# ECE 661 Homework 2

Yifan Fei

Electrical Engineering
Purdue University

Instructor: Prof. Avinash Kak

Fall 2018

# Homography

## Theory Demonstration and Mathematical Calculation

A planar projective transformation is a linear transformation on homogeneous 3-vectors, the transformation being represented by a non-singular 3 by 3 matrix $H$, as is:

$$X' = HX \tag{1}$$

where

$$X' = \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} \tag{2}$$

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{3}$$

$$H = \begin{bmatrix} h_1 1 & h_1 2 & h_1 3 \\ h_2 1 & h_2 2 & h_2 3 \\ h_3 1 & h_3 2 & h_3 3 \end{bmatrix} \tag{4}$$

Here We can set $h_{33} = 1$ since we only care ratios. Considering two points on a physical coordinates (x, y) and (x', y'), we have:

$$x = \frac{x_1}{x_3}, y = \frac{y_1}{y_3}, x' = \frac{x'_1}{x'_3}, y' = \frac{y'_1}{y'_3}$$

Plug into equation 1 we can get:

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

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

which means that for a pair of points, we get two equations. So for 8 unknown variables we need four pairs of points. Finally we arrive at the equation $b = Ah$:

$$\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x'_3 & -y_3x'_3 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_3y'_3 & -y_3y'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x'_4 & -y_4x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y'_4 & -y_4y'_4 \end{bmatrix} \begin{bmatrix} h_1 1 \\ h_1 2 \\ h_1 3 \\ h_2 1 \\ h_2 2 \\ h_2 3 \\ h_3 1 \\ h_3 2 \end{bmatrix} \tag{5}$$

i.e: $h = A^{-1}b$, and in this way homography is found between two planes by four pairs of points.

## Method to determine pixel values

I try to use round down values(math.floor in Python), though it is not very accurate, it is quite fast.

**Task 1(a)**


Figure 1: input image 1


Figure 2: input image 2


Figure 3: input image 3

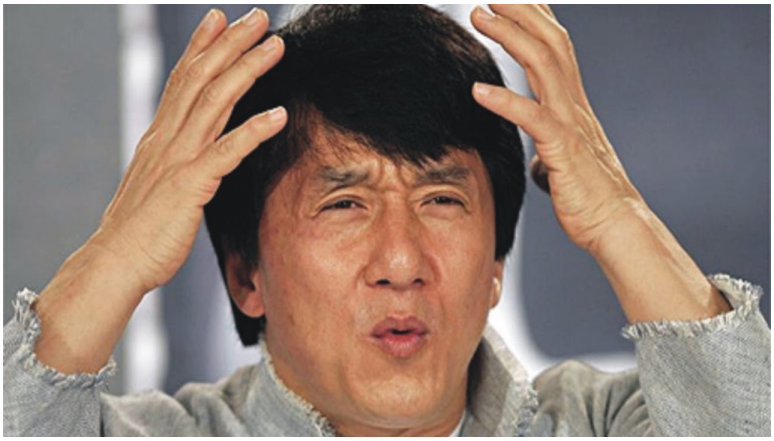

Figure 4: input image of Jackie


Figure 5: The image of Jackie into 1.jpg, task 1(a)-1

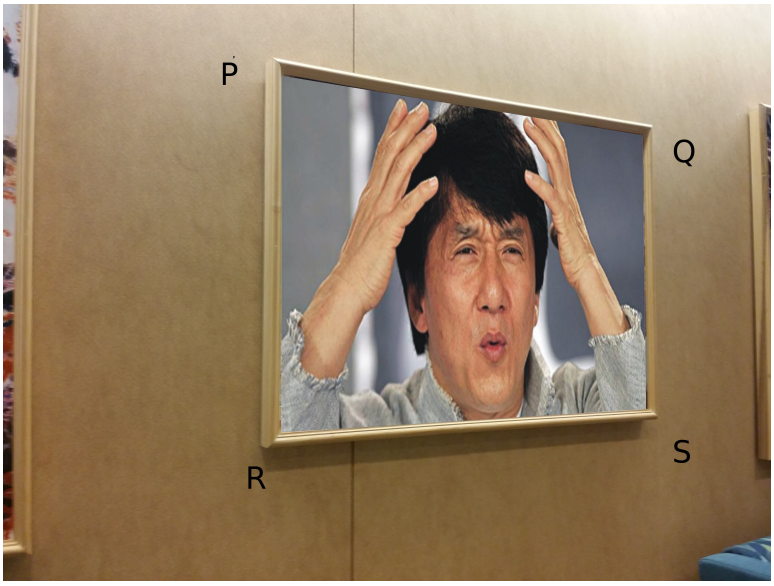


Figure 6: The image of Jackie into 2.jpg, task 1(a)-2


Figure 7: The image of Jackie into 3.jpg, task 1(a)-3

## Task 1(b)

The task is about projecting plane 1 to plane 3.

