ECE 661 (Fall 2016) - Computer Vision - HW 2

Debasmit Das

September 6, 2016

Calculating homography

If we have a point p in a 2D plane and we want to project it into another plane to form point p', they are related by -

$$\mathbf{x}' = \mathbf{H}\mathbf{x} \tag{1}$$

where $\mathbf{x} = (x_1, x_2, x_3)^T$ and $\mathbf{x}' = (x_1', x_2', x_3')^T$ are the points represented in homogeneous co-ordinates.Here $\mathbf{H} \in \mathbb{R}^{3\times3}$ and let -

$$\mathbf{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tag{2}$$

We need to find 9 parameters of which 8 are free. We can set $h_{33} = 1$ since in $\mathbf{H}$ we are concerned with ratios. Since, we are concerned about the physical co-ordinates (x, y) and (x', y'), where $x = \frac{x_1}{x_3}$, $y = \frac{x_2}{x_3}$, $x' = \frac{x_1'}{x_3'}$, $y' = \frac{x_1'}{x_3'}$. Applying equation (1) we get -

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + 1} \tag{3}$$

$$y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + 1} \tag{4}$$

So, we get 2 equations for a single point and we have 8 unknowns. So, in total we need 8 equations. Therefore 4 points are required for calculating the homography uniquely. So we repeat equation (3) and (4) for a total four points we he rearrange the equations such that -

$$\overbrace{\begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4' \\ y_4' \end{bmatrix}}^{\mathbf{t}} = \overbrace{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_3' & -y_3y_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y_4' & -y_4y_4' \end{bmatrix}}^{\mathbf{P}} \overbrace{\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix}}^{\mathbf{h}} \tag{5}$$

The equation can be solved using $\mathbf{h} = \mathbf{P}^{-1}\mathbf{t}$

Pixel RGB color calculating method

When we use the homography matrix to map point p to p', it may occur that the points (x', y'), will not be integral. But pixels are stored as integers in an image. So in case of non-integer pixels we use the weighted average of neighboring pixel values to find the pixel value of (x', y'). The neighboring pixels are chosen as $p_1 : (x_1, y_1)$, $p_2 : (x_2, y_2)$, $p_3 : (x_1, y_2)$ and $p_4 : (x_2, y_1)$ where x_1 is the highest integer less than or equal to x', x_2 is the lowest integer greater than or equal to x', where y_1 is the highest integer is less than or equal to y' and y_2 is the lowest integer greater than or equal to y'. So, the value of the pixel at (x', y') is given by -

$$f(x', y') = \frac{d(p', p_1)f(p_1) + d(p', p_2)f(p_2) + d(p', p_3)f(p_3) + d(p', p_4)f(p_4)}{d(p', p_1) + d(p', p_2) + d(p', p_3) + d(p', p_4)} \tag{6}$$

where $d(p', p_i)$ is the euclidean distance between the points p' and p_i, $i \in \{1, 2, 3, 4\}$

Now, we will describe the results of Task 1,2,3 and comment on them.

