ECE661: Homework 2

Wan-Eih Huang

September 1, 2018

1 Logic and Method

There are two things to do in this assignment. First, find homography between two images. Second, find the pixel value in a image from the corresponding points in the other image. And replace them by the estimated pixel value.

1.1 Compute Homography

Because homography is homogeneous; that is, only the ratio matters. We can represent it as following.

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix}$$

And for each pixel (x, y) in physical world can be mapped to $(\frac{x'}{z'}, \frac{y'}{z'})$, where

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = H \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Hence, we can express the new points in physical world as

$$\begin{aligned} x_{new} = \frac{x'}{z'} &= \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + 1} \\ \implies x_{new} &= h_{11}x + h_{12}y + h_{13} - h_{31}xx_{new} - h_{32}yx_{new} \\ y_{new} = \frac{y'}{z'} &= \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + 1} \\ \implies y_{new} &= h_{21}x + h_{22}y + h_{23} - h_{31}xy_{new} - h_{32}yy_{new} \end{aligned}$$

In H, we have 8 unknowns. Thus, we need at least 4 pixels to solve the equations. Rewrite the equations as matrix multiplication form as following.

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_1' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_1' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_3' & -y_3y_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y_4' & -y_4y_4' \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4' \\ y_4' \end{bmatrix}$$

$$\implies Ah = b$$

So, we can compute A^{-1} to solve h which is the homography matrix.
Note: $(x'_n, y'_n), n = 1, 2, 3, 4$. They are the 4 points in physical world, not homogeneous coordinate.

1.2 Weighted Sum of Neighboring Pixels

It is inevitable that the coordinates of new estimated pixels are not integer after mapping. However, the value of pixels are stored in integer coordinate form. To address this problem, we adopt weighted sum of four neighboring pixels of the estimated pixel coordinate. The weights are obtained by the reciprocal of Euclidean distance between the neighboring pixel and estimated pixel. That is, if the estimated pixel is closer to one of the four pixels, the weighted value of that neighboring pixel is higher. It means the value of estimated pixel should be close to that neighboring pixel.

$$\text{Estimated pixel value} = \frac{W_a * a + W_b * b + W_c * c + W_d * d}{W_a + W_b + W_c + W_d},$$

where a, b, c, d are the pixels' RGB value. And $W_n = \frac{1}{d_n}, n = a, b, c, d.$

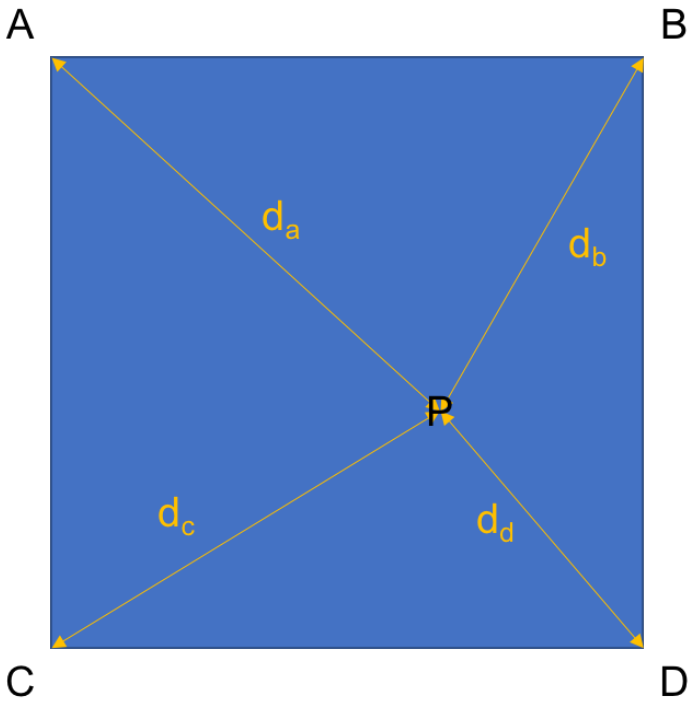


Figure 1: Weighted sum of neighboring pixels.

