Homework 2

Wen Yi

1. Homography Estimating Method

Given a point p in a planar scene and its corresponding pexel p' in the image plane, we can write

$$x' = Hx$$

Assuming that x and x' are expressed using homogeneous coordinates: $x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$ and $x' = \begin{pmatrix} x_1{}' \\ x_2{}' \\ x_3{}' \end{pmatrix}$. With $H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$, we can get

$$x_1{}' = h_{11}x_1 + h_{12}x_2 + h_{13}x_3$$
$$x_2{}' = h_{21}x_1 + h_{22}x_2 + h_{23}x_3$$
$$x_3{}' = h_{31}x_1 + h_{32}x_2 + h_{33}x_3$$

Denoting the physical scene coordinates by (x,y) and the physical pixel coordinates by (x',y'), we have $x = \frac{x_1}{x_3}, y = \frac{x_2}{x_3}$ and $x' = \frac{x_1{}'}{x_3{}'}, y' = \frac{x_2{}'}{x_3{}'}$.

So, we can get the equation:

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}, y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}$$

Which can be expressed as following form:

$$xh_{11} + yh_{12} + h_{13} - xx'^{h_{31}} - yx'h_{32} - x'h_{33} = 0$$
$$xh_{21} + yh_{22} + h_{23} - xy'^{h_{31}} - yy'^{h_{32}} - y'h_{33} = 0$$

Thus, a single point can brings two equation to the linear equation with unknown variables $(h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}, h_{33})$. As the ratio of the whole matrix H doesn't matter, we can set h_{33} to be 1, thus the equation become:

$$xh_{11} + yh_{12} + h_{13} - xx'^{h_{31}} - yx'^{h_{32}} = x'$$
$$xh_{21} + yh_{22} + h_{23} - xy'^{h_{31}} - yy'^{h_{32}} = y'$$

The unknown variables become $(h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32})$.

So, we need at least 8 equations, which means 4 points to solve this problem, while there shouldn't be any of the 3 from the 4 points stay in the same line.

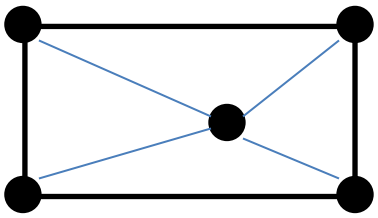
Thus, we can express the problem in Ax=b style like following:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & x_1x_1{}' & y_1x_1{}' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & x_1y_1{}' & y_1y_1{}' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & x_2x_2{}' & y_2x_2{}' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & x_2y_2{}' & y_2y_2{}' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & x_3x_3{}' & y_3x_3{}' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & x_3y_3{}' & y_3y_3{}' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & x_4x_4{}' & y_4x_4{}' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & x_4y_4{}' & y_4y_4{}' \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} x_1{}' \\ y_1{}' \\ x_2{}' \\ y_2{}' \\ x_3{}' \\ y_3{}' \\ x_4{}' \\ y_4{}' \end{bmatrix}$$

Then we can solve the equation with $x = A^{-1}b$, therefore reconstruct the H matrix.

2. Color Value Calculating Method

As mentioned above, for every physical scene coordinates (x,y), we have a physical pixel coordinates (x',y'). However, all these x, y value is a real number in the real life, while the image stored in computer only have pixels with the non-negative integer value of x and y. So, when drawing a new image of real world plane from a picture of scene viewing, we can get any pixel of the new image as $p_0' = (x_0', y_0')$, then get there corresponding position $p_0 = (x_0, y_0)$ in the scene viewing image by $p_0 = H^{-1}p_0'$. Note that (x_0, y_0) is a real number position. Then, in the scene viewing image, this position would be fitted in a box with one pixel of image at each corner (if the point is out of the image, then just let it be the background color).



So, the color on this position should be judged from the combination of the 4 corner pixel. Let's say the 4 pixels are $p_{11}, p_{12}, p_{21}, p_{22}$, the unknown point is p.

In the computation, we want the pixel which is closer to the position be heavier weighted that the far away pixels, so we take the reciprocal of the distance between the computed position and the pixel as the weight. The color of the computed position should be expressed as:

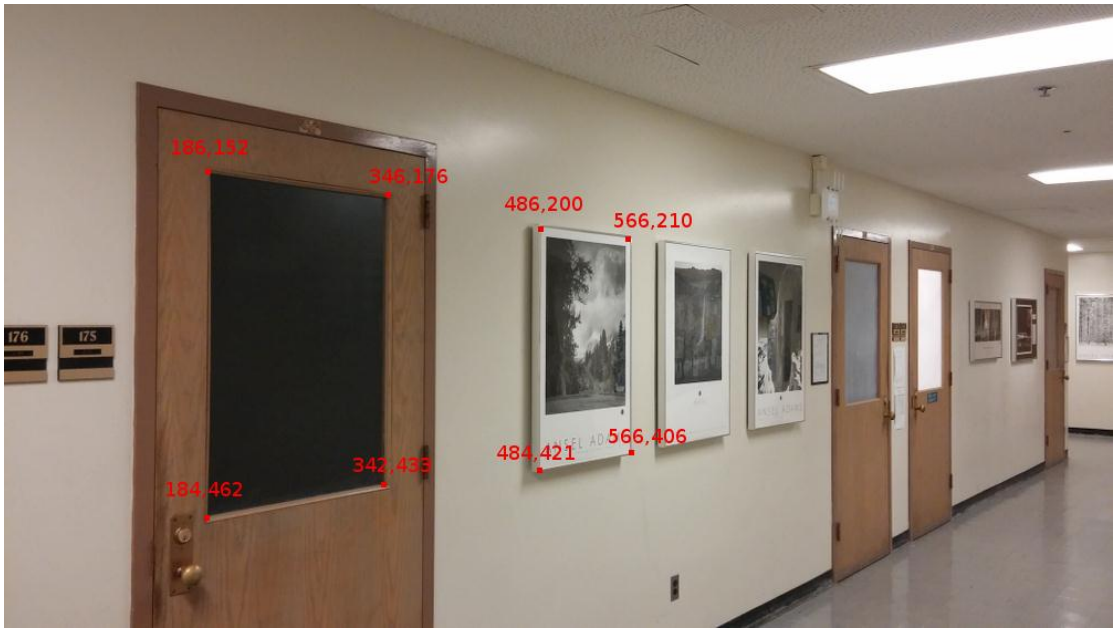
$$Color_p = \frac{\frac{Color_{11}}{Distance_{11}} + \frac{Color_{11}}{Distance_{11}} + \frac{Color_{11}}{Distance_{11}} + \frac{Color_{11}}{Distance_{11}}}{\frac{1}{Distance_{11}} + \frac{1}{Distance_{11}} + \frac{1}{Distance_{11}} + \frac{1}{Distance_{11}}}$$