Tasks

For Task 1 and Task 2 the input images are as follows-

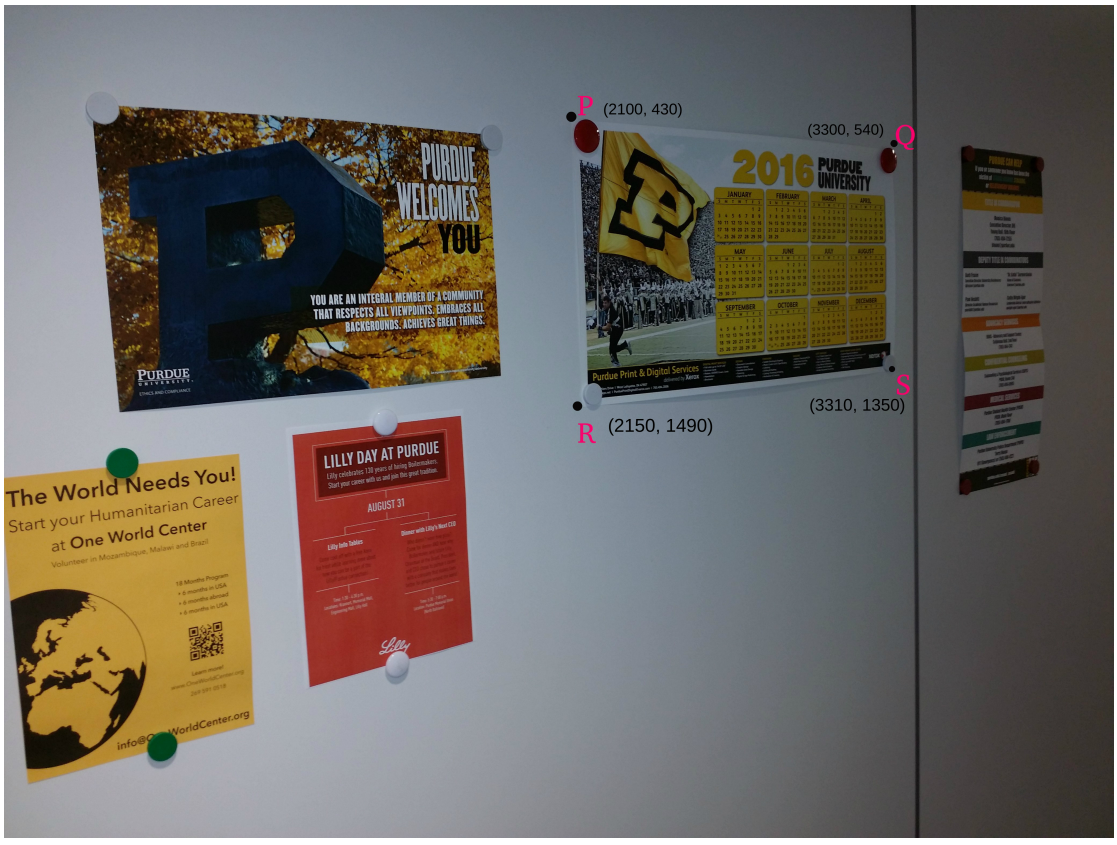

Figure 1: Destination Plane 1 (PQRS) with marked pixels

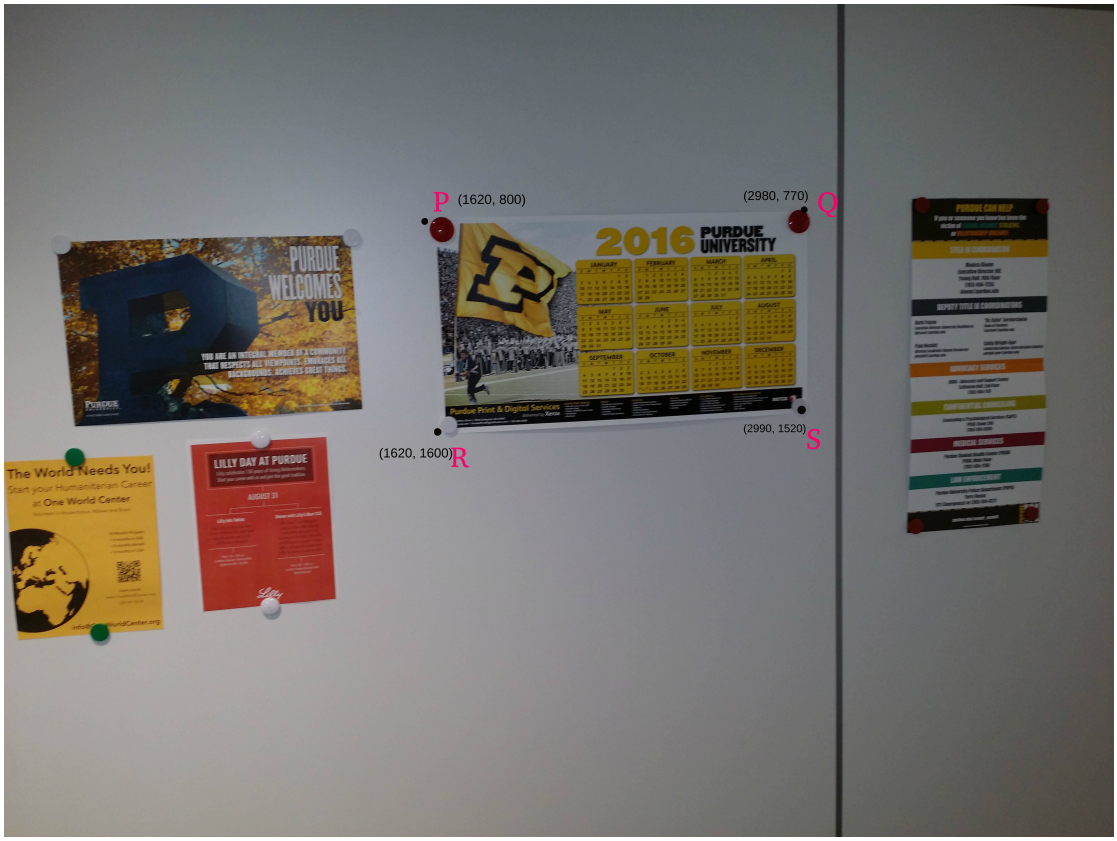

Figure 2: Destination Plane 2 (PQRS) with marked pixels

Figure 3: Destination Plane 3 (PQRS) with marked pixels

Figure 4: Source Image (Seinfeld)

Task1

For task 1, our goal is to project Fig. 4 on the PQRS regions of Fig. 1, Fig. 2 and Fig. 3 respectively. The steps are as follows -

- In this step, we find the homography matrix $\mathbf{H}$ such that $\mathbf{x}' = \mathbf{Hx}$ where $\mathbf{x}'$ would be the 4 corners of Fig. 4 (top-left, top-right, bottom-left, bottom-right) and $\mathbf{x}$ would be the 4 points P,Q,R,S in each Fig. 1, 2, 3 respectively. The method used is described in the section *Calculating Homography*
- Once the homography $\mathbf{H}$ is calculated, we need to map RGB values for all pixels. For that, we loop over all the pixels of the destination image and change only points within the bounding box PQRS. The point in the bounding box is mapped to the source image using $\mathbf{H}$ to retrieve the color of the pixel. In case the mapped pixel co-ordinate is not an integer, we use the method described in *Pixel RGB color calculating method*.

The result is shown below.