1.3 Steps of Implementation

1.3.1 Task 1

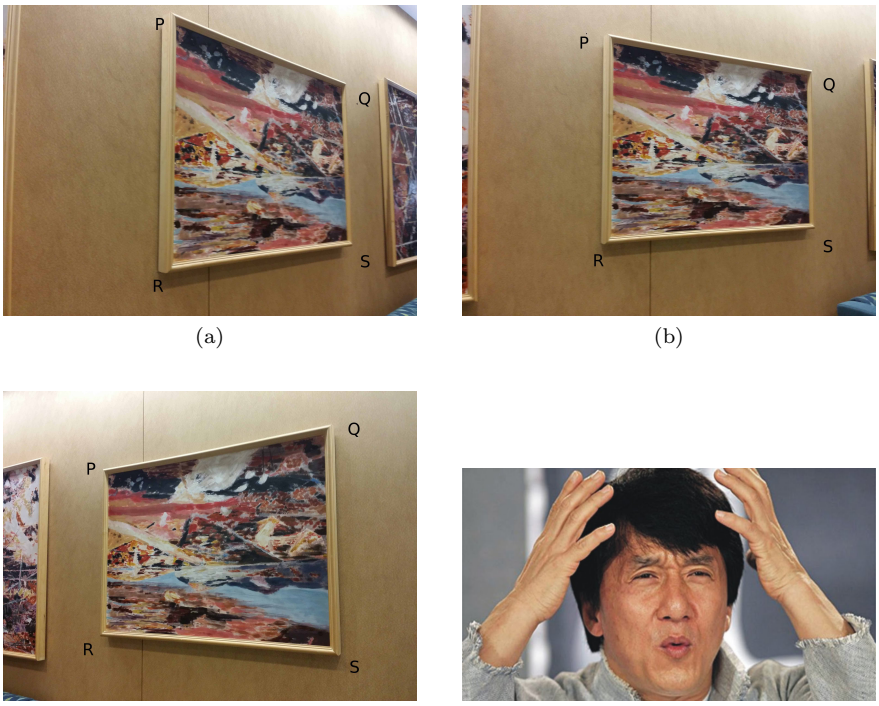
Project the face shown in Fig. 1d on the frame PQRS shown in Figs. 1a, 1b, and 1c. Here we take Fig. 1a as an example to describe the steps. The work is the same to Figs. 1b and 1c.
Step 1: Find homography matrix H that can map the corner points of the frame in Fig. 1a to the corner points of the whole image in Fig. 1d.
Step 2: Using H to find the corresponding points in Fig. 1d by mapping the points inside the frame in Fig. 1a.
Step 3: The estimated points can be non-integer. Using weighted sum of 4 neighboring pixels in Fig. 1d to replace the pixel value in Fig. 1a.

1.3.2 Task 2

Find homographies between Figs. 1a and 1b, and between Figs. 1b and 1c. Then apply the product of the two homographies to Fig. 1a.
Step 1: Find homography H_1 that can map the points in Fig. 1b to Fig. 1a and homography H_2 that can map the points in Fig. 1c to Fig. 1b.
Step 2: Matrix multiplication of H_1 and H_2.
Step 3: Project the image in Fig. 1a on Fig. 1c.

2 Result

2.1 The images provided by course website



(a) (b)

(c) (d)

Figure 2: Original Images

Figure 3: The result of projecting the face on Fig. 1a.