When computing the pixel's color of a scene viewing image from a world plane image, we use the similar strategy. The only difference is that we compute the corresponding position in the world plane using $p_0' = Hp_0$.

3. Method Used for Each Task

3.0 Point Used in Each Task

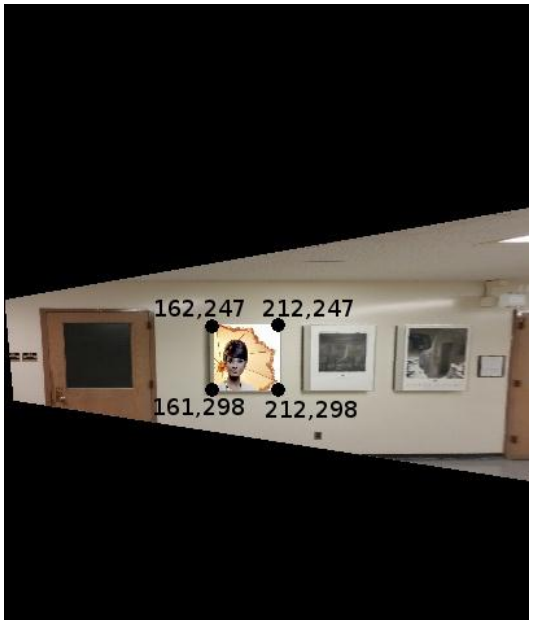
Point used in figure 1a: (horizon axis as x)



To have a better access to the image matrix, which always expressed in the order (row, column), we switch the x, y axis in the above picture in the code. For example, we express the point P **(186,152) as (152, 186, 1)** in the code.

Point used in figure 1b: the 4 corner of the image, which is (0 0 1), (0 img.shape[1] 1), (img.shape[0] 0 1), (img.shape[0] img.shape[1] 1) in the code.

Point used in the output of Task 2:



Point used of real world axis in Task 3:
(0,0,1) (0,width,1) (height,0,1) (height,width,1)

Point used in self-image for Task 1:

Point used in self-image in Task 2: the 4 corner of the image

Point used in self-image in Task 3 part 2:

The width-height ratio for the paint frame in real world is 18:10, so I use (0,0,1) (0,180,1) (100,0,1) (100,180,1)

3.1 Task 1:

Step 1: Create a black image *Region* that has the same size as figure 1a, then draw a filled white polygon using the 4 points. This image will help with projecting the face on to the frame.

Step 2: Use P,Q,R,S as the point in the scene viewing image, use the 4 corner of figure 1b as the corresponding position in the world plane, compute H.

Step 3: For each pixel p in the white region in *Region*, compute the corresponding position in figure 1b using H, then compute the color of that position using the position and the color of the 4 pixel that surround the position, then fill the corresponding pixel p in figure 1a with the result color.

3.2 Task 2:

Step 1: Calculate H using A,B,C,D in figure 1a and the 4 corner in figure 1b.

Step 2: Compute the position in world plane for the 4 corner of figure 1a, then compute the new image size using the length on x-axis and y-axis. Then set ((-1)*most-left-x-position, (-1)*most-up-y-position) as the shift offset for all points, so that we can see all the result pixels on the new image.

Step 3: For all the pixels in the new image, let them minus the offset, then using H to find the corresponding position in figure 1a, then compute the color of that position using the above calculating method.

Step 4: Shift the position of all pixels in figure 1b, then paste it onto the new image.

3.3 Task 3:

Step 1: Calculate H using the position of the 4 corners of the frame and the real position of the frame as (0,0,1) (0,width,1) (height,0,1) (height,width,1).

Step 2:

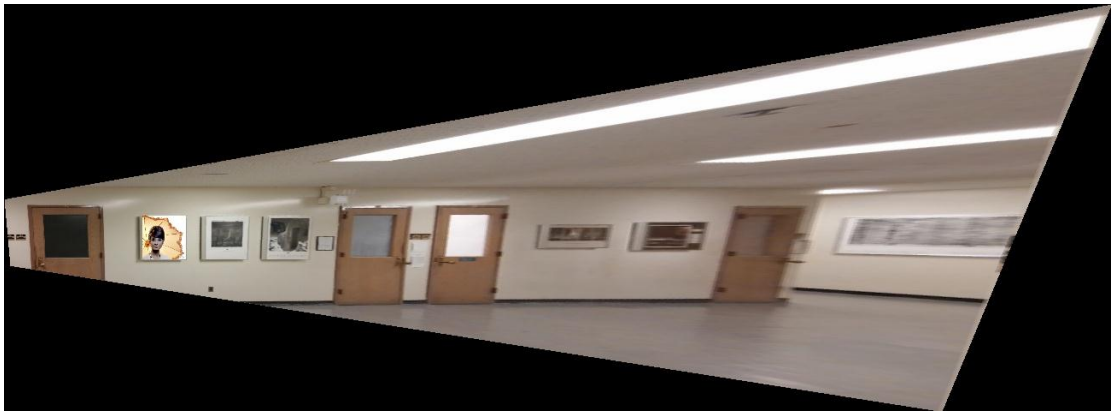
Step 2: Compute the position in world plane for the 4 corner of the source figure, then compute the new image size using the length on x-axis and y-axis. Then set ((-1)*most-left-x-position, (-1)*most-up-y-position) as the shift offset for all points, so that we can see all the result pixels on the new image.

