are implemented with weighted average for pixel acquisition on the images. But for the Task for Changing the view from Left to right the no. of points are increased to 12 from 4, since the transformed image was highly distorted when compared with Right view of the notice board.

3 Source Code

3.1 Task-1: Without Weighted average

```
import cv2
import numpy as np
import math
path = "/home/vishwa/661/PicsHw2"

image_1 = cv2.imread(path+"/1.jpg")
image_2 = cv2.imread(path+"/2.jpg")
image_3 = cv2.imread(path+"/3.jpg")
image_4 = cv2.imread(path+"/Seinfeld.jpg")

# Homogeneous points obtained from gimp
Points_1a=np.array([[421,2108,1],[531,3303,1],[1495,2148,1],
[1357,3312,1]])
```

```
Points_1b=np.array([[795,1590,1],[768,2983,1],[1597,1616,1],
[1518,2987,1]])
Points_1c=np.array([[562,999,1],[421,2425,1],[1412,1026,1],
[1478,2406,1]])
Points_1d=np.array([[0,0,1],[0,2560,1],[1536,0,1],[1536,2560,1]])

# Finding Homography A of Ax=b
def Homography(points_src,points_dest):
    Mat_A=np.zeros((8,8))
    b = np.zeros((1,8))
    if points_src.shape[0] != points_dest.shape[0] or points_src.shape[1]
    != points_dest.shape[1]:
        print "No. of Source and destination points donot match"
        exit(1)
    for i in range(0,len(points_src)):
        Mat_A[i*2]=[points_src[i][0],points_src[i][1],points_src[i][2],
        0,0,0,(-1*points_src[i][0]*points_dest[i][0]),(-1*points
        _src[i][1]*points_dest[i][0])]
        Mat_A[i*2+1]=[0,0,0,points_src[i][0],points_src[i][1],points_src
        [i][2],(-1*points_src[i][0]*points_dest[i][1]),(-1*points_src[i]
        [1]*points_dest[i][1])]
        b[0][i*2] = points_dest[i][0]
        b[0][i*2+1] = points_dest[i][1]
    # A, b matrix formed

    # if no. of points is not 4 then using the below code
    tmp_H=np.dot(np.linalg.pinv(Mat_A),b.T)
    homography= np.zeros((3,3))
    homography[0]= tmp_H[0:3,0]
    homography[1]= tmp_H[3:6,0]
    homography[2][0:2]= tmp_H[6:8,0]
    homography[2][2]= 1
    return homography
# Function for finding Homography ends here

# image mapping code starts here
def image_mapping(src_image,dest_image,points_src,Homography):
    tmp_srcimage=np.zeros((src_image.shape[0],src_image.shape[1],3),
    dtype='uint8')
    pts = np.array([[points_src[0][1],points_src[0]
    [0]],[points_src[1][1],points_src[1][0]],[points_src[3][1],
    points_src[3][0]],[points_src[2][1],points_src[2][0]]])
    cv2.fillPoly(tmp_srcimage,[pts],(255,255,255))
    for i in range(0,(src_image.shape[0]-1)):
        for j in range(0,(src_image.shape[1]-1)):
            if tmp_srcimage[i,j,1]==255 and tmp_srcimage[i,j,0]==255 and
            tmp_srcimage[i,j,2]==255:
                point_tmp = np.array([i, j, 1])
```

```
                trans_coord = np.array(np.dot(Homography,point_tmp))
                trans_coord = trans_coord/trans_coord[2]
                if (trans_coord[0]>0) and (trans_coord[0]< dest_image.shape
                [0])and (trans_coord[1]>0) and (trans_coord[1]<dest_image.
                shape[1]):
                    src_image[i][j]=dest_image [math.floor(trans_coord[0]),
                    math.floor(trans_coord[1])]
            else:
                continue
    return src_image
# image mapping code ends here.

# Transforming the image in 1d to 1a
H=Homography(Points_1a,Points_1d)
output=image_mapping(image_1,image_4,Points_1a,H)
cv2.imwrite('final_image1.jpg',output)

# Transforming the image in 1d to 1b
H=Homography(Points_1b,Points_1d)
output=image_mapping(image_2,image_4,Points_1b,H)
cv2.imwrite('final_image2.jpg',output)

# Transforming the image in 1d to 1c
H=Homography(Points_1c,Points_1d)
output=image_mapping(image_3,image_4,Points_1c,H)
cv2.imwrite('final_image3.jpg',output)
```