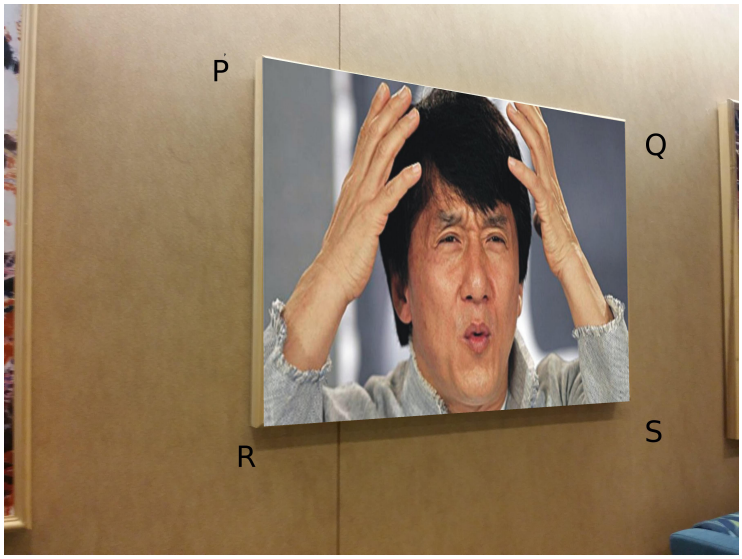
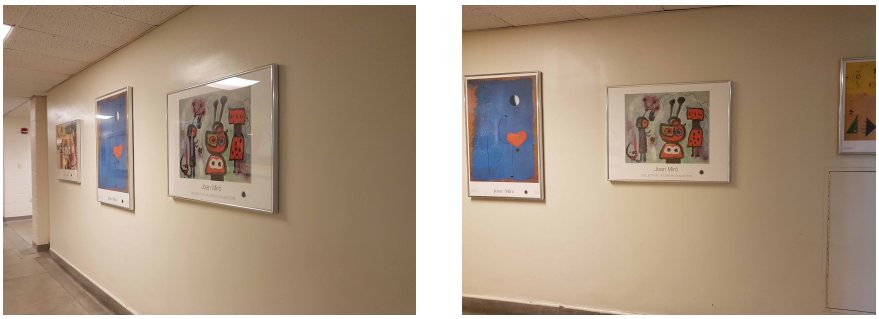


Figure 4: The result of projecting the face on Fig. 1b.

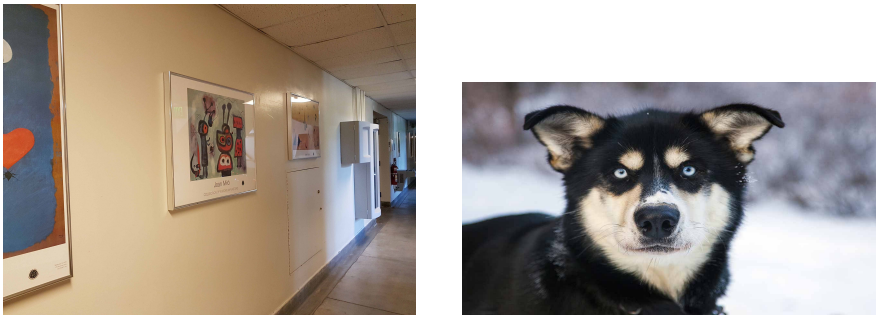
Figure 5: The result of projecting the face on Fig. 1c.

Figure 6: The result of projecting Fig. 1a on Fig. 1c.

2.2 My own images



(a) (b)



(c) (d)

Figure 7: My Original Images

Figure 8: The result of projecting the face on Fig. 1a.

Figure 9: The result of projecting the face on Fig. 1b.

Figure 10: The result of projecting the face on Fig. 1c.

Figure 11: The result of projecting Fig. 1a on Fig. 1c.

