ECE 661 HW03
Rehana Mahfuz
rmahfuz@purdue.edu

Theory

Task 1

Point-to-point correspondences were used to de-distort the image. A homography was found to convert the distorted version of what we know to be a rectangle to the shape of the actual undistorted rectangle.

We know that x' = Hx describes the point correspondences in a homography. In this homework, we want to find the homography between two images. We make use of point correspondences between images to determine the homography.

$$x' = Hx \Rightarrow \begin{bmatrix} x1' \\ x2' \\ x3' \end{bmatrix} = \begin{bmatrix} h11 & h12 & h13 \\ h21 & h22 & h23 \\ h31 & h32 & h33 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} h11x1 + h12x2 + h13x3 \\ h21x1 + h22x2 + h23x3 \\ h31x1 + h32x2 + h33x3 \end{bmatrix} \quad (1)$$

Since we are dealing with homogeneous coordinates, we are only interested in finding ratios, which is why we only want to find the first eight coordinates of the matrix H, setting h_{33} to 1. We will also set x3 as 1, so we can use the physical coordinates $x = \frac{x1}{x3}$ and $y = \frac{x2}{x3}$

The physical coordinates x' and y' can be calculated as:

$$x' = \frac{x1'}{x3} = \frac{h11x1+h12x2+h13x3}{h31x1+h32x2+h33x3} = \frac{h11x+h12y+h13}{h31x+h32y+1} \quad (2)$$

$$y' = \frac{x2'}{x3} = \frac{h21x1+h22x2+h23x3}{h31x1+h32x2+h33x3} = \frac{h21x+h22y+h23}{h31x+h32y+1} \quad (3)$$

Thus, for every pair of correspondence points between the source image and the target image, we get two equations. To get eight equations, we would need four correspondence points. Since we have eight unknowns in the matrix H, these eight equations will be enough to find the unknowns.

$$\begin{bmatrix} x1' \\ y1' \\ x2' \\ y2' \\ x3' \\ y3' \\ x4' \\ y4' \end{bmatrix} = \begin{bmatrix} x1 & y1 & 1 & 0 & 0 & 0 & -x1x1' & -y1x1' \\ 0 & 0 & 0 & x1 & y1 & 1 & -x1y1' & -y1y1' \\ x2 & y2 & 1 & 0 & 0 & 0 & -x2x2' & -y2x2' \\ 0 & 0 & 0 & x2 & y2 & 1 & -x2y2' & -y2y2' \\ x3 & y3 & 1 & 0 & 0 & 0 & -x3x3' & -y3x3' \\ 0 & 0 & 0 & x3 & y3 & 1 & -x3y3' & -y3y3' \\ x4 & y4 & 1 & 0 & 0 & 0 & -x4x4' & -y4x4' \\ 0 & 0 & 0 & x4 & y4 & 1 & -x4y4' & -y4y4' \end{bmatrix} \begin{bmatrix} h11 \\ h12 \\ h13 \\ h21 \\ h22 \\ h23 \\ h31 \\ h32 \end{bmatrix} \quad (4)$$

(4) is obtained by repeating (2) and (3) for four points. (4) can also be expressed as t = Ph. Premultiplying by P^{-1}, we get $h = P^{-1}t$. That is how we find the column vector h, which is then reshaped to obtain the 3x3 matrix H.

Task 2

The 2-step method was used to remove first the projective distortion, and then the affine distortion.

To remove projective distortion, the vanishing line method was used. Two pairs of lines were found such that they are parallel in the world plane. Let these two pairs of lines be l and m, and L and M. These lines are found by taking the cross product of points that fall on the line. The coordinates of these points are found using GIMP. The cross product of lines l and m is used to find the vanishing point p, which is the point of intersection of these two distorted parallel lines. Similarly, the cross product of lines L and M gives us P, another vanishing point. Taking the cross product of vanishing points p and P gives us the vanishing line x. The homography to remove distortion is simply given by H = $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x1 & x2 & x3 \end{bmatrix}$.

To remove the affine distortion, the cos theta method is used. Two pairs of lines that are perpendicular to each other are found. Let us call these two pairs l, m and L,M.

$$\text{Cos(theta)} = \frac{\text{l1m1+l2m2}}{\text{sqrt((l1*l1+l2*l2)*(m1*m1+m2*m2))}} = \frac{transpose(l)C*_{infinity}*m}{\text{sqrt((transpose(l)*C*infinity*m)*(transpose(m)*C*infinity*l))}}$$

If A is the affine part of the homography, let S = AA^T. Then we can write:

[l1'm1'   l1'm2'+l2'm1'] $\begin{bmatrix} s11 \\ s12 \end{bmatrix}$ = [-l2'm2']

Repeating this with lines L and M gives us another set of equations, which, when solved for s11 and s12 give us the matrix S. Because S is symmetric, s12=s21. Also, we set s22=1 since we only want ratios between elements. This is why two pairs of perpendicular lines are sufficient.

Once we calculate S, we find its eigendecomposition. S = VD^{2}V^T, and A = VDV. Once A is found, all the remaining elements in the homography are 0 except the lower rightmost.

<u>Task 3</u>

The single-step method is used to remove both projective and affine distortions. We calculate the image of the degenerate conic at infinity. C*$_{\text{infinity}}$ = $\begin{bmatrix} a & \frac{b}{2} & \frac{d}{2} \\ \frac{b}{2} & c & \frac{e}{2} \\ \frac{d}{2} & \frac{e}{2} & f \end{bmatrix}$.

If l and m are <u>two</u> perpendicular lines in the world plane, we can write:

[l1'm1'   0.5*(l1'm2'+l2'm1')   l2'm2'   0.5*(l1'm3'+l3'm1')   0.5*(l2'm3'+l3'm2')]* $\begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix}$.= [-l3'm3'