3.2 Task-1: With Weighted average transformation

```
import cv2
import numpy as np
import math
path = "/home/vishwa/661/PicsHw2"

image_1 = cv2.imread(path+"/1.jpg")
image_2 = cv2.imread(path+"/2.jpg")
image_3 = cv2.imread(path+"/3.jpg")
image_4 = cv2.imread(path+"/Seinfeld.jpg")

# Homogeneous points obtained from gimp
Points_1a=np.array([[421,2108,1],[531,3303,1],[1495,2148,1],
[1357,3312,1]])
Points_1b=np.array([[795,1590,1],[768,2983,1],[1597,1616,1],
[1518,2987,1]])
Points_1c=np.array([[562,999,1],[421,2425,1],[1412,1026,1],
[1478,2406,1]])
Points_1d=np.array([[0,0,1],[0,2560,1],[1536,0,1],[1536,2560,1]])
```

```
# Finding Homography A of Ax=b
def Homography(points_src,points_dest):
    Mat_A=np.zeros((8,8))
    b = np.zeros((1,8))
    if points_src.shape[0] != points_dest.shape[0] or points_src.shape[1]
    !=points_dest.shape[1]:
        print "No. of Source and destination points donot match"
        exit(1)
    for i in range(0,len(points_src)):
        Mat_A[i*2]=[points_src[i][0],points_src[i][1], points_src [i][2],
        0,0,0,(-1*points_src[i][0]*points_dest[i][0]),(-1*points_src[i]
        [1]*points_dest[i][0])]
        Mat_A[i*2+1]=[0,0,0,points_src[i][0],points_src[i][1],points_
        src[i][2],(-1*points_src[i][0]*points_dest[i][1]),(-1*points_
        src[i][1]*points_dest[i][1])]
        b[0][i*2] = points_dest[i][0]
        b[0][i*2+1] = points_dest[i][1]
    # A, b matrix formed

    # if no. of points is not 4 then using the below code
    tmp_H=np.dot(np.linalg.pinv(Mat_A),b.T)
    homography= np.zeros((3,3))
    homography[0]= tmp_H[0:3,0]
    homography[1]= tmp_H[3:6,0]
    homography[2][0:2]= tmp_H[6:8,0]
    homography[2][2]= 1
    return homography
# Function for finding Homography ends here

# Function to get the RGB pixel data using weighted average starts here
def getdata(point, img):
    tp_left =img[(math.floor(point[0])),
    (math.floor(point[1]))]
    tp_right =img [math.floor(point[0]), math.floor(point[1]+1))]
    bt_left =img[math.floor(point[0]+1),math.floor(point[1])]
    bt_right  =img [math.floor(point[0]+1) ,math.floor(point[1]+1)]
    diff_x = point[0] - math.floor(point[0])
    diff_y = point[1] - math.floor(point[1])
    tp_left_weight= pow(pow(diff_x,2)+pow(diff_y,2),-0.5)
    tp_right_weight = pow(pow(diff_x,2)+pow(1-diff_y,2),-0.5)
    bt_left_weight = pow(pow(1-diff_x,2)+pow(diff_y,2),-0.5)
    bt_right_weight = pow(pow(1-diff_x,2)+pow(1-diff_y,2),-0.5)
    resultant_pt = (tp_left*tp_left_weight+ tp_right*tp_right_weight
    +bt_left*bt_left_weight +bt_right*bt_right_weight)/
    (tp_left_weight+tp_right_weight +bt_left_weight+bt_right_weight)
    return resultant_pt
# Function to get the RGB pixel data using weighted average ends here
```

```
# image mapping code starts here
def image_mapping(src_image,dest_image,points_src,Homography):
    tmp_srcimage=np.zeros((src_image.shape[0],src_image.shape[1],3),dtype
    ='uint8')
    pts = np.array([[points_src[0][1],points_src[0][0]],[points_src[1]
    [1],points_src[1][0]],[points_src[3][1],points_src[3][0]],[points_
    src[2][1],points_src[2][0]]])cv2.fillPoly(tmp_srcimage,[pts],
    (255,255,255))
    src_image2 = src_image
    for i in range(0,(src_image.shape[0]-1)):
        for j in range(0,(src_image.shape[1]-1)):
            if tmp_srcimage[i,j,1]==255 and tmp_srcimage[i,j,0]==255 and
            tmp_srcimage[i,j,2]==255:
                point_tmp = np.array([i, j, 1])
                trans_coord = np.array(np.dot(Homography,point_tmp))
                trans_coord = trans_coord/trans_coord[2]
                if (trans_coord[0]>0) and (trans_coord[0]< dest_image.
                shape[0]-1)and (trans_coord[1]>0) and (trans_coord[1]
                <dest_image.shape[1]-1):
                 src_image[i][j]=getdata(trans_coord,dest_image)

            else:
                continue
    return src_image
# image mapping code ends here.

# Transforming the image in 1d to 1a
H=Homography(Points_1a,Points_1d)
output=image_mapping(image_1,image_4,Points_1a,H)
cv2.imwrite('test_image1_bil.jpg',output)

# Transforming the image in 1d to 1b
H=Homography(Points_1b,Points_1d)
output=image_mapping(image_2,image_4,Points_1b,H)
cv2.imwrite('test_image2_bil.jpg',output)

# Transforming the image in 1d to 1c
H=Homography(Points_1c,Points_1d)
output=image_mapping(image_3,image_4,Points_1c,H)
cv2.imwrite('test_image3_bil.jpg',output)
```