3 Source Code

3.1 Task 1

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 28 22:13:29 2018
ECE661 HW2-task1: Homography
@author: wehuang
"""

import numpy as np
import cv2
import matplotlib.pyplot as plt
import skimage.io as sio

def get_pixelVal(img, pt):
    aVal = img[np.int32(np.floor(pt[1]))][np.int32(np.floor(pt
        [0]))] #[y][x]
    bVal = img[np.int32(np.floor(pt[1]))][np.int32(np.ceil(pt
        [0]))]
    cVal = img[np.int32(np.ceil(pt[1]))][np.int32(np.floor(pt
        [0]))]
    dVal = img[np.int32(np.ceil(pt[1]))][np.int32(np.ceil(pt
        [0]))]

    dx = pt[0]-np.floor(pt[0])
    dy = pt[1]-np.floor(pt[1])

    aWt = 1/np.linalg.norm([dx, dy])
    bWt = 1/np.linalg.norm([1-dx, dy])
    cWt = 1/np.linalg.norm([dx, 1-dy])
    dWt = 1/np.linalg.norm([1-dx, 1-dy])

    pixelVal = (aWt*aVal+bWt*bVal+cWt*cVal+dWt*dVal)/(aWt+bWt+
        cWt+dWt)

    return pixelVal

def find_homography(src_pts, dst_pts):
    A = np.zeros((8,8))
    b = dst_pts.reshape((8,1))

    for i in range(src_pts.shape[0]):
        a_tmp = np.concatenate((src_pts[i], np.array([1, 0, 0,
            0])), axis=None)
        A[i*2] = np.concatenate((a_tmp, np.array(-src_pts[i]*
            dst_pts[i][0])), axis=None)
        A[i*2+1] = np.concatenate((np.roll(a_tmp,3), np.array
            (-src_pts[i]*dst_pts[i][1])), axis=None)

    A_inv = np.linalg.pinv(A)
    h = np.concatenate((np.dot(A_inv, b), 1), axis=None)
```

```
    H = h.reshape((3,3))

    return H

def project(dst_img, src_img, vertices, H):
    # Create a mask for dst image area
    mask = np.zeros(dst_img.shape[0:2], dtype=np.uint8)
    cv2.fillPoly(mask, [np.int32(vertices)], 255)
    # Map the source image to destination image
    img = dst_img
    for j in range(mask.shape[0]): #y
        for i in range(mask.shape[1]): #x
            if mask[j][i] == 255:
                pt_hc = np.dot(H, np.array([i,j,1])) # HC
                    representation
                pt_pos = np.array([pt_hc[0]/pt_hc[2], pt_hc
                    [1]/pt_hc[2]])
                # Get value from src image
                if pt_pos[0] > 0 and pt_pos[0] < (src_img.
                    shape[1]-1) and pt_pos[1] > 0 and pt_pos
                    [1] < (src_img.shape[0]-1):
                    img[j][i] = get_pixelVal(src_img, pt_pos)

    return img

# Main program starts from here!
filename = ['PicsHw2/1.jpg', 'PicsHw2/2.jpg', 'PicsHw2/3.jpg']
savename = ['task1_dtoa.jpg', 'task1_dtob.jpg', 'task1_dtoc.
    jpg']
PQSR_img1 = np.array([[1460, 66], [2990, 680], [3050, 2099],
    [1430, 2336]])
PQSR_img2 = np.array([[1270, 262], [3050, 580], [3073, 1952],
    [1277,2076]])
PQSR_img3 = np.array([[874, 691], [2864, 306], [2897, 2307],
    [848, 2142]])
PQSR = np.concatenate((PQSR_img1, PQSR_img2, PQSR_img3), axis
    =0)

# Image d will be projected on image a/b/c
imgd = sio.imread('PicsHw2/Jackie.jpg')
PQSR_imgd = np.array([[0, 0], [imgd.shape[1]-1, 0], [imgd.
    shape[1]-1, imgd.shape[0]-1], [0, imgd.shape[0]-1]])
for i in range(3):
    # Read image
    dst_img = sio.imread(filename[i])
    # Find homography
    H = find_homography(PQSR[i*4:i*4+4], PQSR_imgd)
    # Map the points
    img = project(dst_img, imgd, PQSR[i*4:i*4+4], H)