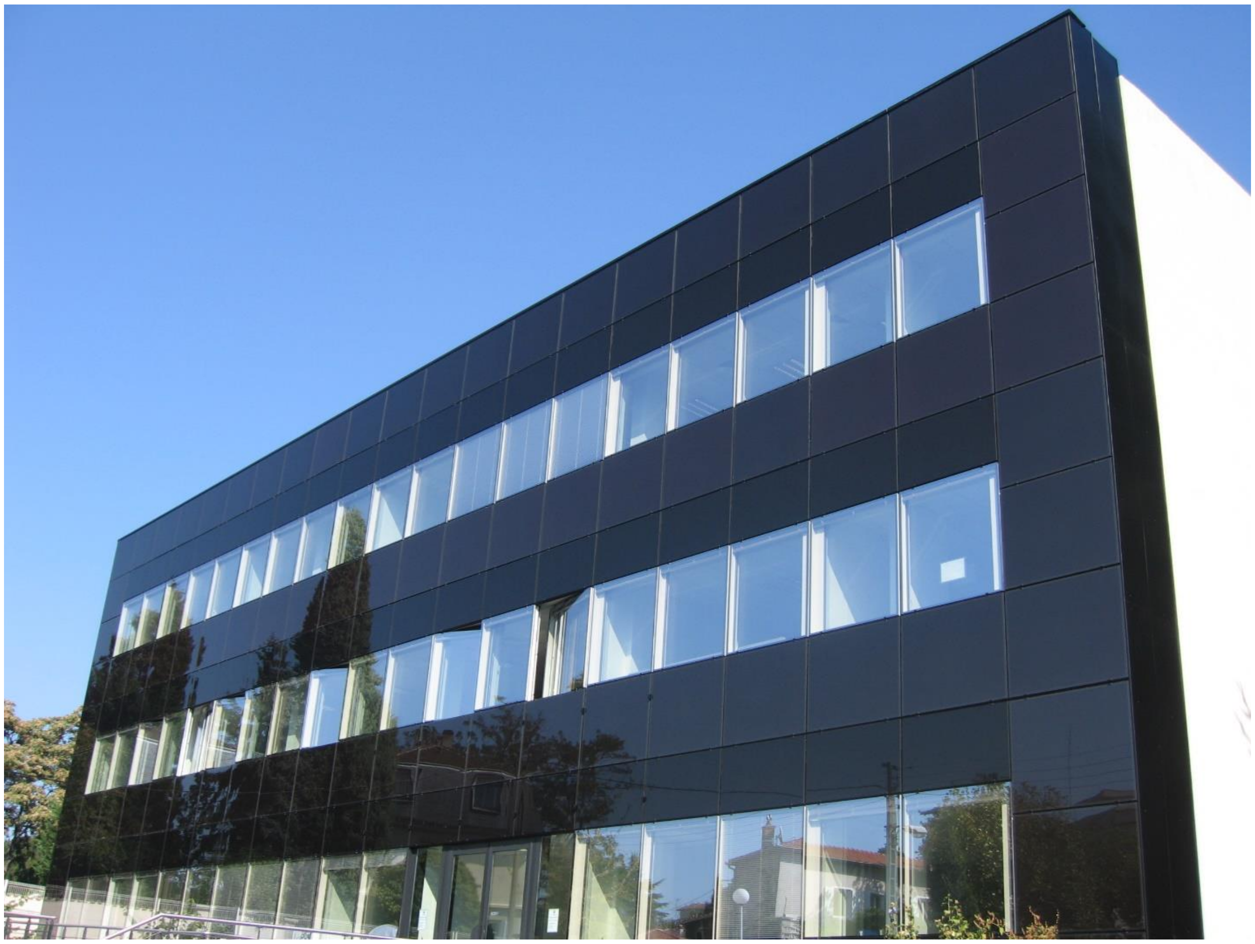
If we do this five pairs of perpendicular lines we will have enough equations to solve for the conic, at which point the eigendecomposition of the upper right 2x2 matrix lets us find A. The bottom left 1x2 matrix of the conic is v^{T}A. Once we find A and v, we have the homography which can be used to remove both distortions in a single step.

<u>Calculation of corner points</u>

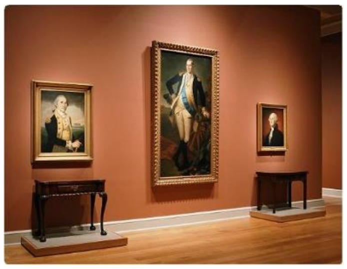
Once we know the homography to convert to world plane from image plane, the result, though correctly shaped, may not necessarily be placed in the desired position (part of the image may be outside of the boundary), and it also may not be scaled correctly. So the inverse of the calculated homography is used to find the corner points in the world plane, after which a new homography is calculated that positons the corner points in the desired place.