1. find $H_{ab}$ such that $x'' = H_{ab}x'$ where $x'', x'$ are HC representation of PQRS in plane 1 and plane 2.
2. find $H_{bc}$ such that $x' = H_{ab}x$ where $x', x$ are HC representation of PQRS in plane 2 and plane 3.
3. $H_{ac} = H_{ab}H_{bc}$ Apply $H_{ac}$ to image 1 to get the result which is similar with image 3.

Here is the result. After projection, the resulting image is very similar to figure 1c.

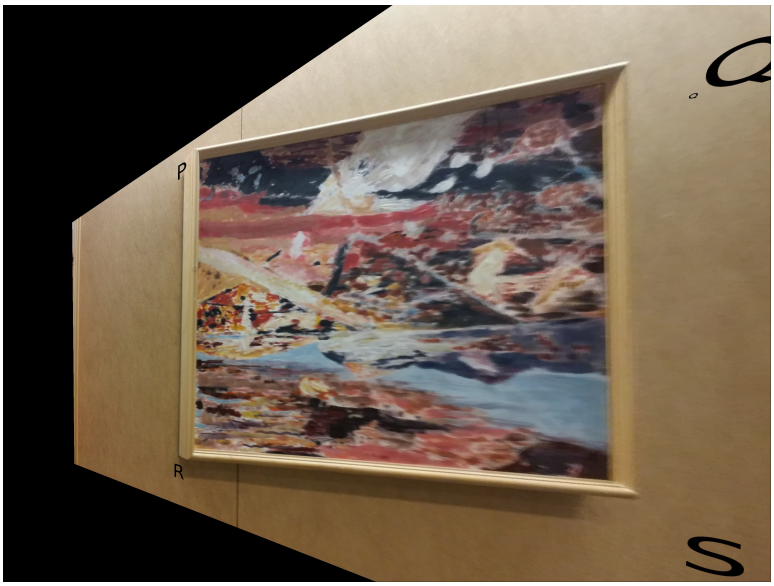


Figure 8: The image that represents 1a to 1c, task 1(b)

## Task 2: Play with my own images

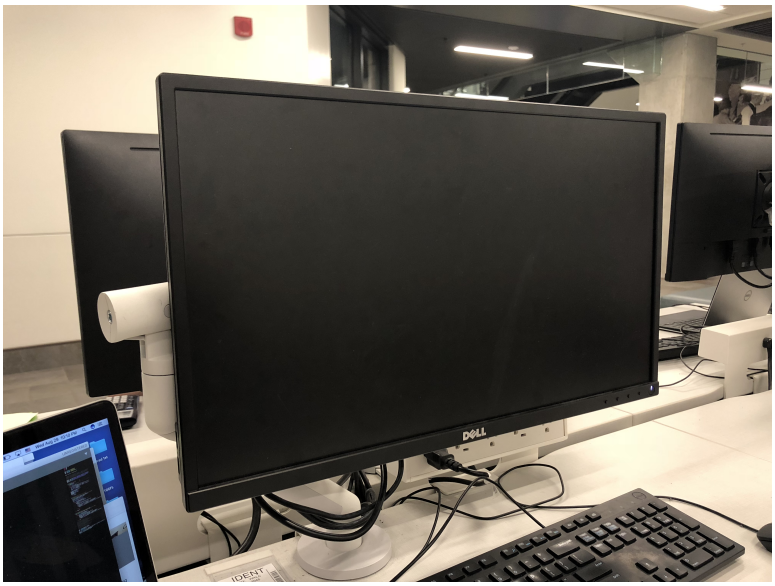


Figure 9: input image 1


Figure 10: input image 2


Figure 11: input image 3


Figure 12: input image of a cat


Figure 13: The image of cat into myimage1.jpg


Figure 14: The image of cat into myimage2.jpg


Figure 15: The image of cat into myimage3.jpg

Figure 16: The image that represents 1a to 1c

# Python Scripts

## task1

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

img1 = cv2.imread('../PicsHw2/1.jpg')
```

```
img2 = cv2.imread('../PicsHw2/2.jpg')
img3 = cv2.imread('../PicsHw2/3.jpg')
imgJ = cv2.imread('../PicsHw2/Jackie.jpg')