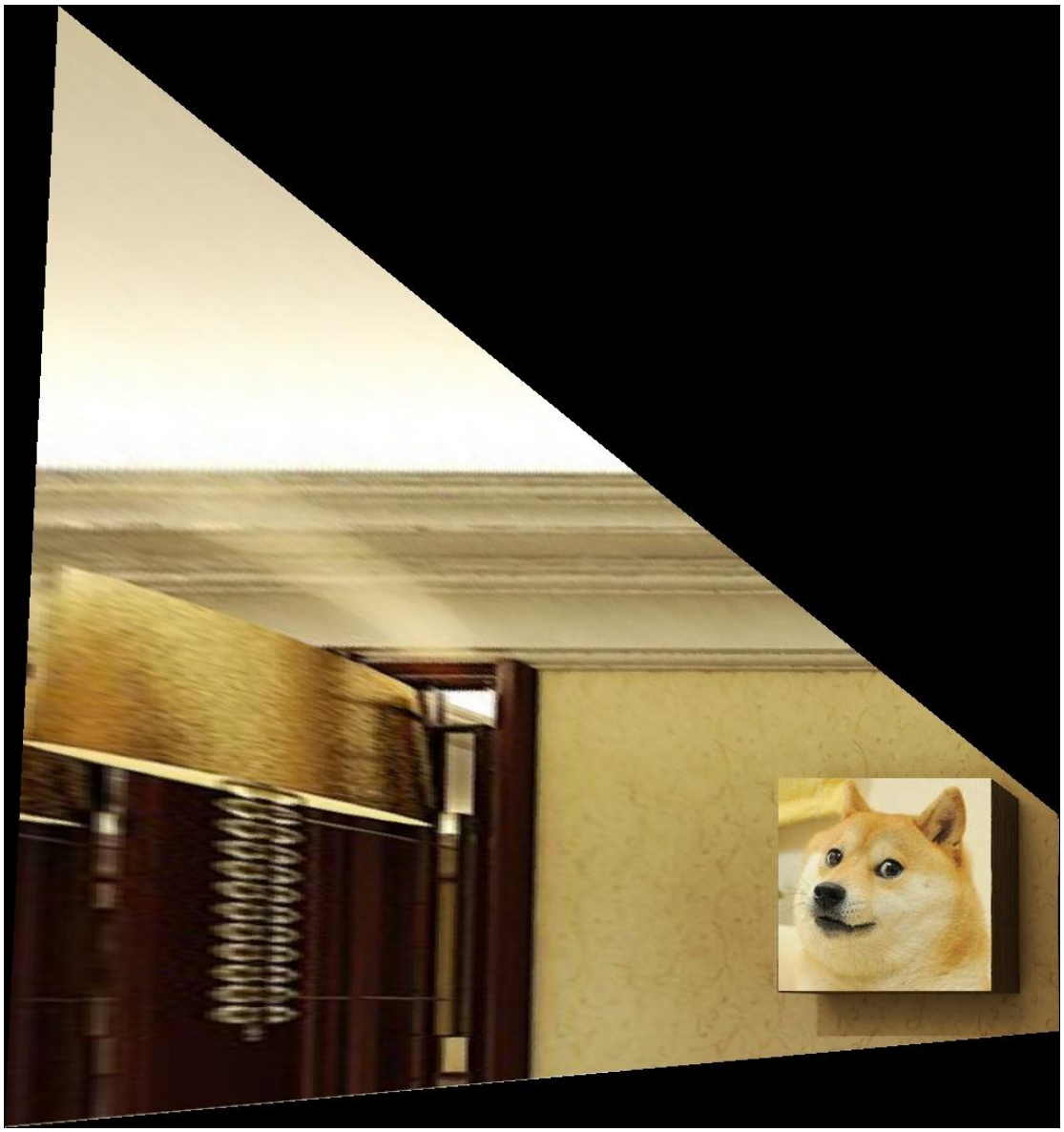
Step 3: For all the pixels in the new image, let them minus the offset, then using H to find the corresponding position in the source figure, then compute the color of that position using the above calculating method.

4. Result Image:

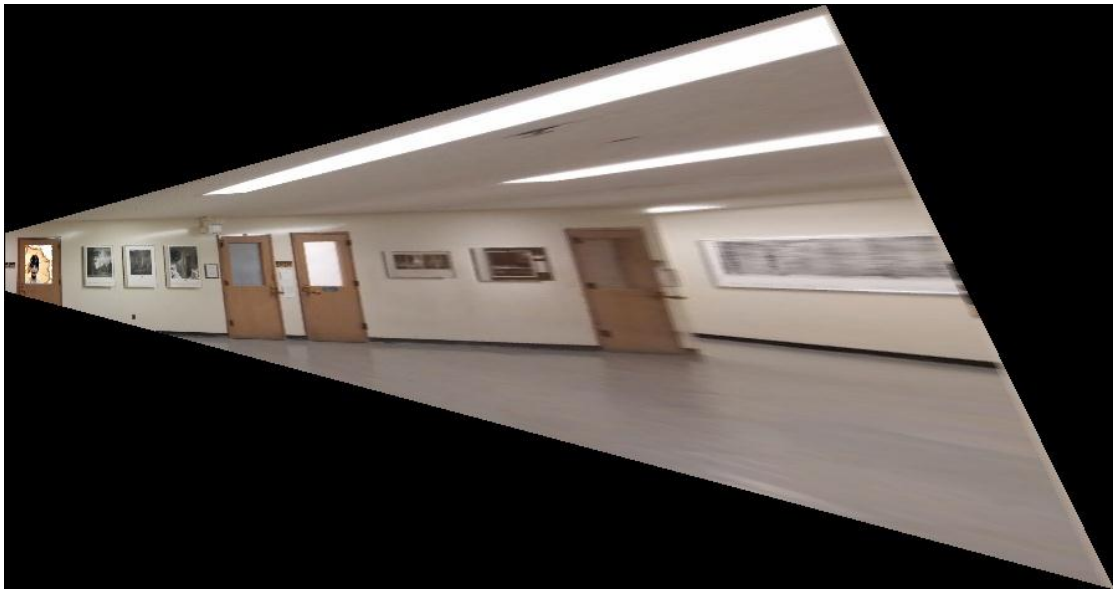
4.1 Task 1:

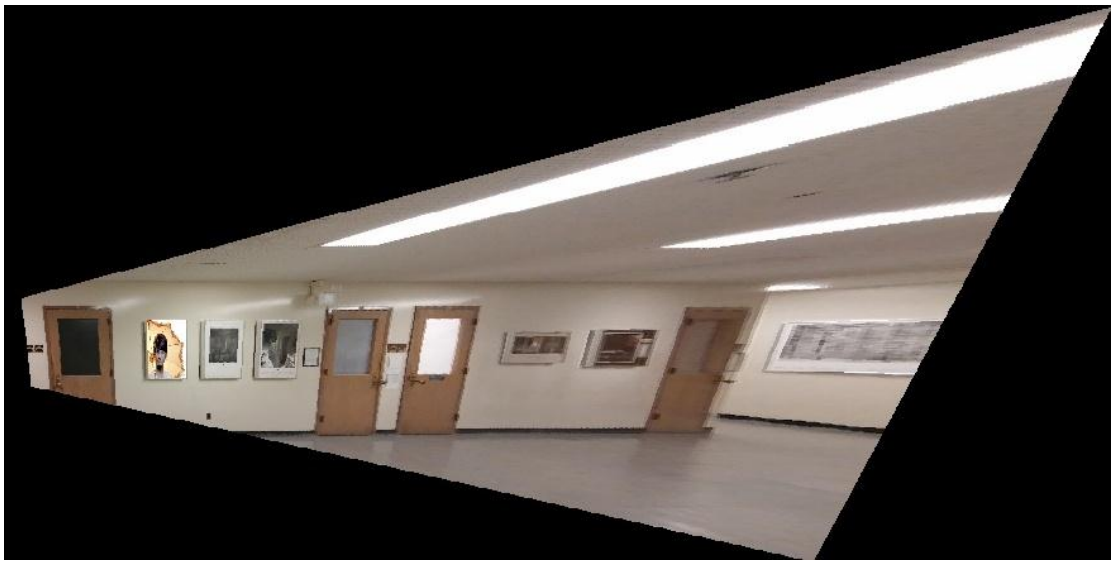
Task 2:

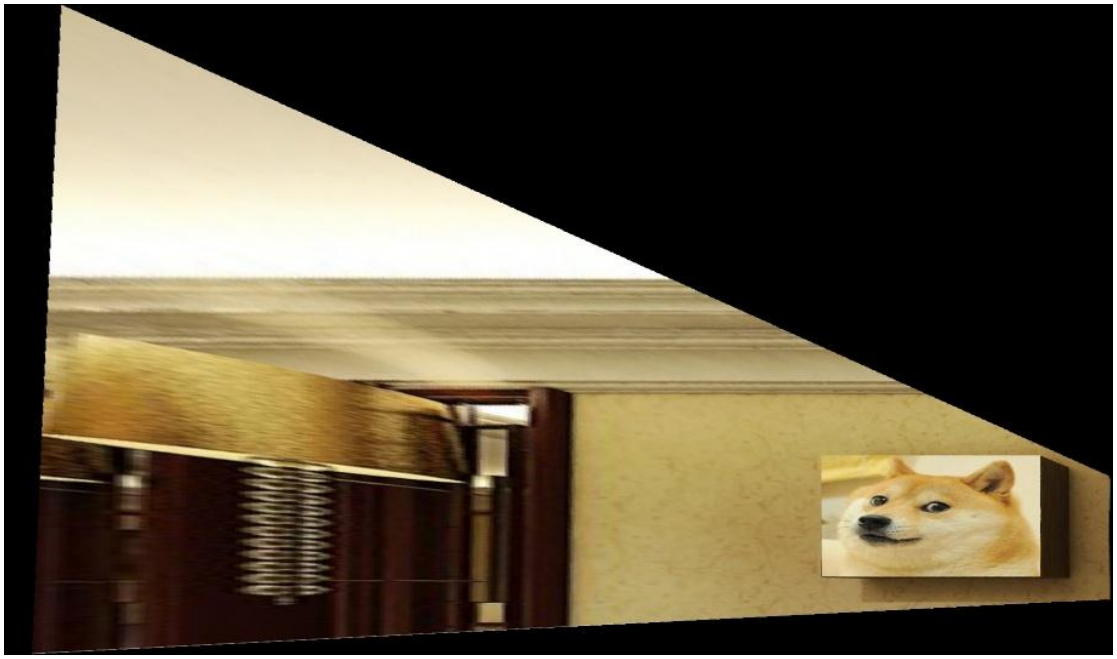




Task 3:







5. Code

5.1 Task 1:

```
import numpy as np
import cv2
import math

img1=cv2.imread('/home/wen/Dropbox/Pics/Frame.jpg')
img2=cv2.imread('/home/wen/Dropbox/Pics/Audrey.jpg')

##draw the selected region
```

```
img3= np.zeros((img1.shape[0],img1.shape[1],3),dtype='uint8')

pts = np.array([[186,152],[346,176],[342,433],[184,462]],np.int32)
pts = pts.reshape((-1,1,2))
cv2.fillPoly(img3,[pts],(255,255,255))

cv2.imshow('image',img3)
cv2.waitKey(0)

cv2.imwrite('selectRegion.jpg',img3)
cv2.destroyAllWindows()

##draw complete

## get the color of a pixel in photo from 'real painting'
def getcolor(point, img):
    p =
img[math.floor(point[0,0])%img.shape[0],math.floor(point[0,1])%img.shape[1]
]
    q =
img[math.floor(point[0,0])%img.shape[0],math.floor(point[0,1]+1)%img.shape[
1]]
    r =
img[math.floor(point[0,0]+1)%img.shape[0],math.floor(point[0,1])%img.shape[
1]]
    s =
img[math.floor(point[0,0]+1)%img.shape[0],math.floor(point[0,1]+1)%img.shap
e[1]]

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

    pweight= pow(pow(x,2)+pow(y,2),-0.5)
    qweight = pow(pow(x,2)+pow(1-y,2),-0.5)
    rweight = pow(pow(1-x,2)+pow(y,2),-0.5)
    sweight = pow(pow(1-x,2)+pow(1-y,2),-0.5)

    newpoint =
(p*pweight+q*qweight+r*rweight+s*sweight)/(pweight+qweight+rweight+sweight)
    return newpoint
## getcolor function end

##set up the P.Q.R.S and there corresponding points in img2
p = np.matrix('152 186 1',dtype=float)
```

```
q = np.matrix('176 346 1',dtype=float)
r = np.matrix('462 184 1',dtype=float)
s = np.matrix('433 342 1',dtype=float)

pr = np.matrix('0 0 1',dtype=float)
qr = np.matrix('0 0 1',dtype=float)
qr[0,1] = img2.shape[1]
rr = np.matrix('0 0 1',dtype=float)
rr[0,0] = img2.shape[0]
sr = np.matrix('0 0 1',dtype=float)
sr[0,0] = img2.shape[0]
sr[0,1] = img2.shape[1]

paraMatrix = np.zeros((8,8),dtype=float)
paraMatrix = np.matrix(paraMatrix,dtype=float)

##set up points end

#Transform: Photo => realFigure

##get ready for paraMatrix*parameter=Rvector

paraMatrix[0,0:3] = p
paraMatrix[0,6:9] = p[0,0:2]*(-1)*pr[0,0]
paraMatrix[1,3:6] = p
paraMatrix[1,6:9] = p[0,0:2]*(-1)*pr[0,1]

paraMatrix[2,0:3] = q
paraMatrix[2,6:9] = q[0,0:2]*(-1)*qr[0,0]
paraMatrix[3,3:6] = q
paraMatrix[3,6:9] = q[0,0:2]*(-1)*qr[0,1]

paraMatrix[4,0:3] = r
paraMatrix[4,6:9] = r[0,0:2]*(-1)*rr[0,0]
paraMatrix[5,3:6] = r
paraMatrix[5,6:9] = r[0,0:2]*(-1)*rr[0,1]

paraMatrix[6,0:3] = s
paraMatrix[6,6:9] = s[0,0:2]*(-1)*sr[0,0]
paraMatrix[7,3:6] = s
paraMatrix[7,6:9] = s[0,0:2]*(-1)*sr[0,1]

Rvector = np.matrix('0 0 0 0 0 0 0 0',dtype=float)
Rvector[0,0:2] = pr[0,0:2]
```

```
Rvector[0,2:4] = qr[0,0:2]
Rvector[0,4:6] = rr[0,0:2]
Rvector[0,6:8] = sr[0,0:2]

parameter = paraMatrix.I*Rvector.T
parameter = parameter.T

H = np.zeros((3,3),dtype=float)
H[0] = parameter[0,0:3]
H[1] = parameter[0,3:6]
H[2,0:2] = parameter[0,6:8]
H[2,2] = 1

print H
##get H

##get every pixel-on-the-door's new color
temp = np.matrix('0 0 1',dtype=float)

for row in range(0,img1.shape[0]):
    for column in range(0,img1.shape[1]):
        if img3[row,column,1] > 0:
            temp[0,0] = row
            temp[0,1] = column
            tr = H*temp.T
            tr = tr.T
            tr = tr/tr[0,2]
            img1[row,column]=getcolor(tr,img2)
            #img1[row,column]=img1[0,0]

cv2.imshow('image',img1)
cv2.waitKey(0)
cv2.imwrite('myNewImage.jpg',img1)
cv2.destroyAllWindows()

## get color end
```