<u>Results:</u>

ECE 661 HW03
Rehana Mahfuz
rmahfuz@purdue.edu
Original image1

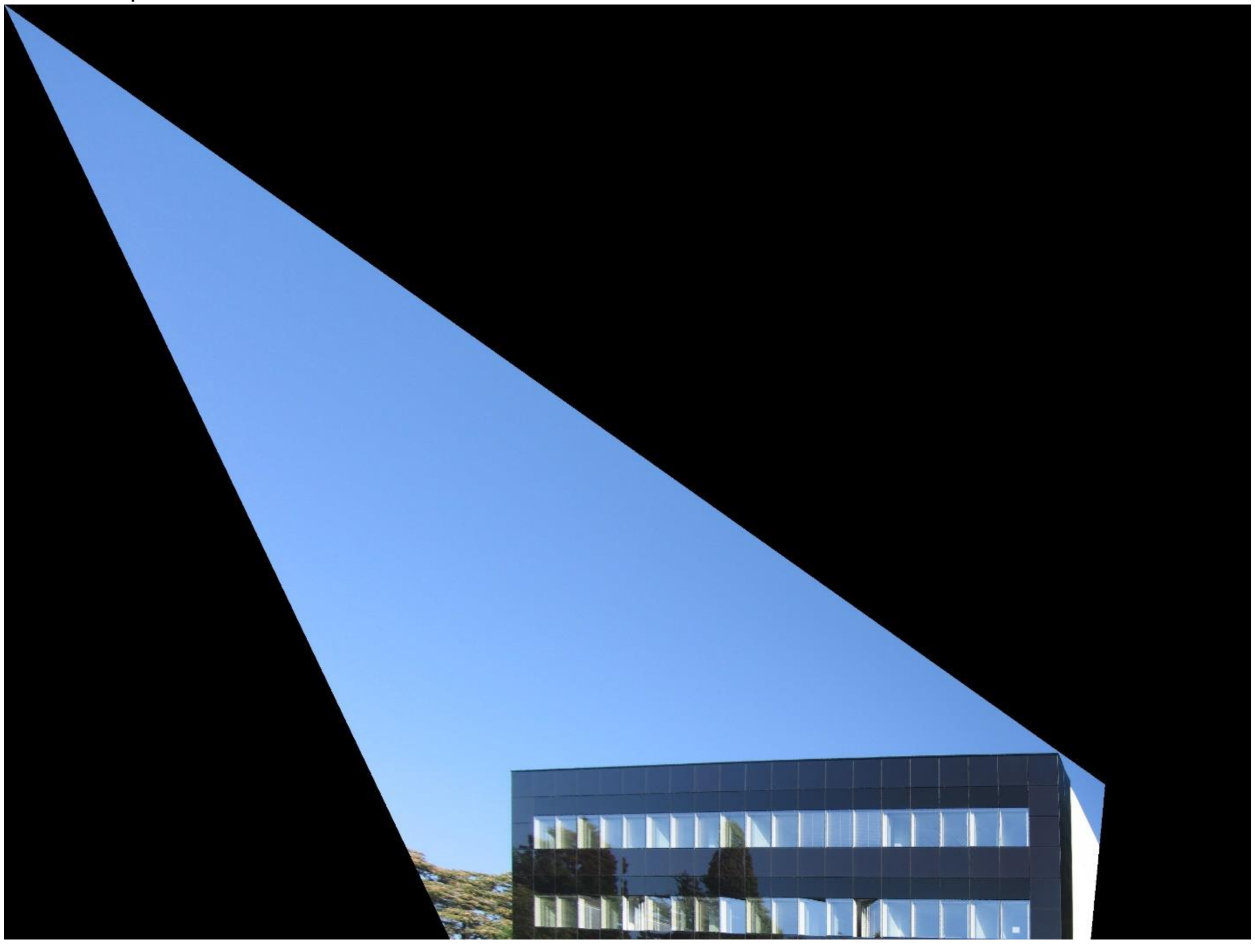


Original image2



Task1 image1 result

ECE 661 HW03