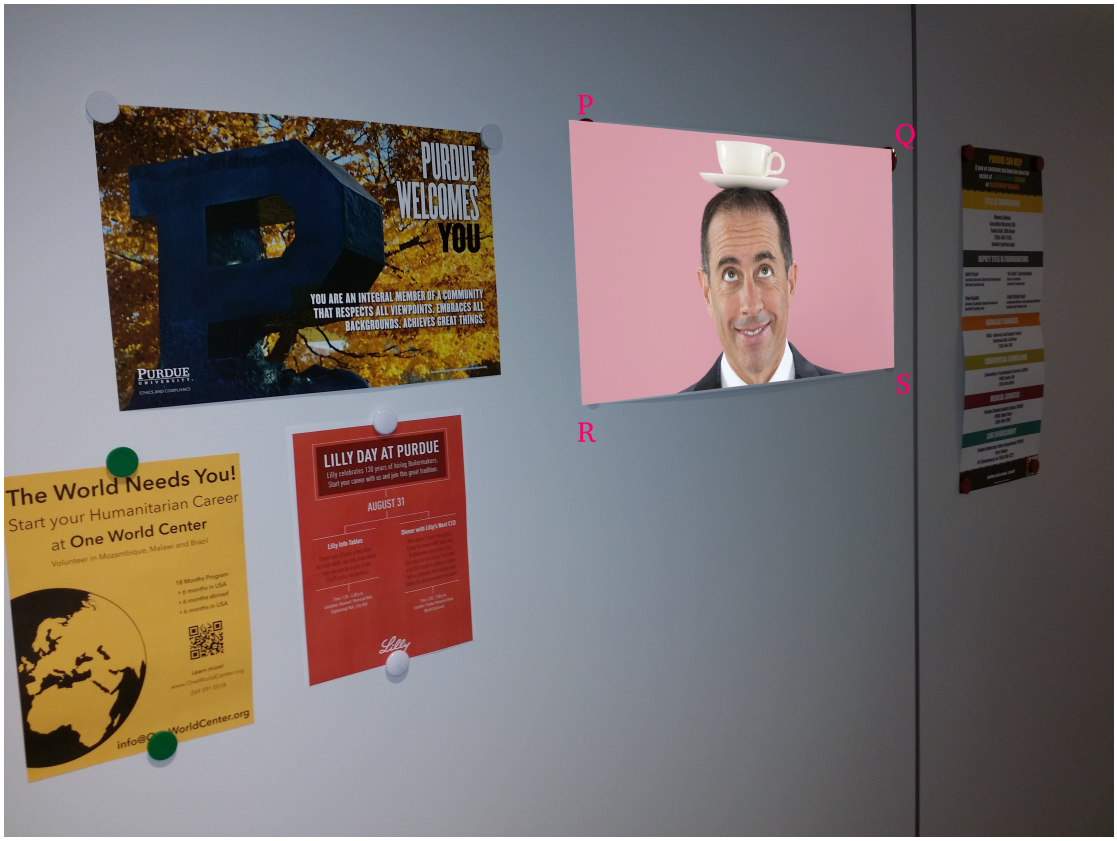

Figure 5: Result on Plane 1

Figure 6: Result on Plane 2

Figure 7: Result on Plane 3

Task2

This involved projecting plane 1 to plane 3. The following steps were involved-

- Firstly, we find $\mathbf{H_1}$ such that $\mathbf{x''} = \mathbf{H}_1\mathbf{x'}$ where $\mathbf{x''}$, $\mathbf{x'}$ are HC representation of PQRS in Fig 1. and Fig.2 respectively.
- Secondly, we find $\mathbf{H_2}$ such that $\mathbf{x'} = \mathbf{H}_2\mathbf{x}$ where $\mathbf{x'}$, $\mathbf{x}$ are HC representation of PQRS in Fig 2. and Fig.3 respectively.
- We find the composite homogeneous matrix $\mathbf{H} = \mathbf{H}_1\mathbf{H}_2$. Then, we loop over the pixel co-ordinates of the Fig. 3 and those points are mapped to Fig.1 using this composite homogeneous matrix $\mathbf{H}$ to retrieve the color of the pixel. In case of non-integral co-ordinates we use the method described in *Pixel RGB color calculating method*.