5.2 Task 2:

```
import numpy as np
import cv2
import math
```

```
scale = 10

##help to transform points use H, and unify it
def Ptransform(H,point):
    newpoint = H*point.T
    newpoint = newpoint.T
    newpoint = newpoint/newpoint[0,2]
    return newpoint
##Ptransform end

## get the color of a pixel in photo from 'real painting'
def getcolor(point, img):
    p =
img[math.floor(point[0,0])%img.shape[0],math.floor(point[0,1])%img.shape[1]
]
    q =
img[math.floor(point[0,0])%img.shape[0],math.floor(point[0,1]+1)%img.shape[
1]]
    r =
img[math.floor(point[0,0]+1)%img.shape[0],math.floor(point[0,1])%img.shape[
1]]
    s =
img[math.floor(point[0,0]+1)%img.shape[0],math.floor(point[0,1]+1)%img.shap
e[1]]

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

    pweight= pow(pow(x,2)+pow(y,2),-0.5)
    qweight = pow(pow(x,2)+pow(1-y,2),-0.5)
    rweight = pow(pow(1-x,2)+pow(y,2),-0.5)
    sweight = pow(pow(1-x,2)+pow(1-y,2),-0.5)

    newpoint =
(p*pweight+q*qweight+r*rweight+s*sweight)/(pweight+qweight+rweight+sweight)
    return newpoint
## getcolor function end

img1=cv2.imread('/home/wen/Dropbox/Pics/Frame.jpg')
img2=cv2.imread('/home/wen/Dropbox/Pics/Audrey.jpg')

##draw the selected region

img3= np.zeros((img1.shape[0],img1.shape[1],3),dtype='uint8')
```

```
pts = np.array([[486,200],[566,210],[566,406],[484,421]],np.int32)
pts = pts.reshape((-1,1,2))
cv2.fillPoly(img3,[pts],(255,255,255))

cv2.imshow('image',img3)
cv2.waitKey(0)

cv2.imwrite('selectRegion.jpg',img3)
cv2.destroyAllWindows()

##draw complete

##set up the P.Q.R.S and there corresponding points in img2
p = np.matrix('200 486 1',dtype=float)
q = np.matrix('210 566 1',dtype=float)
r = np.matrix('421 484 1',dtype=float)
s = np.matrix('406 566 1',dtype=float)

pr = np.matrix('0 0 1',dtype=float)
qr = np.matrix('0 0 1',dtype=float)
qr[0,1] = img2.shape[1]
rr = np.matrix('0 0 1',dtype=float)
rr[0,0] = img2.shape[0]
sr = np.matrix('0 0 1',dtype=float)
sr[0,0] = img2.shape[0]
sr[0,1] = img2.shape[1]

paraMatrix = np.zeros((8,8),dtype=float)
paraMatrix = np.matrix(paraMatrix,dtype=float)

##set up points end

#Transform: Photo => realFigure

##get ready for paraMatrix*parameter=Rvector

paraMatrix[0,0:3] = p
paraMatrix[0,6:9] = p[0,0:2]*(-1)*pr[0,0]
paraMatrix[1,3:6] = p
paraMatrix[1,6:9] = p[0,0:2]*(-1)*pr[0,1]

paraMatrix[2,0:3] = q
paraMatrix[2,6:9] = q[0,0:2]*(-1)*qr[0,0]
```

```
paraMatrix[3,3:6] = q
paraMatrix[3,6:9] = q[0,0:2]*(-1)*qr[0,1]

paraMatrix[4,0:3] = r
paraMatrix[4,6:9] = r[0,0:2]*(-1)*rr[0,0]
paraMatrix[5,3:6] = r
paraMatrix[5,6:9] = r[0,0:2]*(-1)*rr[0,1]

paraMatrix[6,0:3] = s
paraMatrix[6,6:9] = s[0,0:2]*(-1)*sr[0,0]
paraMatrix[7,3:6] = s
paraMatrix[7,6:9] = s[0,0:2]*(-1)*sr[0,1]

Rvector = np.matrix('0 0 0 0 0 0 0 0',dtype=float)
Rvector[0,0:2] = pr[0,0:2]
Rvector[0,2:4] = qr[0,0:2]
Rvector[0,4:6] = rr[0,0:2]
Rvector[0,6:8] = sr[0,0:2]

parameter = paraMatrix.I*Rvector.T
parameter = parameter.T

H = np.zeros((3,3),dtype=float)
H = np.matrix(H,dtype=float)
H[0] = parameter[0,0:3]
H[1] = parameter[0,3:6]
H[2,0:2] = parameter[0,6:8]
H[2,2] = 1

print H
##get H

##recompute the Picture frame
ul = np.matrix('0 0 1',dtype=float)
ur = np.matrix('0 0 1',dtype=float)
dl = np.matrix('0 0 1',dtype=float)
dr = np.matrix('0 0 1',dtype=float)

ur[0,1] = img1.shape[1]
dr[0,1] = img1.shape[1]
dl[0,0] = img1.shape[0]
dr[0,0] = img1.shape[0]

upperleft = Ptransform(H,ul)
```

```
upperright = Ptransform(H,ur)
downleft = Ptransform(H,dl)
downright = Ptransform(H,dr)

newrow = max(downleft[0,0],downright[0,0]) -
min(upperleft[0,0],upperright[0,0])
newrow = math.floor(newrow)+1
newrow = newrow/scale
newcolumn = max(upperright[0,1],downright[0,1]) -
min(upperleft[0,1],downleft[0,1])
newcolumn = math.floor(newcolumn)+1
newcolumn = newcolumn/scale

##draw the background for new picture
img4 = np.zeros((newrow,newcolumn,3),dtype='uint8')

#shift vector into the picture
shiftV = np.matrix('0 0 0',dtype=float)
shiftV[0,0] = min(upperleft[0,0],upperright[0,0])*(-1)
shiftV[0,1] = min(upperleft[0,1],downleft[0,1])*(-1)

#shift img2 into the frame ABCD
shiftp = Ptransform(H,p)
shiftp = shiftp + shiftV
shiftp[0,2] = 0

##get every pixel-on-the-door's new color
temp = np.matrix('0 0 1',dtype=float)

##
print img4.shape
print 'begin loop1'

for row in range(0,img4.shape[0]):
    for column in range(0,img4.shape[1]):
        temp[0,0] = row*scale
        temp[0,1] = column*scale
        temp = temp - shiftV
        tpic = Ptransform(H.I,temp)
        if tpic[0,0]>=0 and tpic[0,0]<=img1.shape[0] and tpic[0,1]>=0 and
tpic[0,1]<=img1.shape[1]:
            if img3[tpic[0,0],tpic[0,0],0] <255:
                img4[row,column] = getcolor(tpic,img1)
```

```
print 'begin loop2'

for row in range(0,img2.shape[0]):
    for column in range(0,img2.shape[1]):

    img4[(row+int(shiftp[0,0]))/scale,(column+int(shiftp[0,1]))/scale]=img2
[row,column]
        #img1[row,column]=img1[0,0]

#cv2.imshow('image',img4)
#cv2.waitKey(0)
cv2.imwrite('myNewImage.jpg',img4)
#cv2.destroyAllWindows()

## get color end
```