Rehana Mahfuz
rmahfuz@purdue.edu

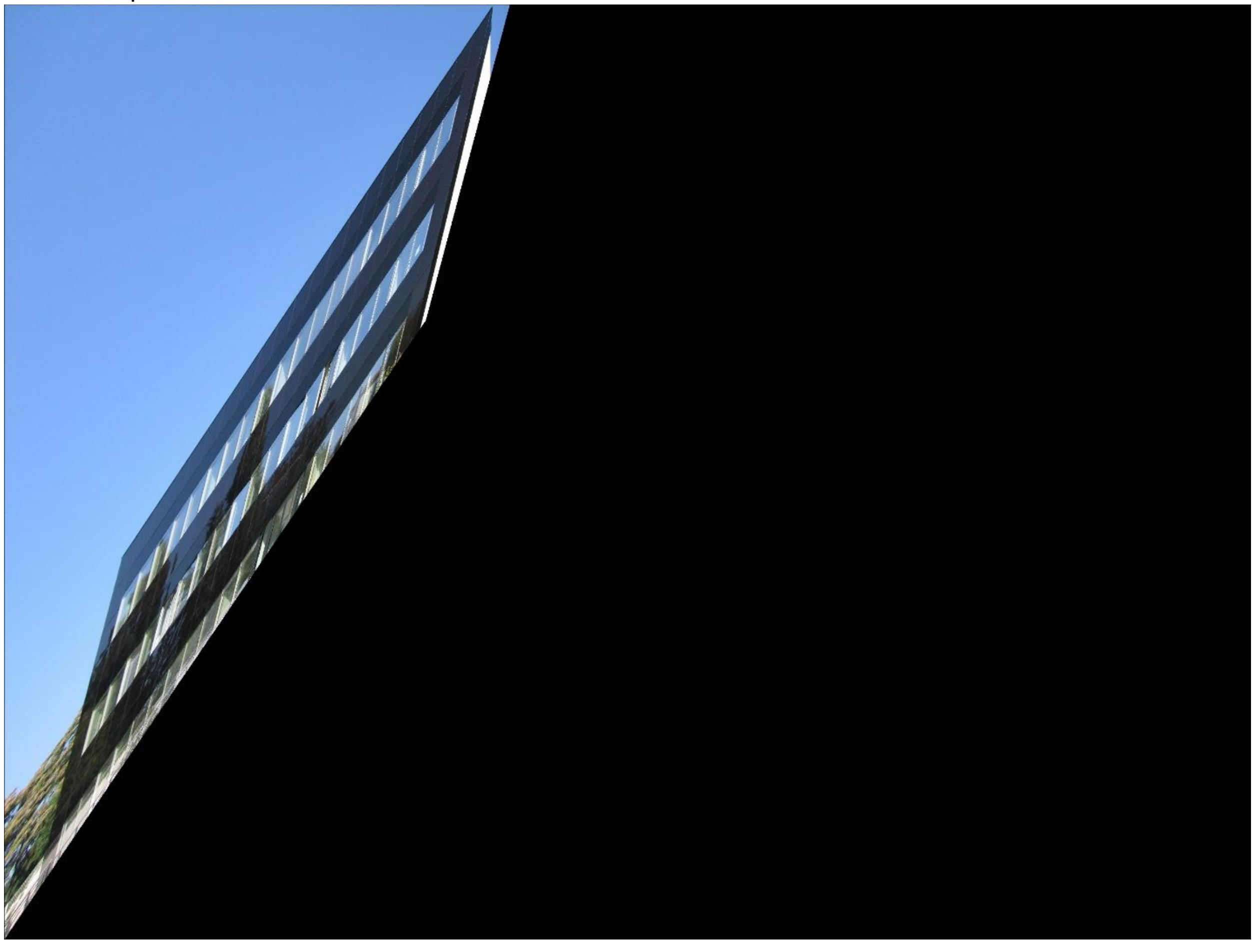


Task1 image2 result

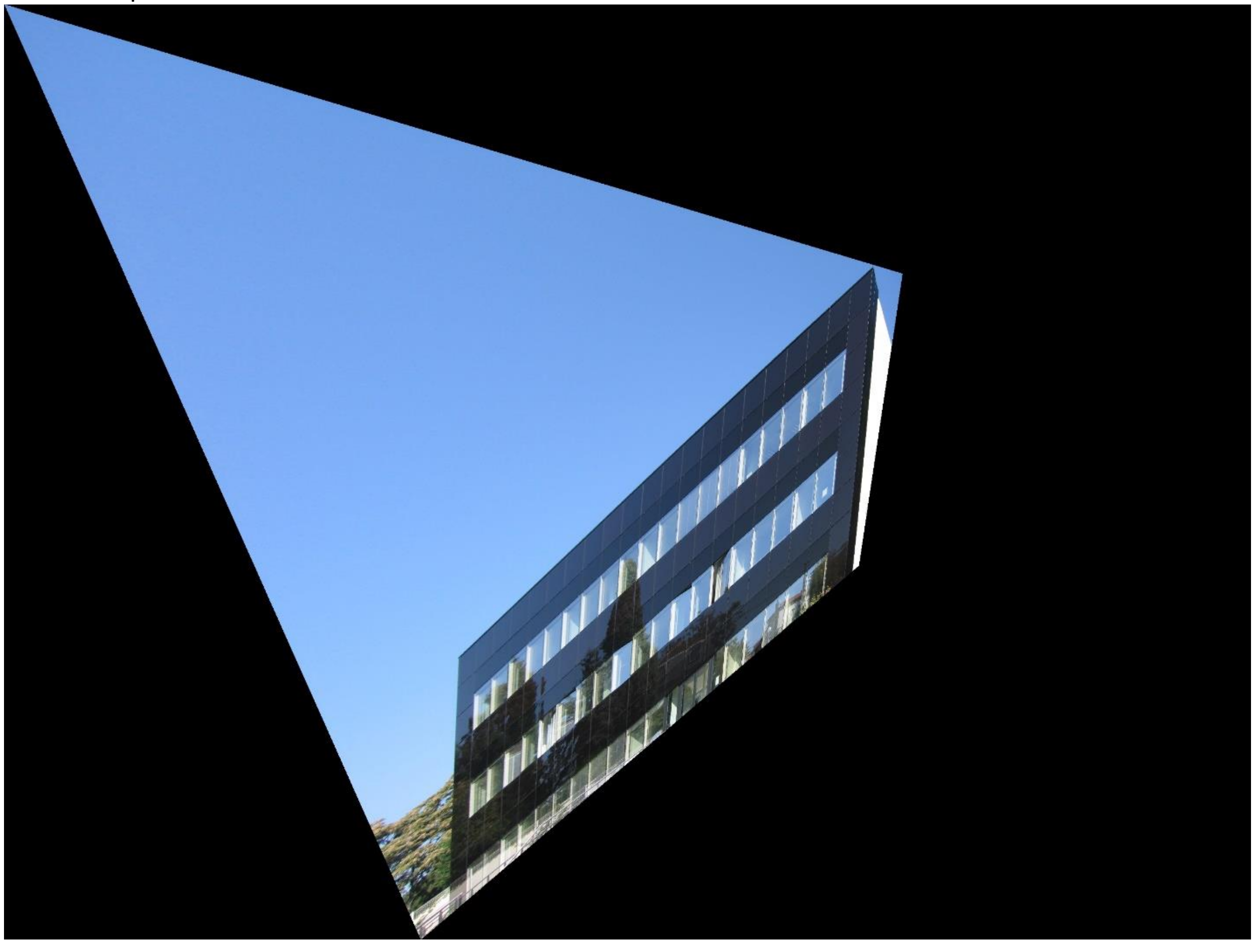
ECE 661 HW03
Rehana Mahfuz
rmahfuz@purdue.edu