# Homogeneous points obtained from gimp
Points_1a = np.array([[176,1504,1], [2248,1492,1], [728,2948,1], [2056,3000,1]])
Points_1b = np.array([[336,1308,1], [2016,1308,1], [616,3008,1], [1904,3024,1]])
Points_1c = np.array([[736,920,1], [2088,904,1], [384,2808,1], [2240,2848,1]])
Points_1d = np.array([[0,0,1], [722,0,1], [0,1280,1], [722,1280,1]])


def Homography(pt, pt_prime):
        # Finding Homography A of Ax=b
    A = np.zeros((8,8))
    b = np.zeros((1,8))

    for i in range(0, len(pt)):
        A[i * 2] = [pt[i][0], pt[i][1], pt[i][2], 0, 0, 0,\
        (-1 * pt[i][0] * pt_prime[i][0]), (-1 * pt[i][1] * pt_prime[i][0])]
        A[i * 2 + 1] = [0,0,0,pt[i][0],pt[i][1],pt[i][2],\
        (-1 * pt[i][0] * pt_prime[i][1]), (-1 * pt[i][1] * pt_prime[i][1])]
        b[0][i * 2] = pt_prime[i][0]
        b[0][i * 2 + 1] = pt_prime[i][1]

    h = np.matmul(np.linalg.pinv(A),b.T)
    homography = np.zeros((3,3))
    homography[0] = h[0:3,0]
    homography[1] = h[3:6,0]
    homography[2][0:2] = h[6:8,0]
    homography[2][2] = 1

    return homography


def projection(image_src,image_target,pt,Homography):
    # initialize pixel as black
    temp = np.zeros((image_src.shape[0],image_src.shape[1],3), dtype='uint8')
    pts = np.array([[pt[0][1], pt[0][0]], [pt[1][1],pt[1][0]],\
    [pt[3][1], pt[3][0]], [pt[2][1], pt[2][0]]])
    # make the target area white
    cv2.fillPoly(temp,[pts],(255,255,255))

    for i in range(0,(image_src.shape[0]-1)):
        for j in range(0,(image_src.shape[1]-1)):
            if temp[i,j,1] == 255 and temp[i,j,0] == 255 and \
            temp[i,j,2]==255:
                point_tmp = np.array([i, j, 1])
                trans_coord = np.array(np.dot(Homography,point_tmp))
                # rescale then
                trans_coord = trans_coord/trans_coord[2]
                if (trans_coord[0] > 0) and (trans_coord[0] < image_target.shape[0]) and \
                (trans_coord[1] > 0) and (trans_coord[1] < image_target.shape[1]):
                    image_src[i][j] = image_target[math.floor(trans_coord[0]),math.floor(trans_coord[1])
```

```
                    # temp[i][j] = image_target[math.floor(trans_coord[0]),math.floor(trans_coord[1])]
            else:
                continue

    return image_src


def projection2(image_src,image_target,pt,Homography):
    temp = np.zeros((image_src.shape[0],image_src.shape[1],3), dtype='uint8')

    for i in range(0,(image_src.shape[0]-1)):
        for j in range(0,(image_src.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] < image_target.shape[0]) and \
            (trans_coord[1] > 0) and (trans_coord[1] < image_target.shape[1]):
                temp[i][j] = image_target[math.floor(trans_coord[0]),math.floor(trans_coord[1])]

    return temp


def write_img(pt1,pt2,output_file,input_img,imgJ):
        # Transforming the image in 1d to 1abc
        H = Homography(pt1, pt2)
        output = projection(input_img, imgJ, pt1, H)
        cv2.imwrite(output_file, output)


def write_img_a2c():
        # Transforming the image from a to c
        H_ab = Homography(Points_1b, Points_1a)
        H_bc = Homography(Points_1c, Points_1b)
        H_ac = np.matmul(H_ab, H_bc)

        Points_13 = np.array([[0,0],[0, img3.shape[0]],[img3.shape[1],0],[img3.shape[1],img3.shape[0]]])
        output13 = projection2(img3,img1,Points_13,H_ac)
        cv2.imwrite('../PicsHw2/image13.jpg',output13)


def main():
        write_img(Points_1a,Points_1d,'../PicsHw2/image1.jpg',img1,imgJ)
        write_img(Points_1b,Points_1d,'../PicsHw2/image2.jpg',img2,imgJ)
        write_img(Points_1c,Points_1d,'../PicsHw2/image3.jpg',img3,imgJ)
        write_img_a2c()