    plt.imshow(img)
    plt.show()
    sio.imsave(savename[i], img)
```

3.2 Task 2

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Aug 29 17:39:33 2018
ECE661 HW2-task2: Homography
@author: wehuang
"""

import numpy as np
import cv2
import matplotlib.pyplot as plt
import skimage.io as sio

def get_pixelVal(img, pt):
    aVal = img[np.int32(np.floor(pt[1]))][np.int32(np.floor(pt
        [0]))] #[y][x]
    bVal = img[np.int32(np.floor(pt[1]))][np.int32(np.ceil(pt
        [0]))]
    cVal = img[np.int32(np.ceil(pt[1]))][np.int32(np.floor(pt
        [0]))]
    dVal = img[np.int32(np.ceil(pt[1]))][np.int32(np.ceil(pt
        [0]))]

    dx = pt[0]-np.floor(pt[0])
    dy = pt[1]-np.floor(pt[1])

    aWt = 1/np.linalg.norm([dx, dy])
    bWt = 1/np.linalg.norm([1-dx, dy])
    cWt = 1/np.linalg.norm([dx, 1-dy])
    dWt = 1/np.linalg.norm([1-dx, 1-dy])

    pixelVal = (aWt*aVal+bWt*bVal+cWt*cVal+dWt*dVal)/(aWt+bWt+
        cWt+dWt)

    return pixelVal

def find_homography(src_pts, dst_pts):
    A = np.zeros((8,8))
    b = dst_pts.reshape((8,1))

    for i in range(src_pts.shape[0]):
        a_tmp = np.concatenate((src_pts[i], np.array([1, 0, 0,
            0])), axis=None)
        A[i*2] = np.concatenate((a_tmp, np.array(-src_pts[i]*
            dst_pts[i][0])), axis=None)
        A[i*2+1] = np.concatenate((np.roll(a_tmp,3), np.array
            (-src_pts[i]*dst_pts[i][1])), axis=None)

    A_inv = np.linalg.pinv(A)
    h = np.concatenate((np.dot(A_inv, b), 1), axis=None)
    H = h.reshape((3,3))
```

```
    return H

def project(dst_img, src_img, vertices, H):
    # Create a mask for dst image area
    mask = np.zeros(dst_img.shape[0:2], dtype=np.uint8)
    cv2.fillPoly(mask, [np.int32(vertices)], 255)
    # Map the source image to destination image
    img = np.zeros(dst_img.shape, dtype=np.uint8)
    for j in range(mask.shape[0]): #y
        for i in range(mask.shape[1]): #x
            if mask[j][i] == 255:
                pt_hc = np.dot(H, np.array([i,j,1])) # HC
                    representation
                pt_pos = np.array([pt_hc[0]/pt_hc[2], pt_hc
                    [1]/pt_hc[2]])
                # Get value from src image
                if pt_pos[0] > 0 and pt_pos[0] < (src_img.
                    shape[1]-1) and pt_pos[1] > 0 and pt_pos
                    [1] < (src_img.shape[0]-1):
                    img[j][i] = get_pixelVal(src_img, pt_pos)

    return img

# Main program starts from here!
# Read input images
img1 = sio.imread('PicsHw2/1.jpg')
img2 = sio.imread('PicsHw2/2.jpg')
img3 = sio.imread('PicsHw2/3.jpg')

PQSR_img1 = np.array([[1460, 66], [2990, 680], [3050, 2099],
    [1430, 2336]])
PQSR_img2 = np.array([[1270, 262], [3050, 580], [3073, 1952],
    [1277,2076]])
PQSR_img3 = np.array([[874, 691], [2864, 306], [2897, 2307],
    [848, 2142]])

# Map img2 to img1
H1 = find_homography(PQSR_img2, PQSR_img1)
# Map img3 to img2
H2 = find_homography(PQSR_img3, PQSR_img2)
H = np.dot(H1, H2)

# Project img1 on img3
img = project(img3, img1, np.array([[0, 0], [img3.shape[1]-1,
    0], [img3.shape[1]-1, img3.shape[0]-1], [0, img3.shape
    [0]-1]]), H)

plt.imshow(img)
plt.show()
sio.imsave('task2.jpg', img)
```