Task2 image1 without projective distortion

ECE 661 HW03
Rehana Mahfuz
rmahfuz@purdue.edu

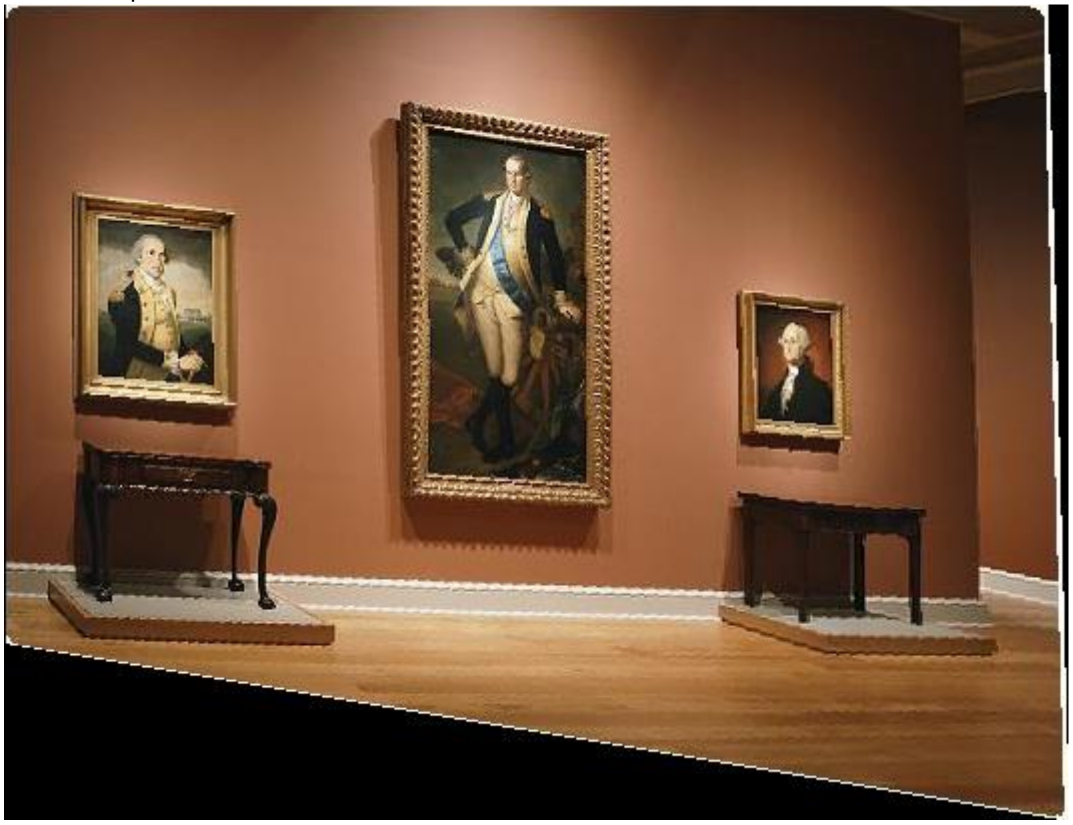

Task2 image1 without affine distortion

ECE 661 HW03
Rehana Mahfuz
rmahfuz@purdue.edu



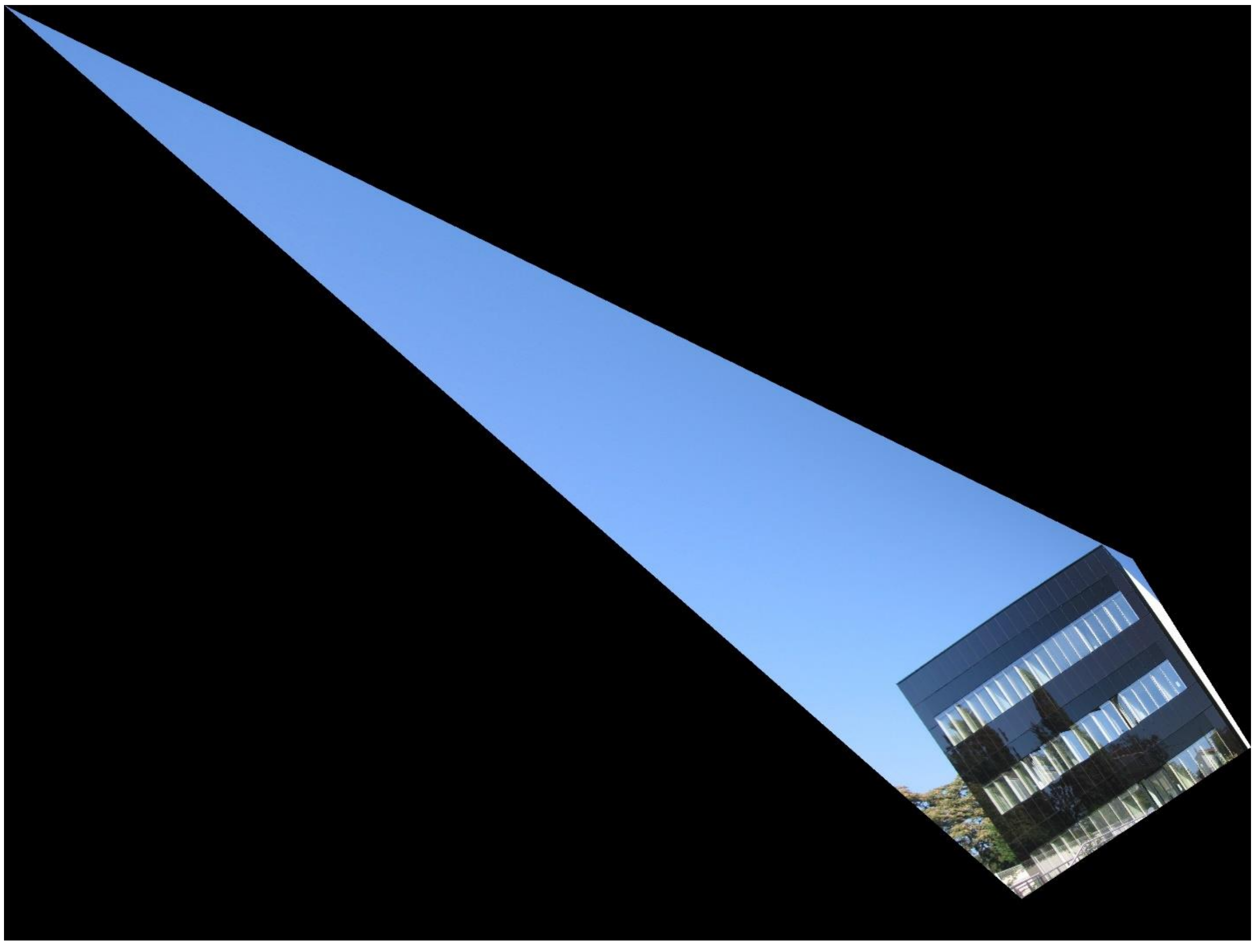
Task2 imag2 without projective distortion

ECE 661 HW03
Rehana Mahfuz
rmahfuz@purdue.edu



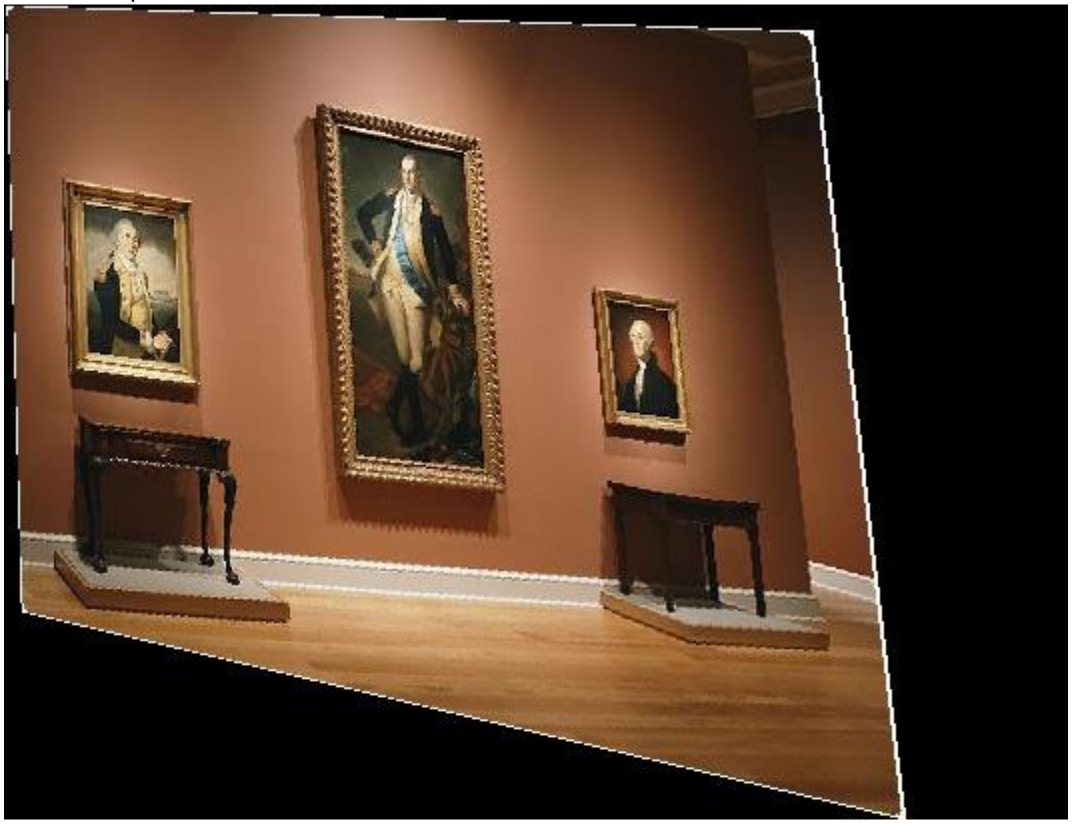
Task2 image2 without affine distortion

ECE 661 HW03
Rehana Mahfuz
rmahfuz@purdue.edu
Task3 image1 result



Task3 image2 result

ECE 661 HW03
Rehana Mahfuz
rmahfuz@purdue.edu



Note: Task 2 removal of affine distortion does not work correctly, so those resulting images are incorrect.