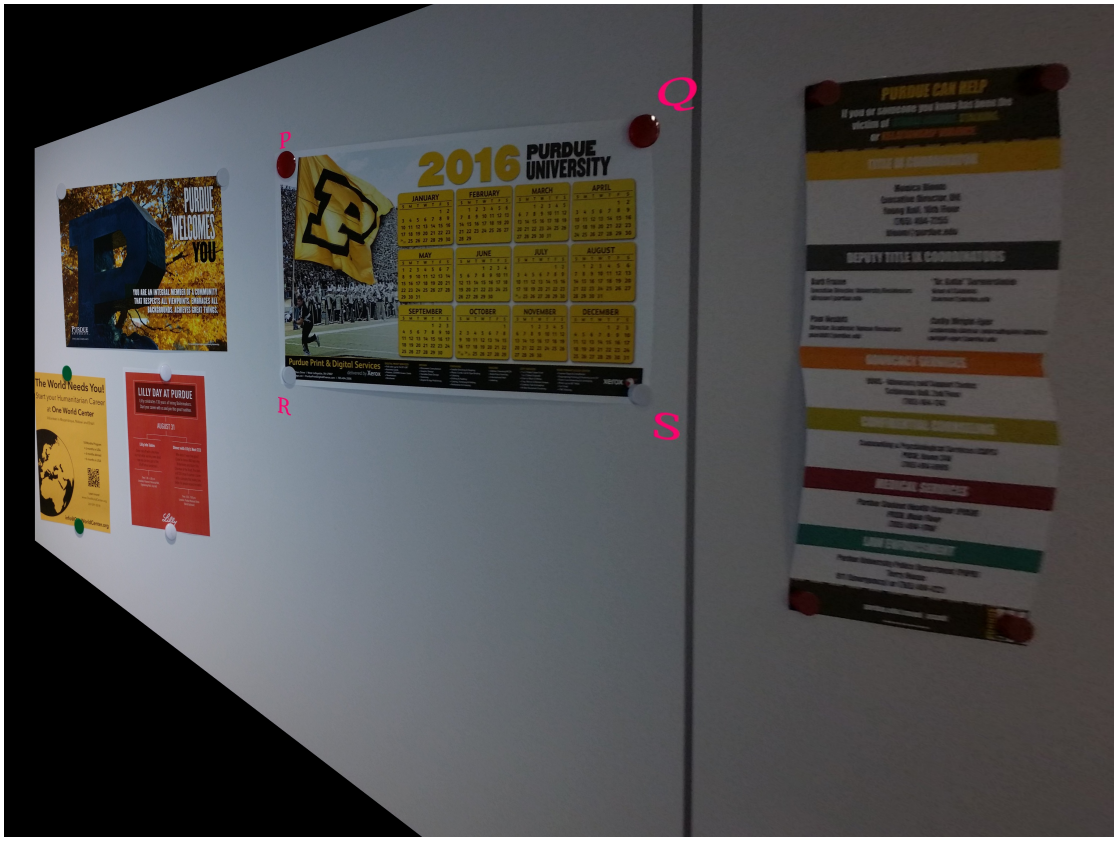

Figure 8: Result of Projecting Plane 1 on Plane 3

Task3

For Task 3 the input images are as follows and it is a repeat of Task 1 and Task 2 on different images- So the logic used is same so I do not want to mention it again.