if __name__ == '__main__':
    main()
```

## task2

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


img1 = cv2.imread('../PicsHw2/my1.jpeg')
img2 = cv2.imread('../PicsHw2/my2.jpeg')
img3 = cv2.imread('../PicsHw2/my3.jpeg')
myimg = cv2.imread('../PicsHw2/myimg.jpeg')

Points_1a = np.array([[448,908,1], [2524,1040,1], [628,3420,1], [1976,3396,1]])
Points_1b = np.array([[284,676,1], [1892,784,1], [268,3612,1], [1868,3508,1]])
Points_1c = np.array([[584,280,1], [2200,352,1], [216,3540,1], [2672,3468,1]])
Points_1d = np.array([[0,0,1], [640,0,1], [0,669,1], [640,669,1]])


def Homography(pt, pt_prime):
    # Finding Homography A of Ax=b
    A = np.zeros((8,8))
    b = np.zeros((1,8))

    for i in range(0, len(pt)):
        A[i * 2] = [pt[i][0], pt[i][1], pt[i][2], 0, 0, 0,\
        (-1 * pt[i][0] * pt_prime[i][0]), (-1 * pt[i][1] * pt_prime[i][0])]
        A[i * 2 + 1] = [0,0,0,pt[i][0],pt[i][1],pt[i][2],\
        (-1 * pt[i][0] * pt_prime[i][1]), (-1 * pt[i][1] * pt_prime[i][1])]
        b[0][i * 2] = pt_prime[i][0]
        b[0][i * 2 + 1] = pt_prime[i][1]

    h = np.matmul(np.linalg.pinv(A),b.T)
    homography = np.zeros((3,3))
    homography[0] = h[0:3,0]
    homography[1] = h[3:6,0]
    homography[2][0:2] = h[6:8,0]
    homography[2][2] = 1

    return homography


def projection(image_src,image_target,pt,Homography):
    # initialize pixel as black
    temp = np.zeros((image_src.shape[0],image_src.shape[1],3), dtype='uint8')
    pts = np.array([[pt[0][1], pt[0][0]], [pt[1][1],pt[1][0]],\
    [pt[3][1], pt[3][0]], [pt[2][1], pt[2][0]]])
    # make the target area white
    cv2.fillPoly(temp,[pts],(255,255,255))

    for i in range(0,(image_src.shape[0]-1)):
        for j in range(0,(image_src.shape[1]-1)):
            if temp[i,j,1] == 255 and temp[i,j,0] == 255 and \
            temp[i,j,2]==255:
```

```
                point_tmp = np.array([i, j, 1])
                trans_coord = np.array(np.dot(Homography,point_tmp))
                # rescale then
                trans_coord = trans_coord/trans_coord[2]
                if (trans_coord[0] > 0) and (trans_coord[0] < image_target.shape[0]) and \
                (trans_coord[1] > 0) and (trans_coord[1] < image_target.shape[1]):
                    image_src[i][j] = image_target[math.floor(trans_coord[0]),math.floor(trans_coord[1])
            else:
                continue

    return image_src


def projection2(image_src,image_target,pt,Homography):
    temp = np.zeros((image_src.shape[0],image_src.shape[1],3), dtype='uint8')

    for i in range(0,(image_src.shape[0]-1)):
        for j in range(0,(image_src.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] < image_target.shape[0]) and \
            (trans_coord[1] > 0) and (trans_coord[1] < image_target.shape[1]):
                temp[i][j] = image_target[math.floor(trans_coord[0]),math.floor(trans_coord[1])]

    return temp


def write_img(pt1,pt2,output_file,input_img,imgJ):
    # Transforming the image in 1d to 1abc
    H = Homography(pt1, pt2)
    output = projection(input_img, imgJ, pt1, H)
    cv2.imwrite(output_file, output)


def write_img_a2c():
    # Transforming the image from a to c
    H_ab = Homography(Points_1b, Points_1a)
    H_bc = Homography(Points_1c, Points_1b)
    H_ac = np.matmul(H_ab, H_bc)

    Points_13_ = np.array([[0,0],[0, img3.shape[0]],[img3.shape[1],0],[img3.shape[1],img3.shape[0]]])
    output13_ = projection2(img3,img1,Points_13_,H_ac)
    cv2.imwrite('../PicsHw2/myimage13.jpg',output13_)


def main():
    write_img(Points_1a,Points_1d,'../PicsHw2/myimage1.jpg',img1,myimg)
    write_img(Points_1b,Points_1d,'../PicsHw2/myimage2.jpg',img2,myimg)
    write_img(Points_1c,Points_1d,'../PicsHw2/myimage3.jpg',img3,myimg)
    write_img_a2c()


if __name__ == '__main__':
```

```
main()
```