5.3 Task 3 Part 1:

```
import numpy as np
import cv2
import math

scale = 3

##help to transform points use H, and unify it
def Ptransform(H,point):
    newpoint = H*point.T
    newpoint = newpoint.T
    newpoint = newpoint/newpoint[0,2]
    return newpoint
##Ptransform end

## get the color of a pixel in photo from 'real painting'
def getcolor(point, img):
    p =
img[math.floor(point[0,0])%img.shape[0],math.floor(point[0,1])%img.shape[1]
]
    q =
img[math.floor(point[0,0])%img.shape[0],math.floor(point[0,1]+1)%img.shape[
1]]
    r =
img[math.floor(point[0,0]+1)%img.shape[0],math.floor(point[0,1])%img.shape[
1]]
    s =
```

```
img[math.floor(point[0,0]+1)%img.shape[0],math.floor(point[0,1]+1)%img.shape[1]]

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

    pweight= pow(pow(x,2)+pow(y,2),-0.5)
    qweight = pow(pow(x,2)+pow(1-y,2),-0.5)
    rweight = pow(pow(1-x,2)+pow(y,2),-0.5)
    sweight = pow(pow(1-x,2)+pow(1-y,2),-0.5)

    newpoint =
(p*pweight+q*qweight+r*rweight+s*sweight)/(pweight+qweight+rweight+sweight)
    return newpoint
## getcolor function end

img1=cv2.imread('/home/wen/pythonlab/hw2Task1.jpg')

##set up the P.Q.R.S and there corresponding points in img2
p = np.matrix('152 186 1',dtype=float)
q = np.matrix('176 346 1',dtype=float)
r = np.matrix('462 184 1',dtype=float)
s = np.matrix('433 342 1',dtype=float)

pr = np.matrix('0 0 1',dtype=float)
qr = np.matrix('0 63 1',dtype=float)
rr = np.matrix('92 0 1',dtype=float)
sr = np.matrix('92 63 1',dtype=float)

paraMatrix = np.zeros((8,8),dtype=float)
paraMatrix = np.matrix(paraMatrix,dtype=float)

##set up points end

#Transform: Photo => realFigure

##get ready for paraMatrix*parameter=Rvector

paraMatrix[0,0:3] = p
paraMatrix[0,6:9] = p[0,0:2]*(-1)*pr[0,0]
paraMatrix[1,3:6] = p
paraMatrix[1,6:9] = p[0,0:2]*(-1)*pr[0,1]

paraMatrix[2,0:3] = q
```

```
paraMatrix[2,6:9] = q[0,0:2]*(-1)*qr[0,0]
paraMatrix[3,3:6] = q
paraMatrix[3,6:9] = q[0,0:2]*(-1)*qr[0,1]

paraMatrix[4,0:3] = r
paraMatrix[4,6:9] = r[0,0:2]*(-1)*rr[0,0]
paraMatrix[5,3:6] = r
paraMatrix[5,6:9] = r[0,0:2]*(-1)*rr[0,1]

paraMatrix[6,0:3] = s
paraMatrix[6,6:9] = s[0,0:2]*(-1)*sr[0,0]
paraMatrix[7,3:6] = s
paraMatrix[7,6:9] = s[0,0:2]*(-1)*sr[0,1]

Rvector = np.matrix('0 0 0 0 0 0 0 0',dtype=float)
Rvector[0,0:2] = pr[0,0:2]
Rvector[0,2:4] = qr[0,0:2]
Rvector[0,4:6] = rr[0,0:2]
Rvector[0,6:8] = sr[0,0:2]

parameter = paraMatrix.I*Rvector.T
parameter = parameter.T

H = np.zeros((3,3),dtype=float)
H = np.matrix(H,dtype=float)
H[0] = parameter[0,0:3]
H[1] = parameter[0,3:6]
H[2,0:2] = parameter[0,6:8]
H[2,2] = 1

print H
##get H

##recompute the Picture frame
ul = np.matrix('0 0 1',dtype=float)
ur = np.matrix('0 0 1',dtype=float)
dl = np.matrix('0 0 1',dtype=float)
dr = np.matrix('0 0 1',dtype=float)

ur[0,1] = img1.shape[1]
dr[0,1] = img1.shape[1]
dl[0,0] = img1.shape[0]
dr[0,0] = img1.shape[0]
```

```
upperleft = Ptransform(H,ul)
upperright = Ptransform(H,ur)
downleft = Ptransform(H,dl)
downright = Ptransform(H,dr)

newrow = max(downleft[0,0],downright[0,0]) -
min(upperleft[0,0],upperright[0,0])
newrow = math.floor(newrow)+1
newrow = newrow/scale
newcolumn = max(upperright[0,1],downright[0,1]) -
min(upperleft[0,1],downleft[0,1])
newcolumn = math.floor(newcolumn)+1
newcolumn = newcolumn/scale

##draw the background for new picture
img4 = np.zeros((newrow,newcolumn,3),dtype='uint8')

#shift vector into the picture
shiftV = np.matrix('0 0 0',dtype=float)
shiftV[0,0] = min(upperleft[0,0],upperright[0,0])*(-1)
shiftV[0,1] = min(upperleft[0,1],downleft[0,1])*(-1)

#shift img2 into the frame ABCD
shiftp = Ptransform(H,p)
shiftp = shiftp + shiftV
shiftp[0,2] = 0

##get every pixel-on-the-door's new color
temp = np.matrix('0 0 1',dtype=float)

##
print img4.shape
print 'begin loop1'

for row in range(0,img4.shape[0]):
    for column in range(0,img4.shape[1]):
        temp[0,0] = row*scale
        temp[0,1] = column*scale
        temp = temp - shiftV
        tpic = Ptransform(H.I,temp)
        if tpic[0,0]>=0 and tpic[0,0]<=img1.shape[0] and tpic[0,1]>=0 and
tpic[0,1]<=img1.shape[1]:
            img4[row,column] = getcolor(tpic,img1)
```

```
#cv2.imshow('image',img4)
#cv2.waitKey(0)
cv2.imwrite('myNewImage.jpg',img4)
#cv2.destroyAllWindows()

## get color end
```