Task-2: With weighted average for pixel value

```
import cv2
import numpy as np
import math
#from Hw_VJ import Homography as Hm
```

```
#from Hw_VJ import image_mapping as Im
path = "/home/vishwa/661/PicsHw2"

# Function for finding Homography starts here
def Homography(points_src,points_dest):
    Mat_A=np.zeros((8,8))
    b = np.zeros((1,8))
    if points_src.shape[0] != points_dest.shape[0] or points_src.
    shape[1] != points_dest.shape[1]:
        print "No. of Source and destination points donot match"
        exit(1)
    for i in range(0,len(points_src)):
        Mat_A[i*2]=[points_src[i][0],points_src[i][1],points_src[i][2],
        0,0,0,(-1*points_src[i][0]*points_dest[i][0]),(-1*points_src[i]
        [1]*points_dest[i][0])]
        Mat_A[i*2+1]=[0,0,0,points_src[i][0],points_src[i][1],points_
        src[i][2],(-1*points_src[i][0]*points_dest[i][1]),(-1*points_
        src[i][1]*points_dest[i][1])]
        b[0][i*2] = points_dest[i][0]
        b[0][i*2+1] = points_dest[i][1]
    # A, b matrix formed

    # If no. of points is not 4 then using the below code
    tmp_H=np.dot(np.linalg.pinv(Mat_A),b.T)
    homography= np.zeros((3,3))
    homography[0]= tmp_H[0:3,0]
    homography[1]= tmp_H[3:6,0]
    homography[2][0:2]= tmp_H[6:8,0]
    homography[2][2]= 1
    return homography
# Function for finding Homography ends here
# Function for getting RGB data using weighted average starts here
def getdata(point, img):
    tp_left =img[(math.floor(point[0])),(math.floor(point[1]))]
    tp_right =img[math.floor(point[0]),math.floor(point[1]+1)]
    bt_left =img[math.floor(point[0]+1),math.floor(point[1])]
    bt_right =img[math.floor(point[0]+1),math.floor(point[1]+1)]
    diff_x = point[0] - math.floor(point[0])
    diff_y = point[1] - math.floor(point[1])
    tp_left_weight= pow(pow(diff_x,2)+pow(diff_y,2),-0.5)
    tp_right_weight = pow(pow(diff_x,2)+pow(1-diff_y,2),-0.5)
    bt_left_weight = pow(pow(1-diff_x,2)+pow(diff_y,2),-0.5)
    bt_right_weight = pow(pow(1-diff_x,2)+pow(1-diff_y,2),-0.5)
    resultant_pt = (tp_left*tp_left_weight+tp_right*tp_right_weight
    +bt_left*bt_left_weight+bt_right*bt_right_weight)
    /(tp_left_weight+tp_right_weight+bt_left_weight+bt_right_weight)
    return resultant_pt
# Function for getting RGB data using weighted average ends here
```

```
# Image mapping code starts here
def image_mapping(src_image,dest_image,points_src,Homography):
    tmp_srcimage=np.zeros((src_image.shape[0],src_image.shape[1],3),
    dtype='uint8')
    for i in range(0,src_image.shape[0]-1):
        for j in range(0,src_image.shape[1]-1):
                point_tmp = np.array([i, j, 1])
                trans_coord = np.array(np.dot(Homography,point_tmp))
                trans_coord = trans_coord/trans_coord[2]
                if (trans_coord[0]>0) and (trans_coord[0]<dest_image.
                shape[0]-1) and (trans_coord[1]>0) and (trans_coord[1]
                <dest_image.shape[1]-1):
                    tmp_srcimage[i][j]=getdata(trans_coord,dest_image)
    return tmp_srcimage
# Image mapping code ends here.

image_1 = cv2.imread(path+"/1.jpg")
image_2 = cv2.imread(path+"/2.jpg")
image_3 = cv2.imread(path+"/3.jpg")
image_4 = cv2.imread(path+"/Seinfeld.jpg")

# Homogeneous points obtained from gimp
Points_1a=np.array([[421,2108,1],[531,3303,1],[1495,2148,1],
[1357,3312,1]])
Points_1b=np.array([[795,1590,1],[768,2983,1],[1597,1616,1],
[1518,2987,1]])
Points_1c=np.array([[562,999,1],[421,2425,1],[1412,1026,1],
[1478,2406,1]])
#Points_1d=np.array([[0,0,1],[0,2560,1],[1536,0,1],[1536,2560,1]])

H1=Homography(Points_1b,Points_1a)
H2=Homography(Points_1c,Points_1b)

Final_Homography=np.dot(H1,H2)
points_for_trans=np.array([[0,0],[image_3.shape[0],0],
[0,image_3.shape[1]],[image_3.shape[0],image_3.shape[1]]])
final_output= image_mapping(image_3,image_1,points_for_trans,
Final_Homography)
cv2.imwrite('1c_transform.jpg',final_output)
```

The source code of Task-3 is similar to one and two. Hence not repeated.