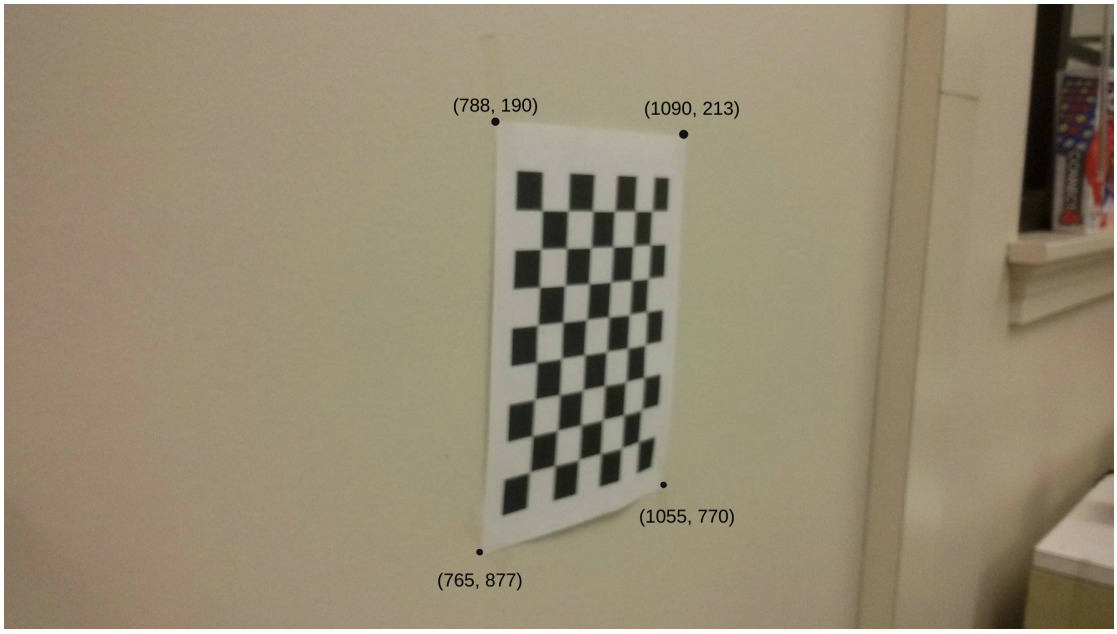

Figure 9: Destination Plane 1 (PQRS) with marked pixels

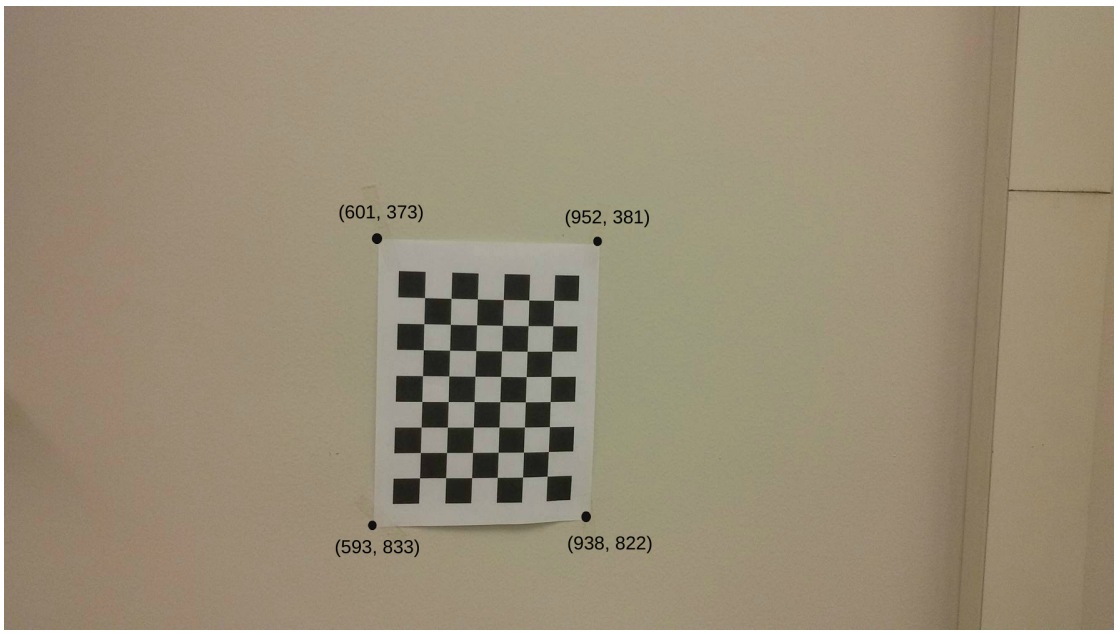

Figure 10: Destination Plane 2 (PQRS) with marked pixels

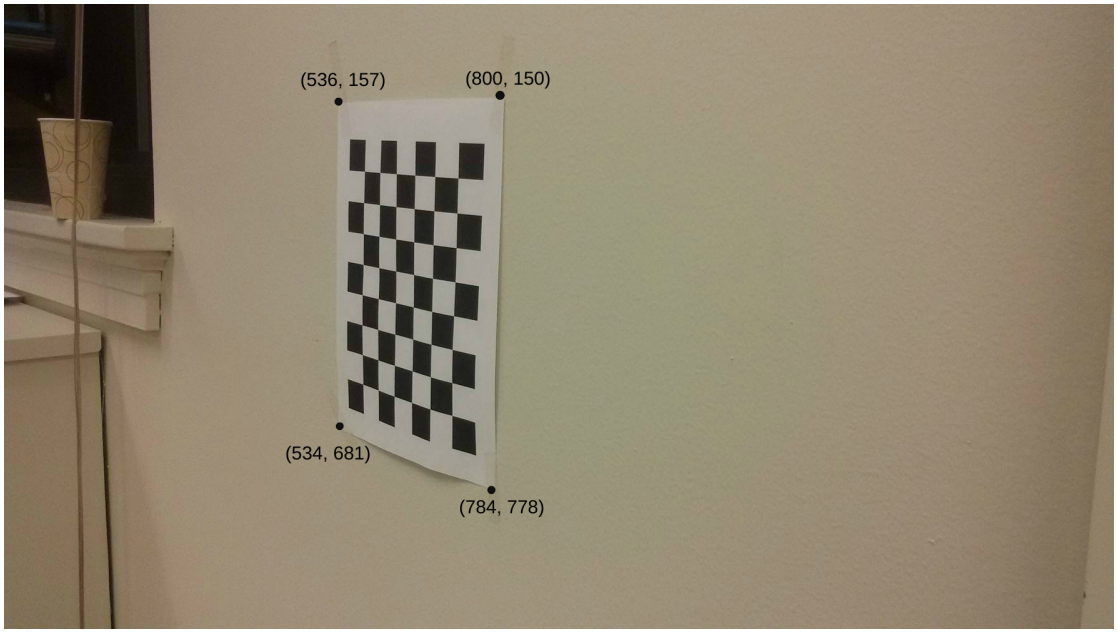

Figure 11: Destination Plane 3 (PQRS) with marked pixels

Figure 12: Source Image (Debasmit)

Results when repeating task 1

Results:

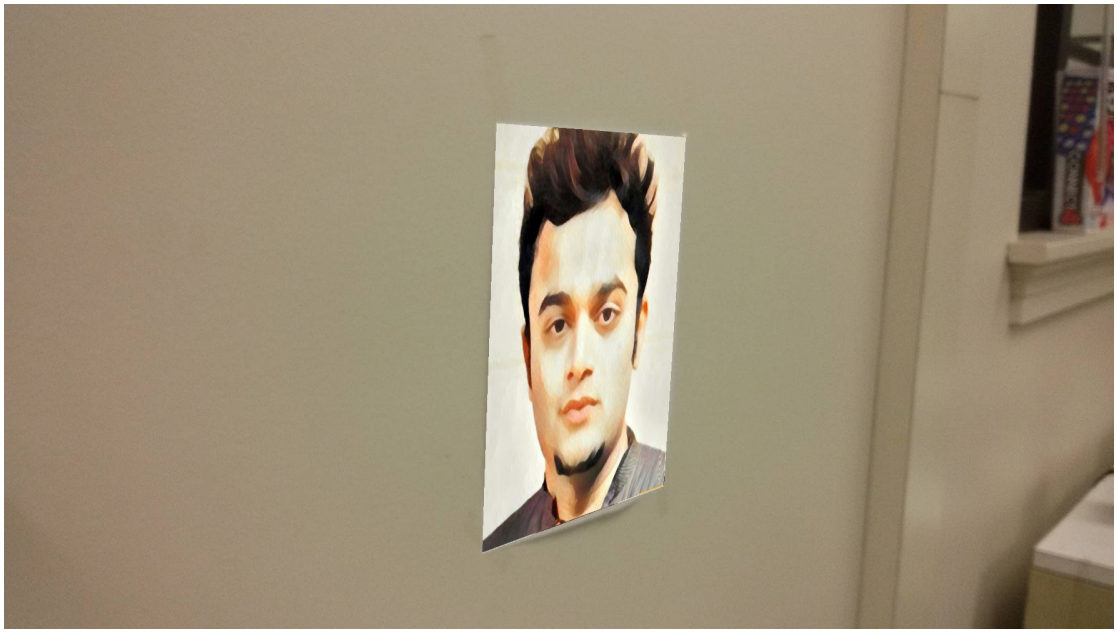

Figure 13: Result on Plane 1

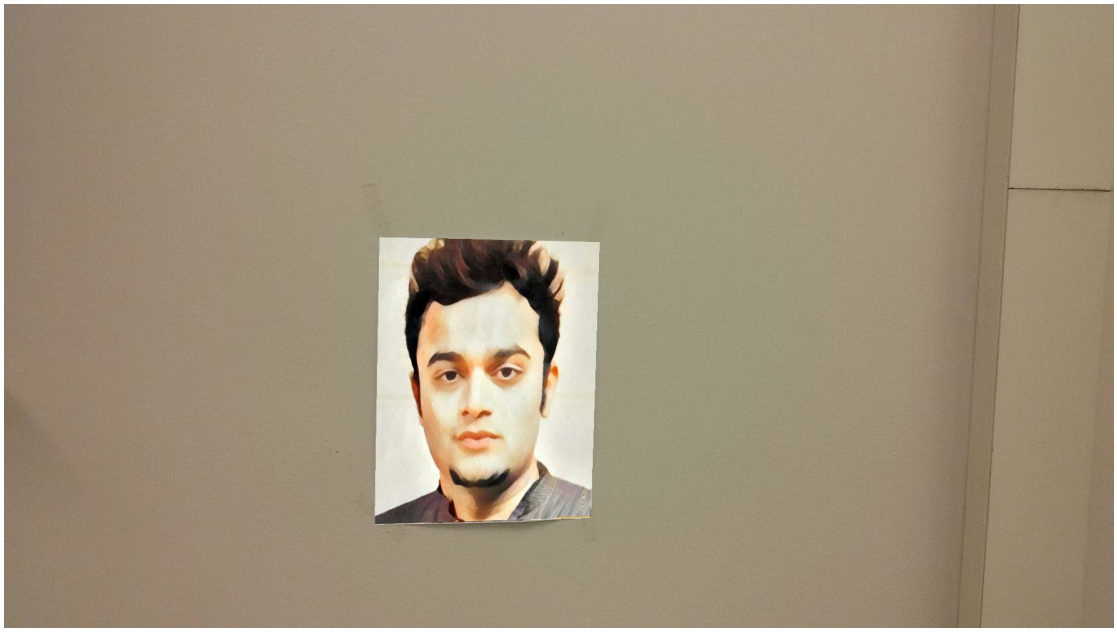

Figure 14: Result on Plane 2

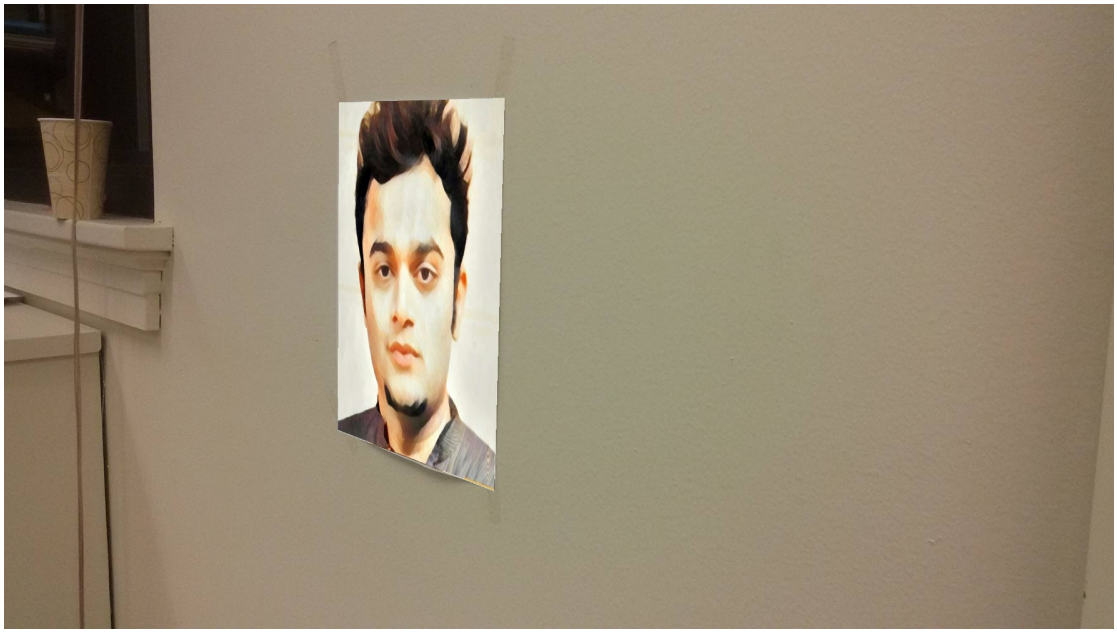

Figure 15: Result on Plane 3

Results when repeating task 2

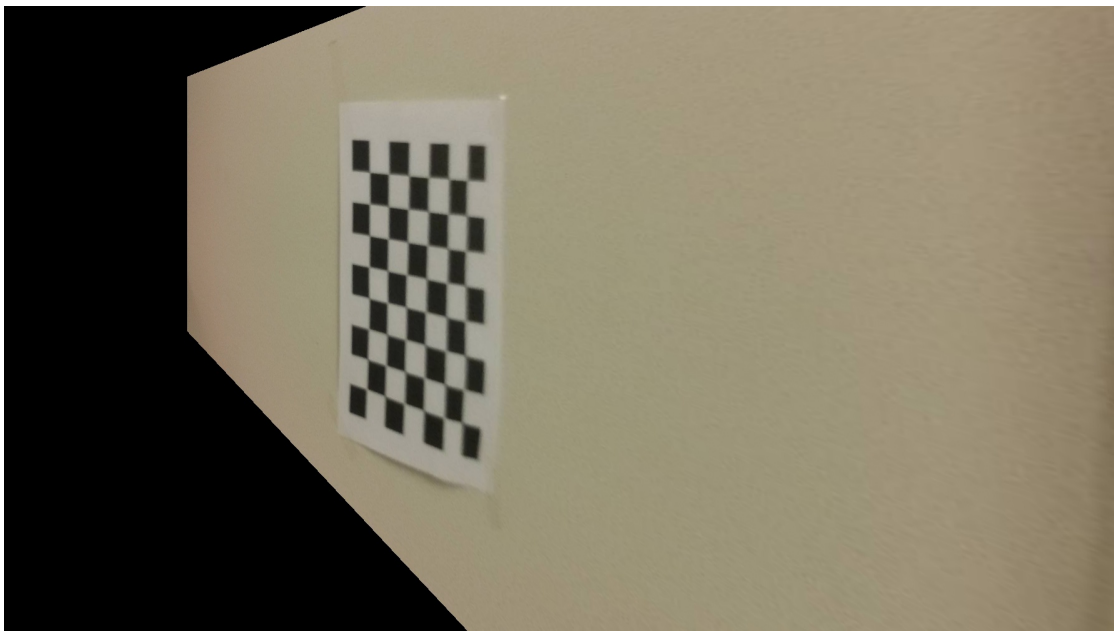

Figure 16: Result of Projecting Plane 1 on Plane3

Code

The script is in Python. Only Task 1 and Task 2 script is provided. Task 3 is just a repetition of Task 1 and Task 2 but with different input images.