5.4 Task 3 part 2:

```
import numpy as np
import cv2
import math

scale = 2

##help to transform points use H, and unify it
def Ptransform(H,point):
    newpoint = H*point.T
    newpoint = newpoint.T
    newpoint = newpoint/newpoint[0,2]
    return newpoint
##Ptransform end

## get the color of a pixel in photo from 'real painting'
def getcolor(point, img):
    p =
img[math.floor(point[0,0])%img.shape[0],math.floor(point[0,1])%img.shape[1]
]
    q =
img[math.floor(point[0,0])%img.shape[0],math.floor(point[0,1]+1)%img.shape[
1]]
    r =
img[math.floor(point[0,0]+1)%img.shape[0],math.floor(point[0,1])%img.shape[
1]]
    s =
img[math.floor(point[0,0]+1)%img.shape[0],math.floor(point[0,1]+1)%img.shap
e[1]]

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

    pweight= pow(pow(x,2)+pow(y,2),-0.5)
    qweight = pow(pow(x,2)+pow(1-y,2),-0.5)
    rweight = pow(pow(1-x,2)+pow(y,2),-0.5)
```

```
    sweight = pow(pow(1-x,2)+pow(1-y,2),-0.5)

    newpoint =
(p*pweight+q*qweight+r*rweight+s*sweight)/(pweight+qweight+rweight+sweight)
    return newpoint
## getcolor function end
img1=cv2.imread('/home/wen/pythonlab/hw2Task2_withoutWeighted.jpg')

##set up the P.Q.R.S and there corresponding points in img2
p = np.matrix('247 162 1',dtype=float)
q = np.matrix('247 212 1',dtype=float)
r = np.matrix('298 161 1',dtype=float)
s = np.matrix('298 212 1',dtype=float)

pr = np.matrix('0 0 1',dtype=float)
qr = np.matrix('0 61 1',dtype=float)
rr = np.matrix('91 0 1',dtype=float)
sr = np.matrix('91 61 1',dtype=float)

paraMatrix = np.zeros((8,8),dtype=float)
paraMatrix = np.matrix(paraMatrix,dtype=float)

##set up points end

#Transform: Photo => realFigure

##get ready for paraMatrix*parameter=Rvector

paraMatrix[0,0:3] = p
paraMatrix[0,6:9] = p[0,0:2]*(-1)*pr[0,0]
paraMatrix[1,3:6] = p
paraMatrix[1,6:9] = p[0,0:2]*(-1)*pr[0,1]

paraMatrix[2,0:3] = q
paraMatrix[2,6:9] = q[0,0:2]*(-1)*qr[0,0]
paraMatrix[3,3:6] = q
paraMatrix[3,6:9] = q[0,0:2]*(-1)*qr[0,1]

paraMatrix[4,0:3] = r
paraMatrix[4,6:9] = r[0,0:2]*(-1)*rr[0,0]
paraMatrix[5,3:6] = r
paraMatrix[5,6:9] = r[0,0:2]*(-1)*rr[0,1]

paraMatrix[6,0:3] = s
```

```
paraMatrix[6,6:9] = s[0,0:2]*(-1)*sr[0,0]
paraMatrix[7,3:6] = s
paraMatrix[7,6:9] = s[0,0:2]*(-1)*sr[0,1]

Rvector = np.matrix('0 0 0 0 0 0 0 0',dtype=float)
Rvector[0,0:2] = pr[0,0:2]
Rvector[0,2:4] = qr[0,0:2]
Rvector[0,4:6] = rr[0,0:2]
Rvector[0,6:8] = sr[0,0:2]

parameter = paraMatrix.I*Rvector.T
parameter = parameter.T

H = np.zeros((3,3),dtype=float)
H = np.matrix(H,dtype=float)
H[0] = parameter[0,0:3]
H[1] = parameter[0,3:6]
H[2,0:2] = parameter[0,6:8]
H[2,2] = 1

print H
##get H

##recompute the Picture frame
ul = np.matrix('0 0 1',dtype=float)
ur = np.matrix('0 0 1',dtype=float)
dl = np.matrix('0 0 1',dtype=float)
dr = np.matrix('0 0 1',dtype=float)

ur[0,1] = img1.shape[1]
dr[0,1] = img1.shape[1]
dl[0,0] = img1.shape[0]
dr[0,0] = img1.shape[0]

upperleft = Ptransform(H,ul)
upperright = Ptransform(H,ur)
downleft = Ptransform(H,dl)
downright = Ptransform(H,dr)

newrow = max(downleft[0,0],downright[0,0]) -
min(upperleft[0,0],upperright[0,0])
newrow = math.floor(newrow)+1
newrow = newrow/scale
newcolumn = max(upperright[0,1],downright[0,1]) -
```

```
min(upperleft[0,1],downleft[0,1])
newcolumn = math.floor(newcolumn)+1
newcolumn = newcolumn/scale

##draw the background for new picture
img4 = np.zeros((newrow,newcolumn,3),dtype='uint8')

#shift vector into the picture
shiftV = np.matrix('0 0 0',dtype=float)
shiftV[0,0] = min(upperleft[0,0],upperright[0,0])*(-1)
shiftV[0,1] = min(upperleft[0,1],downleft[0,1])*(-1)



#shift img2 into the frame ABCD
shiftp = Ptransform(H,p)
shiftp = shiftp + shiftV
shiftp[0,2] = 0

##get every pixel-on-the-door's new color
temp = np.matrix('0 0 1',dtype=float)

##
print img4.shape
print 'begin loop1'

for row in range(0,img4.shape[0]):
    for column in range(0,img4.shape[1]):
        temp[0,0] = row*scale
        temp[0,1] = column*scale
        temp = temp - shiftV
        tpic = Ptransform(H.I,temp)
        if tpic[0,0]>=0 and tpic[0,0]<=img1.shape[0] and tpic[0,1]>=0 and
tpic[0,1]<=img1.shape[1]:
            img4[row,column] = getcolor(tpic,img1)

#cv2.imshow('image',img4)
#cv2.waitKey(0)
cv2.imwrite('myNewImage.jpg',img4)
#cv2.destroyAllWindows()

## get color end
```

Full name: Wen Yi
Purdue mail: yi35@purdue.edu