Code:

```
import numpy as np

import cv2, math

#from scipy.linalg import svd


#path = '../Users/rmahfuz/Desktop/661/HW03/'

path = ''

#=============================================================================================================

def find_homography(x, x_dash):

    '''x and x_dash are lists of four lists, each list containing two coordinates: [x,y]

    returns H = inv(P)*t'''

    t = np.array(x_dash, dtype = float).flatten().reshape((8,1))
    P = []

    for i in range(4):

        P.append([x[i][0], x[i][1], 1, 0, 0, 0, -1*x[i][0]*x_dash[i][0], -1*x[i][1]*x_dash[i][0]])

        P.append([0, 0, 0, x[i][0], x[i][1], 1, -1*x[i][0]*x_dash[i][1], -1*x[i][1]*x_dash[i][1]])

    P = np.array(P, dtype = float)#; print(P)

    P_inv = np.linalg.inv(P)

    H = np.matmul(P_inv, t) # H = inv(P)*t

    return H

#-----------------------------------------------------------------------------------------------------

def apply_homography(im1, im2, pts1, pts2):

    '''img1 is destination img, img2 is source image. Returns im1 with distorted im2 in it'''

    H = find_homography(pts1, pts2)

    H = np.insert(H, 8, 1).reshape((3,3))

    return find_result(im1, im2, pts1, H)

#-----------------------------------------------------------------------------------------------------

def find_result(im1, im2, pts1, H, scale = 1, tx = 0, ty = 0):

    '''Returns the resulting image'''

    #creating the 'dummy' image which is completely blacked out except at the pixels which need to be
replaced

    dummy = np.zeros((im1.shape[0],im1.shape[1],3),dtype='uint8') #completely blacked out

    pts = np.array([[pts1[0][1], pts1[0][0]], [pts1[1][1], pts1[1][0]], [pts1[2][1], pts1[2][0]], [pts1[3][1],
pts1[3][0]]], np.int32) #pixels that need to be whitened

    pts = pts.reshape((-1,1,2))

    cv2.fillPoly(dummy,[pts],(255,255,255)) #whitening those pixels

    #cv2.imwrite(path + 'dummy.jpg', dummy)

    outmark = np.zeros((im1.shape[0],im1.shape[1],3),dtype='uint8') #marking where pixels are taken from
from input im2

    #--------------------------------------------------------------------------------------------------

    #Filling im1 with distorted im2:

    for i in range(im1.shape[0]): #till 2709

        for j in range(im1.shape[1]): #till 3612
```

```
            if dummy[i][j][0] == 255: #change the contents
                I = i/scale + tx; J = j/scale + ty
                source = np.matmul(H, [[I], [J], [1]]);#print('source: ', source, '\n') #x' = Hx
                source = source / source[2][0]#; print(source)
                if source[0][0] > 0 and source[1][0] > 0 and source[0][0] < im2.shape[0] and source[1][0] < 
im2.shape[1]:
                    #print('[', source[0][0], source[1][0], source[2][0], ']', end = '')
                    #print(source)
                    im1[i][j] = im2[int(source[0][0]), int(source[1][0])]
                    outmark[int(source[0][0])][int(source[1][0])] = [255,255,255]
    cv2.imwrite(path + 'outmark.jpg', outmark)
    return im1
#---------------------------------------------------------------------------------------------------------
def calcvanishline(line1pts, line2pts, line3pts, line4pts):
    '''Each argument is a list of 2 3-element lists'''
    def calcvanishpt(line1pt, line2pt):
        line1 = np.cross(line1pt[0], line1pt[1]); line1 = line1/line1[2]
        line2 = np.cross(line2pt[0], line2pt[1]); line2 = line2/line2[2]
        vanishpt = np.cross(line1,line2); vanishpt = vanishpt/vanishpt[2]
        return vanishpt
    vanishline = np.cross(calcvanishpt(line1pts, line2pts), calcvanishpt(line3pts, line4pts)); vanishline = 
vanishline/vanishline[2]
    return vanishline
#---------------------------------------------------------------------------------------------------------
def findAffine(l, m, L, M):
    '''l and m are two perpendicular lines. L and M is another set of perpendicular lines'''
    P = [[l[0]*m[0], l[0]*m[1]+l[1]*m[0]], [L[0]*M[0], L[0]*M[1]+L[1]*M[0]]]
    b = [[-1*l[1]*m[1]], [-1*L[1]*M[1]]]
    x = np.matmul(np.linalg.inv(P), b)
    S = [[float(x[0]), float(x[1])], [float(x[1]), 1]]; print('S = ', S)
    val, vec = np.linalg.eig(S); print('values = ', val, 'vectors = ', vec)

    V = np.array([vec[0], vec[1]]).T

    D = [[math.sqrt(val[0]), 0], [0, math.sqrt(val[1])]]

    tmp = np.matmul(V, D)

    A = np.matmul(tmp, np.array(V).T)

    return A

#-----------------------------------------------------------------------------------------------------

def getAffine(p1,p2,q1,q2,r1,r2,s1,s2):