Task1 Script

```
#Written  by Debasmit Das

#Useful modules are loaded
import numpy as np
import cv2
import math

#Applying task 1 for the 3rd image
img3=cv2.imread('3.jpg')
imgs=cv2.imread('Seinfeld.jpg')

#Creating a box where the source image is filled
img1fill= np.zeros((img3.shape[0],img3.shape[1],3),dtype='uint8')
pts = np.array([[1000,560],[2400,420],[2400,1480],[1030,1410]],np.int32)
pts = pts.reshape((-1,1,2))
cv2.fillPoly(img1fill,[pts],(255,255,255))

#function to get color values
def getrgb(pt, img):
    p = img[math.floor(pt[0,0]), math.floor(pt[0,1])]
    q = img[math.floor(pt[0,0]), math.ceil(pt[0,1])]
    r = img[math.ceil(pt[0,0]), math.floor(pt[0,1])]
    s = img[math.ceil(pt[0,0]), math.ceil(pt[0,1])]

    x = pt[0,0] - math.floor(pt[0,0])
    y = pt[0,1] - math.floor(pt[0,1])

    wp = 1/np.linalg.norm(np.array([x,y]))
    wq = 1/np.linalg.norm(np.array([x,1-y]))
    wr = 1/np.linalg.norm(np.array([1-x,y]))
    ws = 1/np.linalg.norm(np.array([1-x,1-y]))

    color = (p*wp+q*wq+r*wr+s*ws)/(wp+wq+wr+ws)

    return color
## rgb values of the pixels are found

#We have to interchange the x and y co-ordinates in implementation
#This is because of way images are stored in tensors
#PQRS co-ordinates of all the images are found
p3=np.matrix('560 1000 1', dtype=float)
q3=np.matrix('420 2400 1', dtype=float)
r3=np.matrix('1410 1030 1', dtype=float)
s3=np.matrix('1480 2400 1', dtype=float)
ps=np.matrix('0 0 1', dtype=float)
qs=np.matrix('0 2560 1', dtype=float)
rs=np.matrix('1536 0 1', dtype=float)
ss=np.matrix('1536 2560 1', dtype=float)
```

```
Ps1=np.matrix(np.zeros((8,8)), dtype=float)

#Declaring matrix values for mapping between source and destination
Ps1[0,0:3] = p3
Ps1[0,6:9] = -p3[0,0:2]*ps[0,0]
Ps1[1,3:6] = p3
Ps1[1,6:9] = -p3[0,0:2]*ps[0,1]
Ps1[2,0:3] = q3
Ps1[2,6:9] = -q3[0,0:2]*qs[0,0]
Ps1[3,3:6] = q3
Ps1[3,6:9] = -q3[0,0:2]*qs[0,1]
Ps1[4,0:3] = r3
Ps1[4,6:9] = -r3[0,0:2]*rs[0,0]
Ps1[5,3:6] = r3
Ps1[5,6:9] = -r3[0,0:2]*rs[0,1]
Ps1[6,0:3] = s3
Ps1[6,6:9] = -s3[0,0:2]*ss[0,0]
Ps1[7,3:6] = s3
Ps1[7,6:9] = -s3[0,0:2]*ss[0,1]

ts1 = np.matrix('0 0 0 0 0 0 0 0',dtype=float)
ts1[0,0:2] = ps[0,0:2]
ts1[0,2:4] = qs[0,0:2]
ts1[0,4:6] = rs[0,0:2]
ts1[0,6:8] = ss[0,0:2]

#Finding the Homography
param = np.transpose(np.linalg.inv(Ps1)*np.transpose(ts1))
H = np.zeros((3,3),dtype=float)
H[0] = param[0,0:3]
H[1] = param[0,3:6]
H[2,0:2] = param[0,6:8]
H[2,2] = 1

#Looping over the image
tmp = np.matrix('0 0 1',dtype=float)
for row in range(0,img3.shape[0]-1):
    for col in range(0,img3.shape[1]-1):
        if img1fill[row,col,1] > 0:
            tmp[0,0] = row
            tmp[0,1] = col
            tr = np.transpose(H*np.transpose(tmp))
            tr = tr/tr[0,2]
            if tr[0, 0] > 0 and tr[0, 1] > 0 and tr[0, 0] < imgs.shape[0] - 1 and
               tr[0, 1] < imgs.shape[1] - 1:
                img3[row,col]=getrgb(tr,imgs)


cv2.imwrite('Result.jpg',img3)
cv2.destroyAllWindows()
```