    '''lines p and q are perperndicular, so are lines r and s'''

    p = np.cross(p1,p2); p = p/p[2]

    q = np.cross(q1,q2); q = q/q[2]

    r = np.cross(r1,r2); r = r/r[2]

    s = np.cross(s1,s2); s = s/s[2]

    return findAffine(p,q,r,s)

#-----------------------------------------------------------------------------------------------------

def singleStep(l, m):

    A = []; b = []

    for i in range(5):

        L = l[i]; M = m[i]

        A.append([L[0]*M[0], (L[0]*M[1] + L[1]*M[0])/2, L[1]*M[1], (L[0]*M[2] + L[2]*M[0])/2,
(L[1]*M[2] + L[2]*M[1])/2])

        b.append(-1*L[2]*M[2])


    x = np.matmul(np.linalg.inv(A), b)

    C = [[x[0], x[1]/2, x[3]/2], [x[1]/2, x[2], x[4]/2], [x[3]/2, x[4]/2, 1]]

    S = [[C[0][0], C[0][1]], [C[1][0], C[1][1]]]

    U, d2, V = np.linalg.svd(S); print('d2 = ', d2)

    d = [[math.sqrt(d2[0]), 0], [1, math.sqrt(d2[1])]]

    A = np.matmul(np.matmul(V, d), np.array(V).T)

    v = np.matmul([C[2][0], C[2][1]], np.linalg.inv(A).T).T; print('v = ', v)
    H = [[A[0][0],A[0][1],0], [A[1][0],A[1][1],0], [v[0], v[1], 1]]

    return H

#---------------------------------------------------------------------------------------------------

def getSingleStep(x1,x2,x3,x4,x5,x6,x7,x8,x9,x10):

    l1 = np.cross(x1[0],x1[1]); l1 = l1/l1[2]

    l2 = np.cross(x2[0],x2[1]); l2 = l2/l2[2]

    l3 = np.cross(x3[0],x3[1]); l3 = l3/l3[2]

    l4 = np.cross(x4[0],x4[1]); l4 = l4/l4[2]

    l5 = np.cross(x5[0],x5[1]); l5 = l5/l5[2]

    m1 = np.cross(x6[0],x6[1]); m1 = m1/m1[2]

    m2 = np.cross(x7[0],x7[1]); m2 = m2/m2[2]

    m3 = np.cross(x8[0],x8[1]); m3 = m3/m3[2]

    m4 = np.cross(x9[0],x9[1]); m4 = m4/m4[2]

    m5 = np.cross(x10[0],x10[1]); m5 = m5/m5[2]

    return singleStep([l1,l2,l3,l4,l5], [m1,m2,m3,m4,m5])


#============================================================================
============================


if __name__ == '__main__':

    #Reading the images

    #img1 = cv2.imread(path + 'HW3Pics/1.jpg')

    img2 = cv2.imread(path + 'HW3Pics/2.jpg')


    #coordinates found using GIMP:

    img1PQRS = np.array([[820,1140], [760,1255], [944,1244], [988,1125]], dtype = float)

    alt_indices = np.array([[760,1258], [702,1385], [893,1374], [945,1244]], dtype = float)

    img2PQRS = np.array([[826,1263], [840,1368], [1047,1368], [1054,1263]], dtype = float)

    #img2PQRS = np.array([[332,1320], [608,3014], [1900,3030], [2012,1300]], dtype = float)

    #----------------------------Task1--------------------------------------------------------------------------
    '''#cv2.imwrite(path + 'result1.jpg', apply_homography(img1, img4, img1PQRS, img4PQRS))

    hwPQRS = np.array([[0,0], [0,40], [80,40], [80,0]], dtype = float) #height-width PQRS

    #hwPQRS = np.array([[0,0], [60,0], [60,80], [0,80]], dtype = float) #height-width PQRS

    #                    world   image

    H = find_homography(hwPQRS, img2PQRS); H = np.insert(H, 8, 1).reshape((3,3)); print('H = ', H)

    Hinv = np.linalg.inv(H)

    cornerpt = [0,0,0,0]

    cornerpt[0] = np.matmul(Hinv, [0,0,1]) ;cornerpt[0] /= cornerpt[0][2]

    cornerpt[1] = np.matmul(Hinv, [0,img2.shape[1],1]) ;cornerpt[1] /= cornerpt[1][2]

    cornerpt[2] = np.matmul(Hinv, [img2.shape[0],img2.shape[1],1]) ;cornerpt[2] /= cornerpt[2][2]

    cornerpt[3] = np.matmul(Hinv, [img2.shape[0],0,1]) ;cornerpt[3] /= cornerpt[3][2]

    to_add = [abs(cornerpt[0][0]), abs(cornerpt[0][1]), abs(cornerpt[0][2])]

    #to_add = [abs(min(cornerpt[:][0])), abs(min(cornerpt[:][1])), abs(min(cornerpt[:][2]))]#change

    print('cornerpts: ', cornerpt)

    for i in range(4):

        cornerpt[i] += to_add

        #cornerpt[i] /= cornerpt[i][2]

    print('new cornerpts: ', cornerpt)

    #print('new cornerpoints: ',new_cornerpt)

    cornerpt = list(map(lambda x: x[:-1], cornerpt))

    endPQRS = np.array([[0   ,0  ], [0, img2.shape[1]], [img2.shape[0], img2.shape[1]], [img2.shape[0],0
]], dtype = float) #ends of the image

    newH = find_homography(cornerpt, endPQRS); newH = np.insert(newH, 8, 1).reshape((3,3))

    #print('new H: ', newH)

    blank = np.zeros((img2.shape[0],img2.shape[1],3),dtype='uint8') #blank image


    cv2.imwrite(path + 'task1results/result2.jpg', find_result(blank, img2, cornerpt, newH))

    #cv2.imwrite(path + 'part_b.jpg', find_result(blank, img1, endPQRS, np.matmul(Hab, Hbc)))'''

    #-----------------------------Task2-----------------------------------------------------------------------------

    #--------------------------eliminating projective transformation----------------------------
```

ECE 661 HW03
Rehana Mahfuz
rmahfuz@purdue.edu

```
    a = [840,1368,1]; b =
[1047,1367,1];c=[826,1263,1];d=[1054,1264,1];e=[932,1469,1];f=[991,1468,1];g=[992,1435,1]

    #line1pts = [[408,2040,1], [620,2044,1]]

    #line2pts = [[1244,244,1], [1352,224,1]]

    #line3pts = [[408,2040,1], [1244,244,1]]

    #line4pts = [[620,2044,1], [1352,224,1]]

    line1pts = [a,b]

    line2pts = [c,d]

    line3pts = [a,c]

    line4pts = [b,d]


    vanishline = calcvanishline(line1pts, line2pts, line3pts, line4pts); #print(vanishline)

    H = [[1,0,0],[0,1,0], vanishline.tolist()]; print('H = ', H)


    #Hinv = np.linalg.inv(H)

    Hinv = H

    cornerpt = [0,0,0,0] #cornerpoints of world

    cornerpt[0] = np.matmul(Hinv, [0,0,1]) ;cornerpt[0] /= cornerpt[0][2]

    cornerpt[1] = np.matmul(Hinv, [0,img2.shape[1],1]) ;cornerpt[1] /= cornerpt[1][2]

    cornerpt[2] = np.matmul(Hinv, [img2.shape[0],img2.shape[1],1]) ;cornerpt[2] /= cornerpt[2][2]

    cornerpt[3] = np.matmul(Hinv, [img2.shape[0],0,1]) ;cornerpt[3] /= cornerpt[3][2]

    #print('multiplying ', Hinv, 'and ', [img1.shape[0],0,1], 'gives ', np.matmul(Hinv,
[img1.shape[0],0,1]),'\n\n\n')

    print('cornerpts: ', cornerpt,'\n\n')

    #to_add = [abs(cornerpt[0][0]), abs(cornerpt[0][1]), abs(cornerpt[0][2])]

    #to_add = [abs(min(cornerpt[:][0])), abs(min(cornerpt[:][1])), abs(min(cornerpt[:][2]))]#change

    #print(type(cornerpt), type(cornerpt[0]))

    trans = np.array(cornerpt).T;

    to_add = [abs(min(trans[0])), abs(min(trans[1])), abs(min(trans[2]))]

    print('to_add = ', to_add, '\n\n')
    for i in range(4):

        cornerpt[i] += to_add

    cornerpt = list(map(lambda x: x[:-1], cornerpt))

    print('new cornerpts: ', cornerpt, '\n\n')

    trans2 = np.array(cornerpt).T

    scale1 = img2.shape[0]/(max(trans2[0]) )#- min(trans2[0])); print('scale1 = ', scale1)

    scale2 = img2.shape[1]/(max(trans2[1]) )#- min(trans2[1])); print('scale2 = ', scale2)

    if scale1 < scale2:

        scale = scale1

    else:

        scale = scale2

    tx = min(trans2[0])

    ty = min(trans2[1])

    for i in range(4):

        cornerpt[i] = list(map(lambda x: x*scale, cornerpt[i]))

    print('new new cornerpts: ', cornerpt)

    endPQRS = np.array([[0   ,0  ], [0, img2.shape[1]], [img2.shape[0], img2.shape[1]], [img2.shape[0],0
]], dtype = float) #ends of the image

    newH = find_homography(cornerpt, endPQRS); newH = np.insert(newH, 8, 1).reshape((3,3))


    blank = np.zeros((img2.shape[0],img2.shape[1],3),dtype='uint8') #blank image

    projective_free = find_result(blank, img2, endPQRS, newH)#, scale, tx, ty)

    #cv2.imwrite(path + 'task2results/projective_free_img2.jpg', projective_free)

    #-------------------------eliminating affine transformation---------------------------

    #A = getAffine([408,2040,1], [620,2044,1], [408,2040,1], [1244,244,1], [1244,244,1], [1352,224,1],
[1352,224,1], [620,2044,1])

    a = [840,1368,1]; b =
[1047,1367,1];c=[826,1263,1];d=[1054,1264,1];e=[932,1469,1];f=[991,1468,1];g=[992,1435,1]

    A = getAffine(a,b,a,c,a,b,b,d)

    #A = getAffine([1113,237,1], [9,2208,1], [1113,237,1], [1746,108,1], [1401,2334,1], [1746,108,1],
[9,2208,1], [1401,2334,1])
```

```
    #A = getAffine([400,2036,1], [1616,2092,1], [400,2036,1], [1244,248,1], [1244,248,1], [1820,128,1],
[1820,128,1], [1616,2092,1])

    #print('A = ', A)

    H2 = [[A[0][0], A[0][1], 0], [A[1][0], A[1][1], 0], [0, 0, 1]]

    #Hinv = np.matmul(np.linalg.inv(H2), H)

    Hinv = np.matmul(H2, H)




    old_corner = cornerpt

    #Hinv = np.linalg.inv(H)

    cornerpt = [0,0,0,0]

    #cornerpt[0] = np.matmul(Hinv, [old_corner[0][0], old_corner[0][1], 1]) ;cornerpt[0] /= cornerpt[0][2]

    #cornerpt[1] = np.matmul(Hinv, [old_corner[1][0], old_corner[1][1], 1]) ;cornerpt[1] /= cornerpt[1][2]

    #cornerpt[2] = np.matmul(Hinv, [old_corner[2][0], old_corner[2][1], 1]) ;cornerpt[2] /= cornerpt[2][2]

    #c3ornerpt[3] = np.matmul(Hinv, [old_corner[3][0], old_corner[3][1], 1]) ;cornerpt[3] /=
cornerpt[3][2]


    cornerpt[0] = np.matmul(Hinv, [0,0,1]) ;cornerpt[0] /= cornerpt[0][2]

    cornerpt[1] = np.matmul(Hinv, [0,img2.shape[1],1]) ;cornerpt[1] /= cornerpt[1][2]

    cornerpt[2] = np.matmul(Hinv, [img2.shape[0],img2.shape[1],1]) ;cornerpt[2] /= cornerpt[2][2]

    cornerpt[3] = np.matmul(Hinv, [img2.shape[0],0,1]) ;cornerpt[3] /= cornerpt[3][2]


    trans = np.array(cornerpt).T;

    to_add = [abs(min(trans[0])), abs(min(trans[1])), abs(min(trans[2]))]

    #print('to_add = ', to_add, '\n\n')

    for i in range(4):

        cornerpt[i] += to_add

    cornerpt = list(map(lambda x: x[:-1], cornerpt))

    #print('new cornerpts: ', cornerpt, '\n\n')

    trans2 = np.array(cornerpt).T
    scale1 = img2.shape[0]/(max(trans2[0]) )#- min(trans2[0])); print('scale1 = ', scale1)

    scale2 = img2.shape[1]/(max(trans2[1]) )#- min(trans2[1])); print('scale2 = ', scale2)

    if scale1 < scale2:

        scale = scale1

    else:

        scale = scale2


    for i in range(4):

        cornerpt[i] = list(map(lambda x: x*scale, cornerpt[i]))

    print('new new cornerpts: ', cornerpt)


    endPQRS = np.array([[0   ,0  ], [0, img2.shape[1]], [img2.shape[0], img2.shape[1]], [img2.shape[0],0
]], dtype = float) #ends of the image

    newH = find_homography(cornerpt, endPQRS); newH = np.insert(newH, 8, 1).reshape((3,3))

    #print('new H: ', newH)

    blank = np.zeros((img2.shape[0],img2.shape[1],3),dtype='uint8') #blank image


    affine_free = find_result(blank, img2, endPQRS,newH)

    cv2.imwrite(path + 'task2results/affine_freeimg2.jpg', affine_free)

    #-----------------------------Task3----------------------------------------------------------------------------

    '''#H =
getSingleStep([[400,2036,1],[1616,2092,1]],[[400,2036,1],[1616,2092,1]],[[1244,248,1],[1820,128,1]],

    #                 [[1244,248,1],[1820,128,1]],[[408,2040,1],[620,2044,1]],

    #                 [[400,2036,1],[1244,248,1]], [[1616,2092,1],[1820,128,1]],[[400,2036,1],[1244,248,1]]    ,

    #                 [[1616,2092,1],[1820,128,1]],[[408,2040,1],[1244,244,1]]) #for image 1

    a = [840,1368,1]; b =
[1047,1367,1];c=[826,1263,1];d=[1054,1264,1];e=[932,1469,1];f=[991,1468,1];g=[992,1435,1]

    H = getSingleStep([a,b],[a,b],[c,d],[c,d],[g,f],[a,c],[b,d],[a,c],[b,d],[e,f]) #for image 2


    print('H = ', H)

    Hinv = np.linalg.inv(H)
    cornerpt = [0,0,0,0]
    cornerpt[0] = np.matmul(Hinv, [0,0,1]) ;cornerpt[0] /= cornerpt[0][2]
    cornerpt[1] = np.matmul(Hinv, [0,img2.shape[1],1]) ;cornerpt[1] /= cornerpt[1][2]
    cornerpt[2] = np.matmul(Hinv, [img2.shape[0],img2.shape[1],1]) ;cornerpt[2] /= cornerpt[2][2]
    cornerpt[3] = np.matmul(Hinv, [img2.shape[0],0,1]) ;cornerpt[3] /= cornerpt[3][2]


    trans = np.array(cornerpt).T;
    to_add = [abs(min(trans[0])), abs(min(trans[1])), abs(min(trans[2]))]
    #print('to_add = ', to_add, '\n\n')
    for i in range(4):
        cornerpt[i] += to_add
    cornerpt = list(map(lambda x: x[:-1], cornerpt))
    #print('new cornerpts: ', cornerpt, '\n\n')
    trans2 = np.array(cornerpt).T
    scale1 = img2.shape[0]/(max(trans2[0]) )#- min(trans2[0])); print('scale1 = ', scale1)
    scale2 = img2.shape[1]/(max(trans2[1]) )#- min(trans2[1])); print('scale2 = ', scale2)
    if scale1 < scale2:
        scale = scale1
    else:
        scale = scale2


    for i in range(4):
        cornerpt[i] = list(map(lambda x: x*scale, cornerpt[i]))
    print('new new cornerpts: ', cornerpt)


    endPQRS = np.array([[0   ,0  ], [0, img2.shape[1]], [img2.shape[0], img2.shape[1]], [img2.shape[0],0
]], dtype = float) #ends of the image
    newH = find_homography(cornerpt, endPQRS); newH = np.insert(newH, 8, 1).reshape((3,3))
    #print('new H: ', newH)
    blank = np.zeros((img2.shape[0],img2.shape[1],3),dtype='uint8') #blank image


    one_step = find_result(blank, img2, endPQRS, newH)

    cv2.imwrite(path + 'task3results/result2.jpg', one_step)'''
```