Task2 Script

```
#Written  by Debasmit Das

#Useful modules are loaded
import numpy as np
import cv2
import math

#The image files are read
img1=cv2.imread('1.jpg')
img2=cv2.imread('2.jpg')
img3=cv2.imread('3.jpg')

#The image file output is initialised
imgnew=np.zeros((img3.shape[0],img3.shape[1],3))

#function to get color values
def getrgb(pt, img):
    p = img[math.floor(pt[0,0]), math.floor(pt[0,1])]
    q = img[math.floor(pt[0,0]), math.ceil(pt[0,1])]
    r = img[math.ceil(pt[0,0]), math.floor(pt[0,1])]
    s = img[math.ceil(pt[0,0]), math.ceil(pt[0,1])]

    x = pt[0,0] - math.floor(pt[0,0])
    y = pt[0,1] - math.floor(pt[0,1])

    wp = 1/np.linalg.norm(np.array([x,y]))
    wq = 1/np.linalg.norm(np.array([x,1-y]))
    wr = 1/np.linalg.norm(np.array([1-x,y]))
    ws = 1/np.linalg.norm(np.array([1-x,1-y]))

    color = (p*wp+q*wq+r*wr+s*ws)/(wp+wq+wr+ws)

    return color
## rgb values of the pixels are found

#We have to interchange the x and y co-ordinates in implementation
#This is because of way images are stored in tensors
#PQRS co-ordinates of all the images are found
p1=np.matrix('430 2100 1', dtype=float)
q1=np.matrix('540 3300 1', dtype=float)
r1=np.matrix('1490 2150 1', dtype=float)
s1=np.matrix('1350 3310 1', dtype=float)
p2=np.matrix('800 1620 1', dtype=float)
q2=np.matrix('770 2980 1', dtype=float)
r2=np.matrix('1600 1620 1', dtype=float)
s2=np.matrix('1520 2990 1', dtype=float)
p3=np.matrix('560 1000 1', dtype=float)
q3=np.matrix('420 2400 1', dtype=float)
r3=np.matrix('1410 1030 1', dtype=float)
```

```
s3=np.matrix('1480 2400 1', dtype=float)

# H1 homography values
P12=np.matrix(np.zeros((8,8)), dtype=float)

# H2 homography values
P23=np.matrix(np.zeros((8,8)), dtype=float)

#Declaring matrix values for mapping between

P12[0,0:3] = p2; P23[0,0:3] = p3
P12[0,6:9] = -p2[0,0:2]*p1[0,0]; P23[0,6:9] = -p3[0,0:2]*p2[0,0]
P12[1,3:6] = p2; P23[1,3:6] = p3;
P12[1,6:9] = -p2[0,0:2]*p1[0,1] ; P23[1,6:9] = -p3[0,0:2]*p2[0,1]
P12[2,0:3] = q2 ; P23[2,0:3] = q3
P12[2,6:9] = -q2[0,0:2]*q1[0,0]; P23[2,6:9] = -q3[0,0:2]*q2[0,0]
P12[3,3:6] = q2 ; P23[3,3:6] = q3
P12[3,6:9] = -q2[0,0:2]*q1[0,1] ; P23[3,6:9] = -q3[0,0:2]*q2[0,1]
P12[4,0:3] = r2 ; P23[4,0:3] = r3
P12[4,6:9] = -r2[0,0:2]*r1[0,0] ; P23[4,6:9] = -r3[0,0:2]*r2[0,0]
P12[5,3:6] = r2 ; P23[5,3:6] = r3
P12[5,6:9] = -r2[0,0:2]*r1[0,1] ; P23[5,6:9] = -r3[0,0:2]*r2[0,1]
P12[6,0:3] = s2 ; P23[6,0:3] = s3
P12[6,6:9] = -s2[0,0:2]*s1[0,0] ; P23[6,6:9] = -s3[0,0:2]*s2[0,0]
P12[7,3:6] = s2 ; P23[7,3:6] = s3
P12[7,6:9] = -s2[0,0:2]*s1[0,1] ; P23[7,6:9] = -s3[0,0:2]*s2[0,1]



t12 = np.matrix('0 0 0 0 0 0 0 0',dtype=float) ; t23 = np.matrix('0 0 0 0 0 0 0 0',dtype=float)
t12[0,0:2] = p1[0,0:2] ; t23[0,0:2] = p1[0,0:2]
t12[0,2:4] = q1[0,0:2] ; t23[0,2:4] = q1[0,0:2]
t12[0,4:6] = r1[0,0:2] ; t23[0,4:6] = r1[0,0:2]
t12[0,6:8] = s1[0,0:2] ; t23[0,6:8] = s1[0,0:2]

#Finding the Homography
param1 = np.transpose(np.linalg.inv(P12)*np.transpose(t12)) ;
param2 = np.transpose(np.linalg.inv(P23)*np.transpose(t23))
H1 = np.zeros((3,3),dtype=float) ; H2 = np.zeros((3,3),dtype=float)
H1[0] = param1[0,0:3] ; H2[0] = param2[0,0:3]
H1[1] = param1[0,3:6] ; H2[1] = param2[0,3:6]
H1[2,0:2] = param1[0,6:8] ; H2[2,0:2] = param2[0,6:8]
H1[2,2] = 1 ; H2[2,2] = 1


#The composite homogeneous matrix is found
H=np.dot(H1,H2)


#Looping over the image
tmp = np.matrix('0 0 1',dtype=float)
for row in range(0,imgnew.shape[0]-1):
```

```
    for col in range(0,imgnew.shape[1]-1):
        #if img1fill[row,column,1] > 0:
            tmp[0,0] = row
            tmp[0,1] = col
            tr = np.transpose(H*np.transpose(tmp))
            tr = tr/tr[0,2]
            if tr[0,0]>0 and tr[0,1]>0 and tr[0,0]<imgnew.shape[0]-1 and tr[0,1]<imgnew.shape[1]-1 :
                imgnew[row,col]=getrgb(tr,img1)



cv2.imwrite('Result.jpg',imgnew)
cv2.destroyAllWindows()
```